Effective and Efficient Transformer Models for Sequential Recommendation

Aleksandr V. Petrov

SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE
OF
DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING SCIENCE

COLLEGE OF SCIENCE & ENGINEERING

JUNE 2025

To my parents,
my beloved wife Lilia and my daughters Elena and Olga

Abstract

In the last decade, advances in natural language processing have driven significant interest in Deep Learning-based Sequential Recommendation Systems, as user-item interaction sequences resemble word sequences in language models. In particular, the arrival of the Transformer architecture transformed the field of sequential recommendation. It allowed Transformer-based models, such as BERT4Rec and SASRec, to achieve state-of-the-art results on many sequential recommendation problems. However, while these Transformer-based models perform well on small-scale academic datasets, they face challenges in real-life applications due to scalability problems and the complexity of modern recommendation goals, which include beyond-accuracy goals such as recommendation diversity. In this thesis, we closely examine the sources of these limitations and propose generalisable solutions to enable Transformer-based models for large-scale, real-world deployments.

In particular, training sequential recommenders is problematic. Indeed, most recommendation datasets contain different sets of items, making the pre-training foundation models impossible and requiring training recommendation models from scratch for every new recommendation dataset. Long training is problematic because it increases running costs and causes delays in fresh data processing. In our reproducibility study, we find that practitioners often end up with underfit models due to the long training requirement. To tackle the long training problem, we propose Recency Sampling of Sequences (RSS), a novel training objective for sequential recommender systems that allows the achievement of strong results even when training time is limited. For example, on the MovieLens-20M dataset, RSS applied to the SASRec model can result in a 60% improvement in NDCG over a vanilla SASRec and a 16% improvement over a fully trained BERT4Rec model despite taking 93% less training time than BERT4Rec.

Another big challenge for Transformer-based Sequential Recommender Systems is a large catalogue of items that may be several orders of magnitudes larger when compared to the vocabularies of items. Large catalogues create the need for negative sampling during training, but in this thesis, we show that negative sampling causes effectiveness degradation. To mitigate this problem, we design a new gBCE loss, which counters the effects of negative sampling by down-weighting the contribution of the positive sample in the overall cost. We show that gBCE allows for state-of-the-art effectiveness with large catalogues, even with retaining negative sampling.

A large catalogue also makes the item embedding tensor large and model inference slow, as sequence embedding is multiplied by this large embedding tensor. On the large-scale Gowalla dataset, where training non-sampled models is infeasible due to large catalogue size, we obtain substantial improvements by enhancing SASRec with gBCE loss (+47%). We also reduce the memory footprint and speed up model inference using our proposed RecJPQ technique that atomic item IDs into compact compositional sub-item ID representation.

Building upon RecJPQ's sub-item representations, we also address the problem of slow model inference with large catalogues. In particular, we propose two algorithms for fast item scoring. First, we propose the PQTopK algorithm, which computes item scores as the sum of the sub-item scores. Sub-item scores can be pre-computed and re-used between items, which results in up to 4.5$\times$ faster item scoring when compared to regular Transformer's scoring. We further observe similarities between RecJPQ sub-item representation and bag-of-words representations in Information Retrieval (IR). In IR, the problem of fast-scoring large collections of documents has been addressed using Dynamic Pruning approaches that allow finding Top-K items without scoring the whole catalogue exhaustively. Building upon the similarities between item representations in RecJPQ and document representations in IR, we propose the RecJPQPrune dynamic pruning algorithm for the RecJPQ-based recommenders. RecJPQPrune further improves scoring up time to 5.3$\times$ compared to PQTopK and up to 64$\times$ compared to regular Transformer's scoring.

Finally, while existing Transformer-based models perform well when measured using accuracy-based ranking metrics (e.g. NDCG), they usually struggle to optimise more complex goals, such as increasing diversity or promoting popularity bias. To improve model effectiveness on these complex beyond-accuracy goals, we propose an autoregressive Next-K recommendation strategy as an alternative to the traditional "score-and-rank approach". We also propose a universal reinforcement learning-based alignment scheme for the Next-K strategy and show that it is possible to align a generative recommendation model with beyond-accuracy goals, such as diversity promotion. Our experiments on two datasets show that in 3 out of 4 cases, GPTRec's Next-K generation approach offers a better tradeoff between accuracy and secondary metrics than classic greedy re-ranking techniques for diversity optimisation and decreasing popularity bias.

Contents

List of Tables

List of Figures

Acknowledgements

First, I am deeply grateful to my family—my beloved wife, Liliia, and my daughters, Elena and Olga. Without their unwavering support, I would never have begun this thesis. I am also deeply grateful to my parents, who have always been a source of encouragement throughout my PhD journey.

I am especially thankful to my principal supervisor, Craig Macdonald, for teaching me how to be a scientist and instilling in me a strong research culture. I am also grateful to my second supervisor, Iadh Ounis, for his valuable guidance and support throughout my PhD.

I would also like to extend my deep appreciation to my co-authors – Sean MacAvenue, Nicola Tonellotto, Tommaso Di Noia, Vito Walter Anelli, Davide Abbattista, Ildar Safilo, Daria Tikhonovich, Dmitry Ignatov, Efi Karra, Sean Murphy, Krishna Acharya, David Wardrope, Timos Korres, and Anders Uhrenholt. Collaborating with them has broadened my perspective on Recommender Systems and Information Retrieval and significantly enriched my research experience.

Last but not least, I would like to express my deep gratitude to my fellow Terrier group members, with whom I shared an office and many engaging lunchtime discussions, including Andreas Chari, Jack McKenchie, James Nurdin, Andrew Parry, Fangzheng Tian, Jinyuan Fang, Lubingzhi Guo, Shen Dong, Thomas Janich, Zeyang Meng, Zixuan Yi, Hitarth Narvala, Sarawoot Kongyoung, Xiao Wang, Siwei Liu, Xinhao Yi, Ting Su, and Javier Sanz-Cruzado Puig. Our conversations were among the most enjoyable aspects of my PhD journey.

Declaration

I declare that, except where explicit reference is made to the contribution of others, that this dissertation is the result of my own work and has not been submitted for any other degree at the University of Glasgow or any other institution

---

Aleksandr V. Petrov

Chapter 1

Introduction

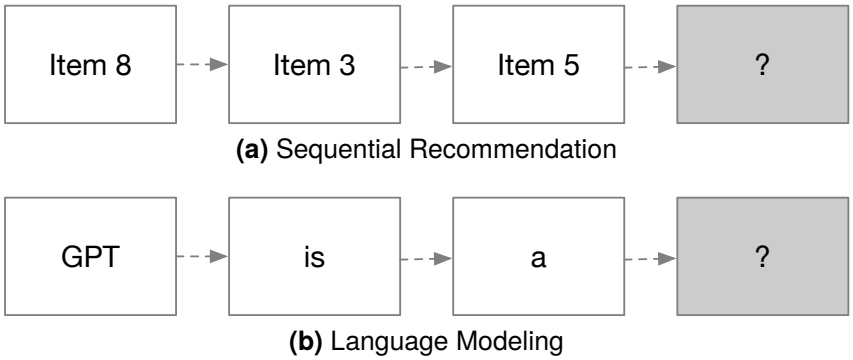
1.1 Motivation and Thesis Statement

We are living in the digital age, where people spend a significant part of their time online, making countless choices: what products to buy, what music to listen to, what movie to watch, where to go on holiday and so on. Frequently, Internet users do not know what options are available and require help finding the items that fit their preferences. *Recommender Systems* are computer systems that help users to find items of their preferences. Recommender Systems analyse users' behaviour patterns to predict what items a particular user is likely to prefer.

Recommender Systems have been researched since the early 1990s: the notion of Recommender Systems was first introduced in the paper "An algebra for recommendations," published in 1990 by Jussi Karlgren [101]; other notable Recommender Systems-related publications of the 1990s include the Tapestry work [62] that created the term "collaborative filtering", and the GroupLens paper [202] that introduced one of the first Neighbourhood-based approaches for recommendation. However, the research field of Recommender Systems remained relatively small until the mid-2000s when it was popularised by Netflix Prize [13], a competition organised by Netflix to predict users' movie preferences. Netflix Prize not only popularised Recommender Systems as a problem but also popularised *Matrix Factorisation* as the most common solution to this problem. First introduced in Funk's blog post [57], Matrix Factorisation was then used as part of the solution that won the Netflix Prize [113] and become a dominant approach in Recommender Systems for over a decade.

The main idea of Matrix Factorisation is to build a user-item interaction matrix, where rows represent users, columns represent items, and cells represent interactions (e.g. one if the user interacted with an item and zero if not interacted). Matrix Factorisation then approximates this matrix as a product of lower-rank matrices. The unobserved item scores from this approximated matrix are then used as item score predictions for a user. There are a number of recommendation approaches that build upon Matrix Factorisation ideas; the most popular among which include Bayesian Personalised Rank (BPR) [199] and Alternating Least Squares (ALS) [235].

Despite dominating the field of Recommender Systems research, Matrix Factorisation approaches have a number of limitations, one of the most significant of which is that MF doesn't account for the order of the interactions. However, the order of interactions is important in many real-life scenarios, in particular:



(a) Sequential Recommendation

(b) Language Modeling

Figure 1.1: The next item prediction task used in Sequential Recommendation models and the next token prediction task used in language models are essentially the same.

1. Making recommendations when there are natural sequential patterns (e.g. buying a case for a phone after buying a phone, but not the other way around);
2. Taking into account evolving user interests (the fact that a user listened to rock music recently may be more important for a Recommender System than listening to pop music 10 years ago);
3. Taking into account the geographical proximity of recommended items (e.g. it makes sense to recommend the next city on a trip that is located close to the last visited city);
4. Recommending repeated items (classical Matrix Factorisation assumes that the user interacts only once with each item and therefore does not recommend items repeatedly).

Some of these limitations were addressed within the Matrix Factorisation paradigm. For example, the TimeSVD++ [111] approach adds temporal dynamics into Matrix Factorisation; However, later, a novel paradigm of *Sequential Recommendation* [190] has emerged to address the limitations of traditional Matrix Factorisation-based methods. Sequential Recommender Systems are the key focus of this Thesis.

Sequential Recommender Systems use ordered sequences of user-item interactions to predict future user interactions. Compared to more traditional Matrix Factorisation approaches [114, 199, 235], these approaches can consider sequential patterns in data, model evolving user interests and recommend repeating items. Early Sequential Recommender Systems were based on Markov Chains [289] and extensions of Matrix Factorisation-based methods [200]. However, since the arrival of GRU4Rec model [81], the most advanced Sequential Recommender Systems have been based on deep neural networks [80, 236, 270]. In particular, the state-of-the-art Sequential Recommendation models, such as SASRec [100], BERT4Rec [230] are based on the Transformer [247] deep learning architecture, which was originally designed for the natural language processing domain.

The high effectiveness of Transformer-based models for Sequential Recommendation is not surprising. Indeed, the Sequential Recommendation problem is usually cast as the *next item prediction* task, where the goal of a Recommender System is to predict what item will come next in the sequence of User-Item interactions. As illustrated in Figure 1.1, this task is essentially the same as the next token prediction task that is used in many language models, such as GPT-2 [192]. Since the end of the 2010s, Transformer-based models have dominated the field of language modelling; hence, it is reasonable that the same architecture works well for the same task in a different domain. Hence, to adapt the Transformer architecture for the Sequential Recommendation problem, researchers use item IDs in the sequences instead of tokens in the sentences. However, as this Thesis will argue, this mapping has a number of limitations that make usage of Transformers problematic in real-world applications.

First, Transformer-based models require a lot of computational resources to train. For example, our experiments with the BERT4Rec model show that training the original implementation of the BERT4Rec model on a relatively small MovieLens-1M dataset may require up to 20 hours on a modern GPU in order to reproduce results reported in the original paper [230] (see details in Chapter 3). In the Natural Language Processing domain, this large training time is less problematic because, usually, Transformer models are pre-trained once on a large corpus of texts and then fine-tuned for every specific task. However, this pre-training/fine-tuning approach doesn't work well for Recommender Systems: most recommendation datasets contain a unique set of items (in contrast to the sets of tokens being the same in different Natural Language Processing tasks), so the models trained on different sets of items are incompatible with each other. Therefore, we have to train a new instance of a model for every new dataset from scratch.

Second, the number of items in an online platform may be several orders of magnitude larger than the vocabulary size of a language model. For example, a popular video-on-demand platform YouTube has more than 1 Billion videos available[1], whereas most of the language Transformer models have much smaller vocabularies; for example BERT [46] has vocabulary of just 30,000 tokens. During both training and inference, BERT calculates a score distribution across all tokens in the vocabulary, and therefore, applying the BERT model directly to a Recommender System with a large number of items may be prohibitively costly. Indeed, a BERT-based recommendation model BERT4Rec [230] was originally applied only to relatively small datasets with no more than 55000 items. We argue that a large catalogue with millions of items makes training poses a number of challenges, including (i) challenging training of a Transformer, as a direct adaptation of language model methods would require computing softmax over the whole catalogue at each

1. https://earthweb.com/how-many-videos-are-on-youtube/

training step; (ii) expensive memory requirement for storing item embeddings; and (iii) slow inference of the model due to need to score millions of items. Indeed, our experiments show that direct adaptations of language models are prohibitively costly with catalogues containing more than 1 million items [176].

Finally, while existing generation of Transformer-based models achieves state-of-the-art results for *ranking accuracy*, typically measured such metrics as NDCG or Recall@K, many researchers recently argued that a good Recommender System should also optimise for beyond-accuracy goals [3, 59, 103], such as diversity or novelty. Optimising for beyond-accuracy goals remains unsolved, and most existing solutions rely on greedy reranking techniques, such as Maximal Marginal Reranking [25] or Serendipity Oriented Greedy [116].

Despite these challenges, the rapid progress in language processing models, information retrieval, and machine learning suggests that there can be a path to resolving the scalability and beyond-accuracy effectiveness issues. For example, Clark et al. [34] showed that choosing an appropriate training objective may significantly reduce required computations for training a high-quality Transformer model. Jean et al. [92] showed that language models with a large vocabulary can be trained effectively and efficiently when using a *Negative Sampling* technique together with an adjusted loss function. Zhan et al. [273] demonstrated that the memory footprint of Transformer-based retrieval models can be reduced by using *Joint Product Quantisation* (JPQ) technique. Ouyang et al. [165] proposed to use *autoregressive generation* optimised with Reinforcement Learning-based techniques for tuning the generative Transformer models for intricate goals. Additionally, there are methods from the pre-Transformer era that may be applicable to improving Transformer efficiency. For example, *dynamic pruning* methods [16, 240, 246] are used in search engines to short-circuit the scoring of documents that cannot make the final top-K ranking - we hypothesise that similar methods can improve model inference efficiency.

In summary, although Transformer-based Sequential Recommenders, such as BERT4Rec [230], exhibit good ranking accuracy, these models' existing training and inference methods are inefficient. These methods don't scale to larger datasets, which are common in real-world applications. In addition, current training and inference methods focus on ranking accuracy but not on beyond-accuracy goals, such as diversity. However, the progress in the adjacent domains of Language Modelling and Information Retrieval suggests some mechanisms to resolve these issues. Therefore, efficient training and inference of Transformer models for large-scale Sequential Recommendations require further research, which is the focus of our Thesis. This leads us to the following Thesis Statement:

Thesis Statement

This Thesis states that Transformer-based models can be used for large-scale Sequential Recommender Systems efficiently and effectively for both training and inference, even when effectiveness includes beyond-accuracy objectives. In particular, we can make the training of a Transformer-based model more efficient with recency-based sampling of training sequences. Moreover, we can scale Transformer models to millions of items using negative sampling coupled with an improved loss function to avoid unnecessary computations and, therefore, improve training efficiency without degrading inference effectiveness. Additionally, we can reduce the memory footprint and speed up the inference of the model using quantisation techniques for item embeddings, which can also help scale the model inference to millions of items using dynamic pruning techniques. Finally, we state that we can make Transformer-based models effective, even when effectiveness includes beyond-accuracy goals, such as increasing diversity or decreasing popularity bias, by using a generative approach for Sequential Recommendation and using reinforcement learning to align the model with these goals.

To structure our research toward the posed Thesis Statement, we divided it into a number of smaller research projects. Each of these smaller projects resulted in a research paper. We describe these papers in the next section.

1.2 Origins of The Materials

Most of the material presented in this Thesis is based on papers published in journals and conferences throughout this PhD programme:

1. In reproducibility paper [174] we establish the current state-of-the-art in Sequential Recommendation. We perform a systematic review and replicability study of BERT4Rec [230] and find that it indeed achieves state-of-the-art performance even when compared to the most recent reported results. The study was published as a full paper in the reproducibility track of ACM RecSys'22. The paper contributes to Chapter 3.

2. In the research paper [176], we analyse performed the study of existing training objectives and propose a novel objective called "Recency Sampling of Sequences (RSS)", which uses ideas from both Items Masking and Sequence Shifting. We show that RSS is more effective and efficient than the existing objectives. The study was published as a full research paper at the ACM RecSys'22 conference and was nominated for the Best Student Paper award. The paper contributes to Chapter 4.
3. In addition, in the extended journal paper [181], we further analyse *why* RSS training objective works through an extensive analysis of the similarity matrices of the positional embeddings. The paper was published as an invited publication in the special issue "Highlights of RecSys'22" of the ACM Transactions on Recommender Systems journal. The paper contributes to Chapter 4.
4. In the research paper [178], we analyse the problem of negative sampling in Sequential Recommendation models. We find that negative sampling causes the *overconfidence* in the models (the models tend to overestimate probabilities of items being positive). We then propose gBCE – a novel loss function that can mitigate overconfidence and improve the effectiveness of the models. The paper was published as a full research paper at the ACM RecSys'23 conference, where it received the Best Paper award. An extended abstract [179] of the paper was also published in the "Best Papers from Sister Conferences" track of the IJCAI'24 conference. The paper contributes to Chapter 5.
5. In the extended journal paper [184], we further analyse the properties of the gBCE loss. We show that gBCE not only improves the effectiveness of the trained models but also improves models *calibration* – the ability of the model to predict actual interaction probabilities. In addition, we show that gBCE is effective not only for Sequential Recommendation but also for a broad range of recommendation and information retrieval tasks. The paper was published as an invited publication in the special issue "Highlights of RecSys'23" of the ACM Transactions on Recommender Systems journal. The paper contributes to Chapter 5.
6. In the research paper [180], we tackle the problem of the large embedding tensor in Sequential Recommendation models. We show that it is possible to reduce the size of the Sequential Recommendation models while keeping their effectiveness using our novel RecJPQ technique. The paper was published as a full research paper at the ACM WSDM'24 conference. The paper contributes to Chapter 6.
7. In a short research paper [182], we further show that RecJPQ can improve the inference efficiency of the Sequential Recommendation models, allowing for deployments in real-world scenarios with millions of systems. We propose PQTopK, an algorithm that exploits the salient characteristics of RecJPQ sub-item representation to achieve efficiency gains. The paper was published as a short research paper at the ACM RecSys'24 conference. The paper contributes to Chapter 7.

8. In a full research paper [172], we further improve inference efficiency by showing that it is possible to find the highest-scored items without scoring all items exhaustively. We Propose RecJPQPrune, a dynamic pruning-based algorithm for RecJPQ-based models that is even more effective than PQTopK for large-catalogue deployments. The paper is accepted for publication in the proceedings of the ACM SIGIR'25 conference and also contributes to Chapter 7.
9. In the workshop paper [177] we analyse the feasibility of *generative* approach for Sequential Recommendation. We propose an autoregressive Next-K generation strategy as an alternative to the traditional Top-K recommendation and show that the model remains efficient. The paper was presented at the GenIR workshop at the ACM SIGIR'23 conference. The paper contributes to Chapter 8.
10. In the workshop paper [175], we further analyse the autoregressive Next-K Generation strategy. In this paper, we specifically focus on the applications of generative models for beyond-accuracy goals. We show that generative models can be aligned with beyond-accuracy goals using reinforcement learning. The paper was presented at the Generative Recommendations workshop at the ACM WWW'24 conference. Together these two papers contribute to Chapter 8.

We now describe how the contributions originally presented in these papers are organised within the Thesis.

1.3 Contributions & Outline

This Thesis contributes a number of improvements to existing Transformer-based Sequential Recommendation models that make these models effective and efficient in real-world scenarios. Most of our proposed improvements are generalisable and can be used with a broad class of models and datasets. In particular, this thesis contributes:

1. *Recency Sampling of Sequences (RSS)*; a novel training objective for Sequential Recommendation systems that enables training effective models within limited training time (described in Chapter 4);
2. *Generalised Binary Cross-Entropy (gBCE)*; a novel loss function for Sequential Recommender Systems that counters undesirable effects of negative sampling. The thesis also contributes gSASRec and gBERT4Rec, Sequential Recommender models based on SASRec and BERT4REc for large-catalogue systems that use gBCE (described in Chapter 5).

3. *Joint Product Quantisation for Recommender Systems (RecJPQ)*; a novel method for compressing item embedding tensors for large-catalogue Sequential Recommender Systems (described in Chapter 6);
4. *PQTopK*; an efficient inference method for RecJPQ-based Recommender Systems (described in Chapter 7);
5. *RecJPQPrune*; another efficient inference method for RecJPQ-based models that avoids exhaustive scoring, and hence more efficient than PQTopK with large-catalogue systems (described in Chapter 7);
6. *Next-K*; an autoregressive recommendation generation strategy for beyond-accuracy goals, as well as GPTRec, a Sequential Recommendation model that uses Next-K generation and can be aligned with complex recommendation goals (described in Chapter 8).

Overall, this thesis has nine chapters, including the Introduction and the Background chapters, six methodological chapters based on the published papers and the Conclusions chapter. The detailed outline of this Thesis is as follows:

- *Chapter 1* contains an introduction to the problem, Thesis Statement, Thesis contributions, outline and origins.
- *Chapter 2* contains a background on Sequential Recommender systems and Transformer models as well as introduces information relevant to other chapters, such as common experimental details (datasets, metrics, splitting strategies, implementation frameworks).
- *Chapter 3* probes the state-of-the-art in Sequential Recommendation. It shows that BERT4Rec exhibits state-of-the-art effectiveness when fully converged.
- *Chapter 4* analyses training objectives for Sequential Recommender systems, finds that existing training objectives have efficiency or effectiveness limitations, and proposes RSS, a novel training objective that is effective and efficient simultaneously.
- *Chapter 5* analyses the effects of negative sampling in recommender systems and finds that negative sampling is unavoidable in large-scale scenarios; however, it also finds that negative sampling leads to a negative *overconfidence* problem, which in turn hinders the model's effectiveness. The Chapter proposes the gBCE loss function, which successfully mitigates the undesirable effects of negative sampling.
- *Chapter 6* shows that in large-catalogue scenarios, there is a need for item embedding compression, a proposes RecJPQ, a novel method that allows to achieve compression without hindering effectiveness.
- *Chapter 7* argues that with large-catalogue scenarios, model inference becomes slow and expensive and proposes PQTopK and RecJPQPrune, two efficient inference methods for RecJPQ-based models.

- *Chapter 8* shows that for beyond-accuracy recommendation goals, the traditional "score-and-rank" Top-K recommendation approach is not sufficient and proposes the autoregressive Next-K recommendation strategy that can be successfully aligned with such recommendation goals as increased diversity or decreased popularity bias.
- *Chapter 9* contains concluding remarks and suggestions for future work.

We now turn to Chapter 2, where we cover the necessary background on Sequential Recommendation and Transformer models.

Chapter 2

From Matrix Completion to Transformers

Foundations, Limitations, and Evaluation of Sequential Recommender Systems

This Chapter provides an overview of the background and related work for the thesis. We start with a brief overview of the Recommender Systems problem space in Section 2.1. In Section 2.2, we specifically cover the Sequential Recommendation problem, which is the main focus of this thesis. In Section 2.3, we discuss Transformer-based models as state-of-the-art solutions for Sequential Recommendation and their limitations, which we address in the thesis. Section 2.4 contains the description of the experimental evaluation framework that is common across the methodological chapters of this thesis. Section 2.5 summarises the background and the research gaps.

2.1 Foundations of Recommender Systems

Recommender Systems are computer software systems that provide users with the items of their interest [203]. These systems can be seen as intelligent agents that make decisions about what items to return to every individual user. There is a great variety of ways these systems can make decisions, from the most simple systems that provide all users with the same pre-defined recommendations (e.g. editorial selection on news websites) to the most advanced systems that analyse lots of details about users, items and current context and make decisions using the most advanced methods of Artificial Intelligence. Given this broad spectrum of approaches within Recommender Systems, our exploration begins with the foundational Matrix Completion methods, which, despite being surpassed by more advanced techniques in popularity since the mid-2010s due to advances in Deep Learning, remain invaluable in specific contexts, such as serving as candidate generators in multi-stage pipelines. However, as we show in Section 2.1.4, Matrix Completion methods have a number of limitations, such as failure to model evolving user interests, which can addressed with the *Sequential* methods that we discuss later in this Chapter.

2.1.1 Early Matrix Completion Methods

Since the earliest studies on Recommender Systems [101], researchers have noted that analysing user's interactions with items can help predict future interactions and, in turn, generate useful recommendations. For example, suppose that most of the users who watched "The Matrix" movie also watched "The Terminator" movie. In that case, it makes sense to recommend "The Terminator" to a user who has only watched "The Matrix" but hasn't watched "The Terminator" yet. Note that this decision does not require any extra information about the user or the items; it only

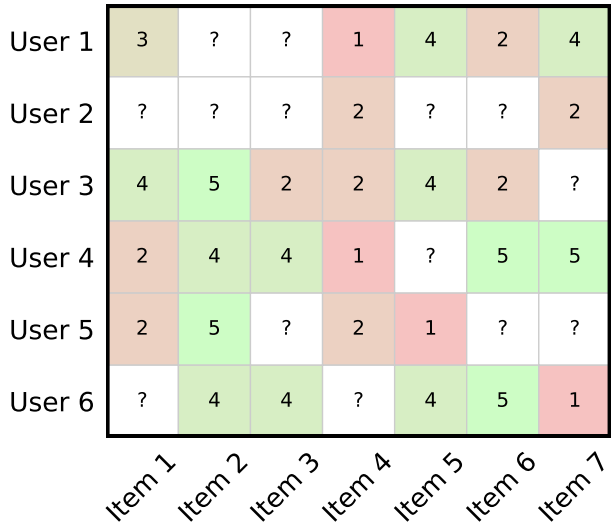


Figure 2.1: Matrix Completion Problem. In the Matrix Completion problem, the goal of a Recommender System is to reconstruct the partially filled user-item rating matrix, where each row corresponds to a user, and each column corresponds to an item. The entries represent ratings, which can be explicit (e.g., a numerical score) or implicit (e.g., a click or view). The question marks correspond to unknown or missing ratings that the recommender system aims to predict. The aim is to fill in these missing entries in a way that is consistent with the known data and can generalise well to future user-item interactions.

requires a dataset of existing user-item interactions from which the system can mine the pattern. The approaches that only rely on user-item interactions and do not require any extra information (such as item descriptions) are called *collaborative filtering* methods. The term *collaborative filtering* was coined in the early Tapestry project by Xerox Research [62], although the authors used it with a slightly different meaning: indeed, it was a mail list *filtering tool* and not a Recommender System; however, it had the same spirit as modern Recommender Systems, as users helped each other to find interesting mailing lists. Hence, Tapestry can be seen as a precursor to modern-day collaborative filtering Recommender Systems.

Most of the early collaborative filtering-based methods, such as GroupLens [202], relied primarily on explicit user ratings and computed similarity between users (user-based CF) or between items (item-based CF). The user-based CF approach follows these steps:

1. Find the behaviour similarity between a user and all other users in the system (e.g., using Pearson correlation or using cosine similarity);
2. Select N the most similar users according to the chosen similarity measure;
3. Select items that these most similar users rated as *candidate items*;
4. Compute scores for candidate items as a weighted sum of scores that the selected neighbouring users gave to these items (the weight depends on the similarity between users);

5. Seturn Top-K recommendations ranked according to these scores.

Item-based collaborative filtering follows a similar approach, but instead of identifying similar users, it finds items that are most similar to those in the user's history. In this case, similarities are computed between items (rather than users), and recommendations are generated using a weighted sum of ratings from similar items, with the weights determined by item similarity. Key item-based methods include work [211] by GroupLens research group and paper [139] by Amazon.

Despite the majority of work being focused on collaborative recommendation, some early researchers also explored *content-based* and *hybrid* approaches [10]. In content-based methods, a Recommender System mostly relies on content associated with the items (e.g., textual descriptions), while hybrid methods combine both content and collaborative signals. While content-based and hybrid methods are important research directions, they are orthogonal to this thesis. Indeed, most of the methodology proposed in this thesis is generic and can be applied to different backbone methods, including content-based and hybrid methods.

Early works methods set the paradigm for recommender systems as the *Matrix Completion* problem: in this paradigm, the goal of a Recommender System the main goal of a recommender system is to fill gaps in the user-item interactions matrix in a way that is consistent with past user-item interaction and can generalise to the future interactions, as illustrated by Figure 2.1. While neighbourhood-based methods are an initial solution to this problem, *Matrix Factorisation*-based methods soon outperformed the early neighbourhood-based methods. We now discuss Matrix Factorisation-based methods in more detail.

2.1.2 Matrix Factorisation

As we discussed in Section 2.1.1, many early recommendation approaches have seen recommender systems as the Matrix Completion problem: the user-item interaction matrix contained observed ratings, and the goal of the system is to predict the values of the ratings that the users are likely to give to the unobserved items. The paradigm of a Recommender System as a Matrix Completion problem was solidified by the famous Netflix Prize [13] competition that ran between 2006 and 2009. For the competition, Netflix not only released one of the largest rating prediction datasets but also offered 1 million US dollars to the authors of the winning solution. The Netflix Prize popularised the *Matrix Factorisation* (MF) family of approaches that domin-

ated the field of Recommender Systems for many years. Indeed, Matrix Factorisation worked exceptionally well for the Nextflix Prize rating predictions dataset and was used as a part of the winning solution. In this Section, we briefly discuss the basics of Matrix Factorisations and the key methods based on the MF ideas.

Let's denote a given user-item interaction matrix as $M \in \mathbb{R}^{|U|\times|I|}$, where U is the set of users and I is the set of items. The interaction matrix M may contain *explicit* ratings given by users to items or *implicit* (in that case, for example, it has the value of "1" in the cells of the matrix where the user has interacted with items).

Figure 2.2 illustrates the main idea of Matrix Factorisation approaches. As can be seen from the Figure, Matrix Factorisation approximates matrix M as the product of two lower dimensions matrices, matrix $V \in \mathbb{R}^{|U|\times d}$ that contains user representations (aka. *user embeddings*) and matrix $E^T; E \in \mathbb{R}^{|I|\times d}$ that contains item representations (*item embeddings*)[1]. A hyperparameter d is the number of *latent features* (i.e. the dimensionality of the embeddings associated with each item and each user). Overall, matrix factorisation can be formalised as:

$$M \approxeq V \times E^T = R \tag{2.1}$$

where R is the approximation of matrix M obtained from the factorisation. The embedding dimensionality d is usually chosen much smaller than the number of users and the number of items, i.e.,

$$d \ll \min(|I|, |U|)$$

Hence, usually R is a lower rank matrix than M (rank of R is bounded by d whereas rank of M is bounded by $\min(|I|, |U|)$). Therefore, due to approximation, R contains non-zero value values for unobserved user-item interactions. The main idea of matrix factorisation is that if the value associated with an unobserved user-item interaction is high, then the user is likely to be interested in the item (and vice-versa, not interested in items where their reconstructed Matrix has a low value).

To return recommendations for a user, Matrix Factorisation methods sort reconstructed ratings for unobserved items from the reconstructed matrix R and output the list of K highest-scored items as a result.

---

1. In the literature, these low dimensional matrices are frequently called U and V [234], or U and Q [37]. However, we use U to denote the set of users; hence, we use V for the user embedding matrix. We also use E for the item embeddings to highlight the equivalence with the item embeddings used by Sequential Recommendation models, which we will discuss in Section 2.2

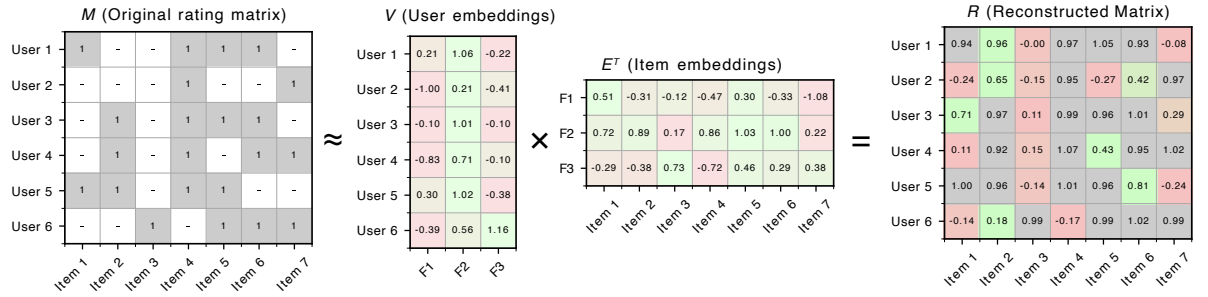


Figure 2.2: Matrix Factorisation. Grey corresponds to observed interactions.

To find matrices V and E, matrix factorisation approaches typically use versions of gradient descent to minimise the reconstruction error (for example, many use the rooted mean squared error (RMSE) as the reconstruction error measure).

There is a large number of Matrix Factorisation-based approaches. Some of the most notable approaches include:

- **Funk SVD** [57], is the first MF-based approach, which was introduced in Funk's blog post regarding his progress on the Netflix Prize.
- SVD++ [112] uses both explicit and implicit interactions, as well as added biases for uses and items.
- **PureSVD**, also known as **Truncated SVD** [37], applies Singular Value Decomposition (SVD) to the user-item interaction matrix M, decomposing it into three matrices: $M \approx U \times \Sigma \times E$ where Σ is a diagonal matrix containing the d largest singular values, and V and E contain the corresponding singular vectors. Unlike **Funk SVD** and **SVD++**, which approximate the interaction matrix using two-factor representations with bias terms, PureSVD follows the traditional SVD decomposition into three matrices. Truncated SVD is often used for dimensionality reduction by selecting only the top d singular values, capturing the most important latent structure of the interaction data – we utilise this property for our proposed RecJPQ approach in Chapter 6.
- **Bayesian Personalised Rank (BPR)** [199] introduces a ranking-based objective function for recommendation, which is better suited for real-life scenarios than rating-based objectives such as RMSE.
- **Factorisation Machines** [196] generalise Matrix Factorisation and allow the addition of other important information beyond user-item interactions (e.g. item categories) into consideration.

- **Alternating Least Squares (ALS)** [235] enables efficient computations with large matrices by alternating updates of matrices V and E. Differently from other methods, ALS is not a gradient-based solution; instead, it efficiently finds a closed-form solution for the optimal update in each step, allowing for efficiency gains compared with gradient-based methods.

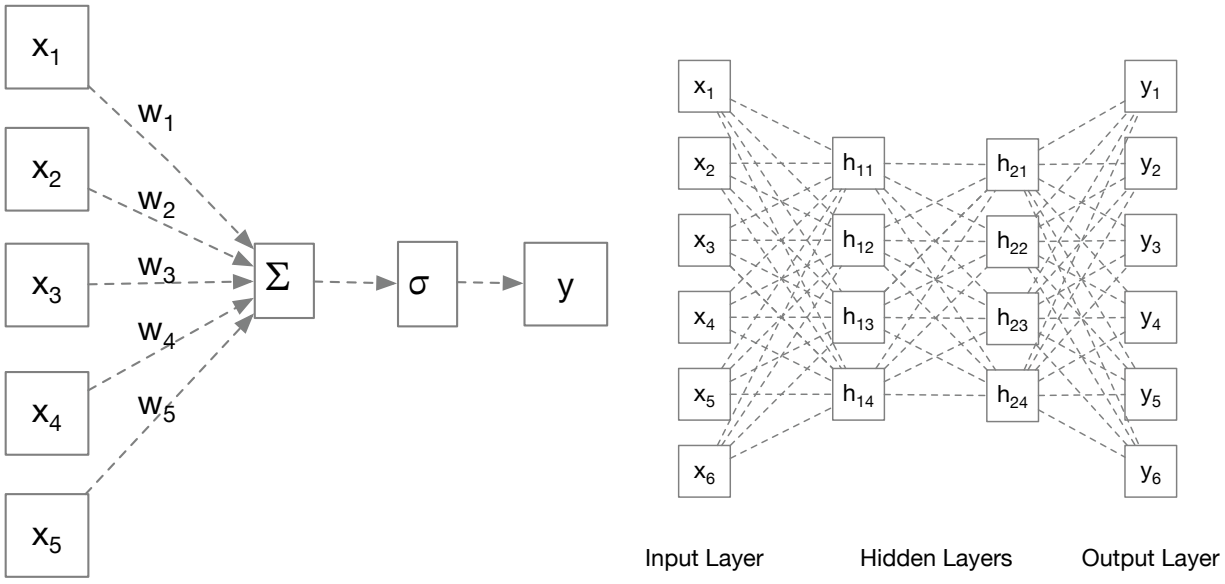
This list is far from being exhaustive. Matrix Factorisation-based methods are still in active use and remain competitive [55, 157, 201] in traditional Matrix Completion settings even after almost two decades since their introduction. However, later *Deep Learning*-based methods outperformed traditional Matrix Factorisation. In the next Section, we briefly discuss applications of the Deep Learning-based method for recommender systems.

2.1.3 Deep Learning for Traditional Recommender Systems

In the middle of the 20th century, *Neural Networks* were one of the earliest methods for building intelligent systems [154, 208]. However, until the early 2010s, Neural Networks capabilities were limited because of the lack of large training datasets and expensive computation requirements. Both problems were solved in the 2000s: the popularity of the Internet allowed gathering very large training datasets, and the progress in Graphics Processing Units (GPU) hardware made the computations cheaper. Hence, in the 2010s, Neural Networks became very popular and achieved state-of-the-art performance in many fields, including Natural Language Processing [46, 192, 247], Computer Vision [109, 118], Information Retrieval [105, 163] and Recommender Systems [81, 100, 134, 230]. Most of the methodology described in this thesis is based on Neural Networks. Since the mid-2010s, approaches that use Neural Networks have also been known as *Deep Learning*, as the term "deep" highlights the multi-layer structure of these networks.

A typical Neural Network consists of a number of interconnected *Artificial Neurons*. Figure 2.3 illustrates an Artificial Neuron (Fig. 2.3a) and a Neural Network (Fig. 2.3a). As we can see from Figure 2.3b a figure, an Artificial Neuron takes a vector $x = \langle x_1, x_2, ..x_n \rangle$ as an input, computes a weighted sum of the inputs with the weights defined by a vector of learnable parameters $w = \langle w_1, w_2, ..w_n \rangle$, and applies a nonlinear transformation σ. The output of this nonlinear transformation is the scalar output of the neuron y:

$$y = \sigma\left(\sum_{i=1}^{n} x_i w_i\right) = \sigma(x \cdot w)$$



(a) Neuron **(b)** Neural Network

Figure 2.3: An Artificial Neuron and a Neural Network

Typical choices for the nonlinear function transformation include the logistic sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ or the Rectified Linear Unit function $\text{ReLU}(x) = \max(0, x)$.

Figure 2.3b shows a simple Neural Network consisting of interconnected Artificial Neurons, where the output of some neurons becomes the input of others. These neurons are typically organised into multiple *layers*. The *Input Layer* receives an input vector x. A stack of *Hidden Layers* follows the Input Layer. At the end of the network, the *Output Layer* returns the output vector y. Unlike a single neuron, the output of a neural network is a vector rather than a scalar value.

In recommender systems, the input vector x, for example, may represent a user's observed interactions derived from a row of the user-item interaction matrix M. The network then computes the output vector y, where each entry corresponds to a predicted relevance score for an item in the catalogue.

Like Matrix Factorisation-based methods, Deep Learning-based models are usually trained iteratively using a version of gradient descent. This training can be highly efficient using Graphics Processing Unit (GPU) hardware and specialised accelerated computing libraries, such as TensorFlow [1]. There are multiple ways to design the architecture of a neural network. For details on different neural network architectures, we refer the reader to the foundational Deep Learning book [64, Chapters 6, 9, 10].

There are a number of Deep-Learning recommendation methods, most notably including:

- Generalisations of matrix factorisation methods [75, 265];
- Generalisations of factorisation machines [250, 251] for including side information;
- Variational Autoencoder models [134], that use variational inference for predictions;

Most importantly for this thesis, there are a number of Deep-learning-based recommendation methods that are based on Neural Language Models [50, 100, 230]. We discuss these models in detail in Section 2.3.

We also note that in the Matrix Completion settings, the non-neural methods remain competitive with the Deep Learning-based methods [55, 201]. For example, the famous reproducibility paper by Dacrema et al. [55] showed that simple linear methods, such as EASE [227] can outperform some Deep Learning-based methods, such as the Neural Collaborative Filtering [75]. However, the same paper [55] shows the advantages of the state-of-the-art Deep Learning-based methods, such as Mult-VAE [134] over non-neural methods. Indeed, most modern approaches for Matrix Completion are based on Deep Learning due to the flexibility and effectiveness of these methods.

All approaches we discussed so far were *Matrix Completion* recommendation approaches, meaning that these methods work with unordered user-item interaction. However, not taking the order of interactions into account results in a number of limitations, which we discuss next.

2.1.4 Limitations of Matrix Completion Methods

While the traditional Matrix Completion approaches enabled many practical applications, recommender systems practitioners soon realised that the Matrix Completion setting has a number of limitations, which are hard to address without reformulating the problem. In particular, we identify the following limitations of traditional approaches:

1. No account for natural sequential patterns in data. In many situations, there is a clear sequential (e.g. causal) relation between interactions; for example, after buying a phone, the users may tend to buy a case for the phone. Hence, it makes sense to recommend the case just after the user purchased the phone. However, with a traditional Matrix Completion setup, the recommender system doesn't know what the last purchase in the sequence was and, hence, can not effectively utilise these causal patterns.

2. No account for a series of items. For example, the set of users who watched the fourth Harry Potter movie is highly similar to the set of users who watched the first Harry Potter movie. Hence, for most Matrix Completion methods, it is natural to recommend the fourth movie to everyone who watched the first movie. However, it only makes sense to recommend the fourth movie to those who have already watched the second and the third movies.
3. No account for evolving user interests. For example, if a user listened to pop music ten years ago and recently switched to rock music, it is better to make recommendations based on the recent interest in rock. However, in the traditional Matrix Completion recommenders, every listening event would have the same influence on the recommendation result.

4. No account for geographical proximity with recent interactions. For example, consider a platform that recommends the next city to visit on a road trip. The next city to visit for somebody who follows the route Paris $\Rightarrow$ London $\Rightarrow$ Newcastle $\Rightarrow$ Edinburgh will be very different from somebody who follows the Edinburgh $\Rightarrow$ Newcastle $\Rightarrow$ London $\Rightarrow$ Paris. However, in the Matrix Completion recommenders, these two interaction sequences will correspond to the same representation in the user-item interaction matrix. As a result, these two users will receive the same sets of recommendations.
5. No repeated interactions. Traditional Matrix Completion models clearly separate "observed" and "unobserved" interactions and aim to predict the score for the "unobserved" part of the matrix. In reality, in many scenarios, the user-item interactions can repeat; for example, a user may listen to the same music many times.

There were attempts to solve these problems within the traditional Matrix Completion paradigm. For example, TimeSVD++ [111] added temporal dynamics into the SVD++ model and made user representation time-dependent, hence addressing some of the limitations. However, more generally, these limitations created a need for a new recommendation paradigm, which was developed in the form of Sequential Recommendation. We now discuss how Sequential Recommender systems consider the recommendation problem and how they address the limitations of Matrix Completion recommenders.

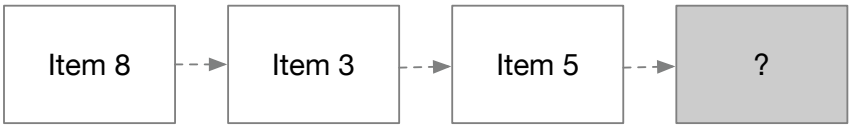


Figure 2.4: Next Item Prediction Problem. The goal of the Next Item Prediction problem (used by Sequential Recommender Systems) is to determine which item a user will interact with next, given a sequence of previously consumed items. In this figure, a user has engaged with items 8, 3, and 5 in succession, and the next item is unknown. This type of recommendation is also referred to as Sequential Recommendation. Unlike Matrix Completion (see Figure 2.1), which focuses on filling missing entries in a static user-item rating matrix, Next Item Prediction explicitly models the sequential aspects of user behaviour, making it a distinct recommendation paradigm.

2.2 Sequential Recommendation

Sequential Recommender Systems are a class of recommender systems that consider the order of the user-item interactions [190]. In contrast to traditional Matrix Completion-based models, such as Matrix Factorisation-based methods, these models account for the changes in individual user preferences, as well as global changes in item popularity, hence allowing the limitation of the Matrix Completion methods to be addressed. Differently from the traditional Matrix Completion task, the goal of a Sequential Recommender system is usually cast as to predict the next interaction in a *sequence* of user-item interactions. This task is also known as the *Next Item Prediction* problem, which Figure 2.4 also illustrates.

More formally, consider a set of users $u \in U$, where each user has made a sequence of interactions $s \in S = \{i_1, i_2, i_3 ... i_t\}$, and $i_k \in I$ denotes items ordered according to interaction time. The task of a Sequential Recommender System is to predict the next element i_{t+1} in the sequence s. In other words, the goal of the Sequential Recommender System M is to produce a list of K items ranked by the probability of being the sequence continuation:

$$M(s) \rightarrow \{i_{s_1}, i_{s_2}, i_{s_3}, i_{s_4} ... \, i_{s_K}\}$$

Early Sequential Recommender Systems used the Markov Chains [200, 289]: they statistically computed the conditional probability of an item appearing next in the sequence conditioned on a few previous items. One of the most popular Markov Chains-based approaches was the FPMC model [200]. FPMC employed the ideas from the traditional Matrix Factorisation approaches, but instead of factorising a 2-dimensional user-item matrix, it factorised a 3-dimensional (user, "from item", "to item") cube. Figure 2.5 shows an example of the 3d cube used by FPMC. An element of this cube corresponds to an empirical (counting-based) probability of the "to" item

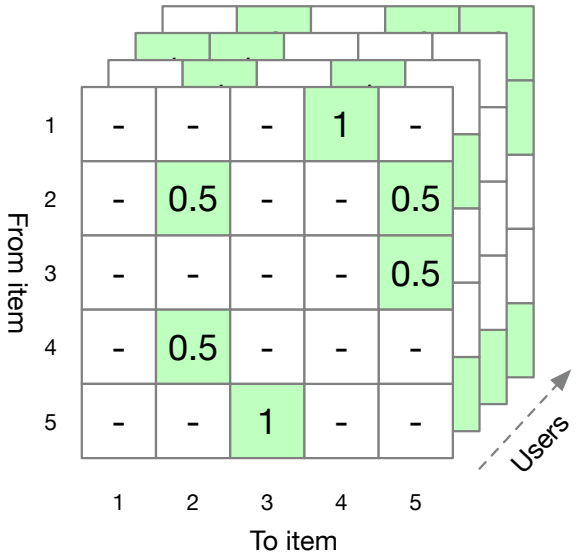


Figure 2.5: Transitions tensor in the FPMC approach. The Figure is adapted from [200].

following the "from" item for this particular user. By factorising this cube and recovering the probabilities for unobserved (user, "from item", "to item") triplets, the authors were able to improve their effectiveness over Matrix Factorisation-based methods by a large margin, particularly for sequential data.

Since the mid-2010s, the best Sequential Recommender Systems have been based on Deep Learning; we discuss these Deep Learning-based methods next.

2.2.1 Neural Sequential Models

In the following, we provide an overview of neural Sequential Recommendation models. Indeed, over the last several years, most of the Next Item prediction approaches have applied deep neural network models. Some of the first solutions based on deep neural networks were GRU4Rec [81] and the improved GRU4Rec$^+$ [80] (using an improved listwise loss function), both of which are based on Gated Recurrent Units (GRUs), a variant of Recurrent Neural Networks (RNNs). Figure 2.6 illustrates the architecture of GRU4Rec. As the Figure shows, the model receives a sequence of item ids as the input. It then uses the item embeddings layer to convert item ids to vector representations (i.e. a sequence of item ids is converted to a sequence of item embeddings). The Embeddings layer is then followed by a stack of Recurrent Neural Network

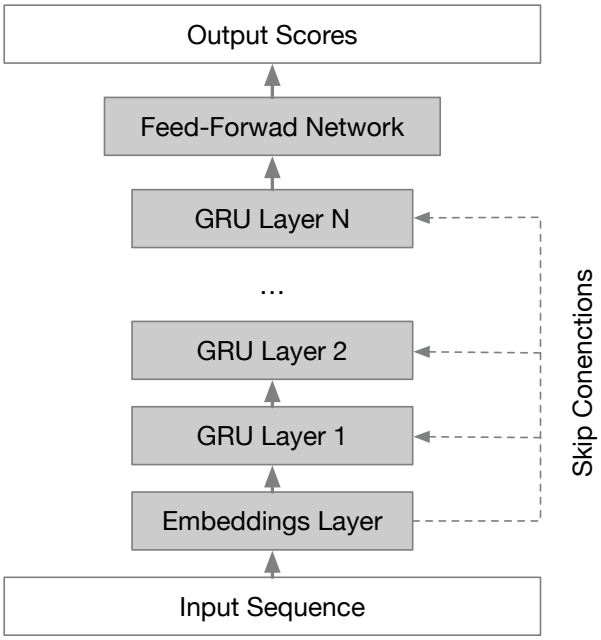


Figure 2.6: Architecture of GRU4Rec. The figure is adapted from [81].

layers (GRU4Rec uses Gated Recurrent Units as the base for the Recurrent Neural Network). To help the model training, GRU4Rec also connects the embeddings layer to some of the deeper GRU layers. Finally, the stack of the GRU Layers is followed by a Feed-Forward neural network, which returns a score distribution across all items in the catalogue.

GRU4Rec is a very representative Deep Learning-based Sequential Recommendation model. Indeed, Figure 2.7 illustrates a more generic architecture, which, with minor modifications, is shared by many models, including GRU4rec [81], GRU4rec$^+$ [80], Caser [236], SASRec [100] and BERT4Rec [230]. As can be seen from the Figure, in this generic architecture, there is an "Encoder Network" block, which depends on the specific model. For example, in GRU4Rec, this network corresponds to a stack of GRU Layers; in Caser, they correspond to convolutional layers, while in SASRec and BERT4Rec, the Encoder Network is based on the Transformer architecture. The output of Encode Network is a vector (embedding) that represents the whole sequence of interactions (shown as the "Sequence embedding" in Figure 2.7). Finally, this Sequence embedding is multiplied by an Item Embedding Ttensor to obtain a score distribution across all items in the catalogue (an equivalent of the Feed Forward layers at the end of the GRU4Rec model). These output embeddings frequently share parameters with the input embeddings: this way, a model does not need to learn two separate sets of item embeddings for input and output.

These models are trained using a version of the Gradient Descent algorithm to minimise the discrepancy between the returned scores distribution and the ground truth item scores (for example, one hot encoded).

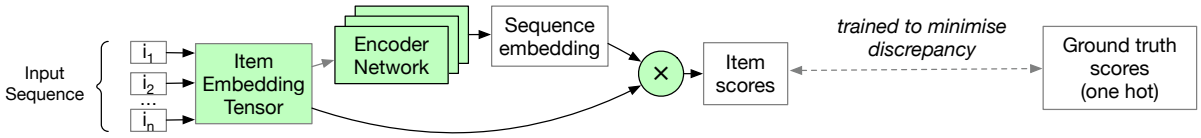


Figure 2.7: Principal architecture of many Sequential Recommenders.

Out of all the models that use this principal architecture, we are mostly interested in Transformer [247]-based models. Indeed, within the last several years, Transformer-based models have achieved state-of-the-art effectiveness and have been dominating the field of Sequential Recommendation. In the next Section, we discuss why Transformer architecture is well-suited for the Sequential Recommendation problem, describe key transformer-based recommendation models, and discuss their limitations, which we will address in this thesis.

2.3 Transformers for Sequential Recommendation

This Section shows the parallels between Language Modelling and Sequential Recommendation, describes the most popular Transformer-based models, outlines their limitations, and describes related work directions that are outside the scope of this thesis. We start by discussing why Sequential Recommendation and Language Modelling are very similar problems.

2.3.1 Parallels Between Language Modelling and Sequential Recommendation

As already briefly mentioned in Chapter 1, the Next Item Prediction task in Sequential Recommendation is very similar to the Next Token Prediction task in Language Modelling. Indeed, training a Sequential Recommendation model can be seen as training a model to understand a "*behavioural language*" in which user-item interactions correspond to words (or to tokens), and the sequences of user-item interactions correspond to documents. Then, in this behavioural language, the goal of a Sequential Recommender System is to determine what word is most likely to appear next in a given sequence of words. According to Robertson [206], "for optimal retrieval, documents should be ranked in order of the probability of relevance," a principle known as the Probability Ranking Principle. In recommender systems, "documents" correspond to items; therefore, the goal of a Sequential Recommender System can be formulated as to estimate the

conditional probability distribution[2]:

$$P(i_{n+1}|i_1, i_2, ...i_n) \tag{2.2}$$

where $i_1, i_2...i_n$ are the historical interactions and i_{n+1} is a possible next item. In other words, using our behavioural language terminology, the goal of a Sequential Recommender System is to estimate the conditional probability distribution across all possible words in our behaviour language. Note that this probability estimation task exactly matches the Probabilistic Language Modelling task, which has been known to the Natural Language Modelling community for many years [12] and can be traced back to the early Claude Shannon works [216, 217]. Hence, the Sequential Recommendation problem can be seen as a special case of Language Modelling, and it is not surprising that Sequential Recommendation models adapt the best Natural Language Models to achieve the best possible effectiveness (note that in Section 2.3.5 we argue that the parallels between Language Modelling and Sequential Recommendation have limitations, making direct adaptations of Language Models challenging in some situations). In particular, the state-of-the-art results for both Language Modelling and Sequential Recommendation are achieved using versions of the Transformer [247] architecture. We now briefly describe the Transformer architecture itself and the key recommendation models that are based on this architecture.

2.3.2 Transformer Architecture

Transformer [247] is a family of neural architectures that was initially introduced for Language Modelling for producing high-quality semantic representations of texts.

A key component of the Transformer architecture is the *Transformer block*, which is based on the Self-Attention mechanism [247]. Self-Attention enables effective encoding of the item representations in the context of the other items in the same sequence.

According to Vaswani et al. [247], Self-Attention is defined as:

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.3}$$

2. Note that for ranking purposes, the absolute value of these probabilities is not important as we are only interested in the ranking of the items according to this probability. Hence, any monotonic transformation of the actual probabilities suits for ranking purposes.

where K, Q and V are three independent linear projections of the original item representation matrix E (we use E for item embeddings to highlight similarities with item embeddings in Matrix Factorisation methods, see Section 2.1.2):

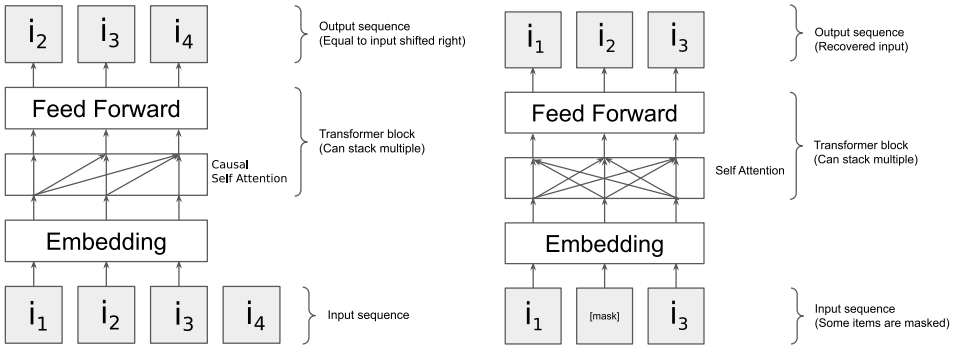
$$\begin{aligned} Q_i &= W_{qi}E_i \\ K_i &= W_{ki}E_i \\ V_i &= W_{vi}E_i \end{aligned} \tag{2.4}$$

Here W_{qi}, W_{ki} and W_{vi} represent three different learnable projection matrices and index i corresponds to item representations after i Transformer blocks. A Transformer block also includes a small pointwise feed-forward network, as well as residual connections and layer normalisations – the standard machine learning techniques to improve model training.

Most Transformer-based models are similar to the Principal Architecture (Figure 2.7), with the Encoder Network corresponding to a stack of Transformer blocks. We note that while the original Transformer architecture included both the *Encoder* and the *Decoder* sub-networks, many modern-day Transformer-based models only use one of these two sub-networks. The main difference between the two is that Decoder uses an *unidirectional* attention, meaning that the contextualised representations of sequence elements only depend on past elements in the sequence. In contrast, Encoder uses a *bidirectional* attention that allows the use of both past and future elements of the sequence for contextualised representations. For more details on Transformer architecture and the differences between Encoder and Decoder, we refer to the original paper [247].

As can be observed from Equations (2.3) and (2.4), Attention-encoded representations do not depend on item position in the sequence. On the other hand, information about the positions of items in a sequence is crucial for Sequential Recommendation, as this information is the only difference between Sequential and Non-Sequential Recommendations. Therefore, information about item positions in the sequence has to be encoded in item representations themselves. To achieve this, following the original Language Models, Transformer-based recommendation models sum item embeddings (obtained from Item Embedding Tensor, recall Figure 2.7) and *positional embeddings* before passing them through the stack of Transformer blocks:

$$E_0 = ItemEmbeddings(i_1, i_2, i_3, ...i_n) + PE \tag{2.5}$$



(a) SASRec **(b)** BERT4Rec

Figure 2.8: Architectures of SASRec & BERT4Rec. SASRec uses causal (unidirectional) Self Attention, whereas BERT4Rec uses regular (bidirectional) Self Attention. SASRec aims to predict the input sequence shifted right, whereas BERT4Rec recovers masked items.

were PE is a matrix of positional embeddings, which only depend on item positions, but do not depend on the items themselves. The original Transformer architecture [247] uses an *absolute* (constant) PE matrix, whose elements p_{ij} are defined as:

$$p_{ij} = \begin{cases} sin\left(\frac{i}{10000^{\frac{j}{d}}}\right); j = 2k \\ cos\left(\frac{i}{10000^{\frac{j-1}{d}}}\right); j = 2k+1 \end{cases} \tag{2.6}$$

where d is the size of the embeddings.

Another approach employed by later Transformer-based models [46, 192] is to use a *learnable* PE Matrix; in that case, we train the positional embeddings as just one of the model parameters. According to Tunstall et al. [245, Chapter 3], absolute positional embeddings are preferable when the dataset size is small, whereas learnable embeddings are a good choice with large datasets. Most Sequential Recommender Systems based on transformers use learnable position embeddings [100, 230].

We now briefly describe the two most popular Sequential Recommendation models: SASRec [100], a Transformer Decoder-based model, and BERT4Rec, a Transformer Encoder-based model.

2.3.3 SASRec

Figure 2.8a illustrates the SASRec model, a Transformer Encoder-based model. The model uses a sequence of items (a history of interactions of a single user) as its input and predicts the same sequence shifted by one element to the right. This means that the last element in the predicted sequence corresponds to the next iteration, which exactly matches the next item prediction—the main task of the Sequential Recommendation. Indeed, at the inference time, SASRec only uses the last element in the predicted sequence.

SASRec uses the causal (unidirectional) version of self-attention: to predict element i_k of the output sequence, it can only access elements $1, 2....i_k$ from the input sequence. Causal self-attention prevents the model from experiencing information leakage: with regular (bidirectional) self-attention, the task would be trivial - the model would copy element i_{k+1} from the input sequence as its k^{th} output.

2.3.4 BERT4Rec

Figure 2.8b illustrates BERT4Rec architecture, a Transformer Decoder-based model. As we can see, it is very similar to the SASRec architecture. One important difference is that BERT4Rec uses regular (bidirectional) self-attention instead of causal attention. BERT4Rec also uses a different training task – instead of predicting the shifted sequence, it employs an item masking training task. Specifically, at the training time, some of the items in the input sequences are replaced with a special *[mask]* token. The goal of the model is to recover these masked tokens. Items masking task allows to generate more training samples – up to $\binom{n}{k}$, where n is the sequence length and k is the number of masked items – out of a single sequence. At the inference time, BERT4Rec adds a *[mask]* token to the end of the sequence of interactions and then for each item in the catalogue, BERT4Rec computes the probability of being this *[mask]* token.

Items masking is loosely connected to the next item prediction task, which makes it harder to train. However, as shown in the original BERT4Rec publication [230], its bidirectional nature may be advantageous and improve the quality of the model. Nevertheless, as we show in the next chapter, the question of whether or not BERT4Rec is a superior model compared to SASRec is still an open question.

2.3.5 Limitations of SASRec and BERT4Rec

As discussed in 2.3.1, the parallels between Sequential Recommendation and Language Modeling enable the reuse of Language Models by substituting token ids for item ids as the model input. These parallels between items in Sequential Recommender Systems and tokens in Language Models allowed SASRec and BERT4Rec to achieve state-of-the-art effectiveness on many Sequential Recommendation datasets.

However, we argue that this substation has a number of limitations, hindering practical deployments of these models. Indeed, the specifics of the "behavioural language" where items are used in place of tokens results in a number of limitations of these models. In particular, we identify five limitations:

Limitations of Transformer-based Sequential Recommendation Models

L2.1: Training is slow.

L2.2: Computing all item scores is expensive during training.

L2.3: The Item Embedding Tensor is too large.

L2.4: Computing all scores is expensive during inference.

L2.5: Complex, beyond-accuracy goals with little training data.

Figure 2.9 also illustrates these limitations on the Principal Architecture of Sequential Recommenders. We now discuss these limitations in turn.

L2.1: *Training is slow*. Differently from Natural Language Processing, where many language-related tasks can be expressed using the same language (i.e. same tokens), most recommendation datasets contain unique sets of tokens. Hence, it is impossible to pre-train a Transformer-based recommendation model on one recommendation dataset and fine-tune it on another. Indeed, researchers usually train these models from scratch for every recommendation dataset. As we

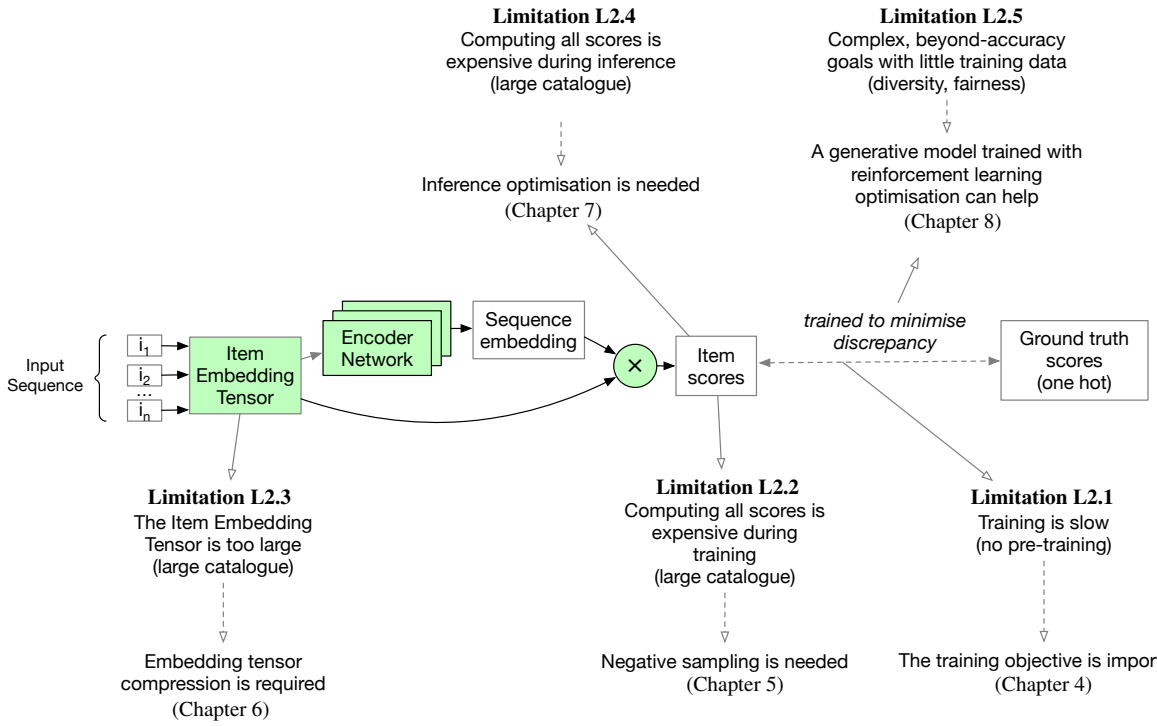


Figure 2.9: Limitations of Transformer-based Sequential Recommendation Models and their coverage in this thesis.

show in the next chapter, training a fully converged BERT4Rec model may require over 16 hours, even on a relatively small MovieLens-1M dataset. As a result, there is a need for efficient training techniques, the problem which we address in Chapter 4 by developing a novel training objective that focuses on recent interactions.

L2.2: *Computing all item scores is expensive during training*. The number of items in the catalogue of typical recommender systems may be much larger when compared to Language Models, making it impossible to compute scores for all items in the catalogue at every training step. Hence, with large catalogues, models use *negative sampling*, which means that the model computes the score for all positive interactions but only for *a few* negative interactions. However, as

we show in Chapter 5, negative sampling drives the model to overestimate interaction probabilities (the problem we call *overconfidence*), which in turn results in low model effectiveness. We address the low effectiveness caused by negative sampling in Chapter 5 by developing a novel loss function that counters the undesirable effects of negative sampling.

L2.3: *The Item Embedding Tensor is too large*. The number of items in a catalogue of Transformer-based Sequential Recommendation models can be many times larger than the number of tokens in Language Models. In this case, the size of the item embedding tensor can easily become too large to fit into a GPU memory, making training of the model unfeasible. Hence, there is a need for techniques that reduce of the size of the item embedding tensor. We address this problem in Chapter 6 by decomposing item ids into a small number of sub-item ids, and hence, reducing the memory required for storing item embeddings.

L2.4: *Computing all scores is expensive during inference*. When there are many items in the catalogue, inference of Transformer-based models becomes very expensive. Indeed, by default, these models require computing the score of every item for every user. With large catalogues, this expensive scoring can make the model deployments unfeasible. We address this problem in Chapter 7 by re-using sub-item ids developed for compressing item embeddings and applying the Dynamic Pruning techniques.

L2.5: *Complex, beyond-accuracy goals with little training data*. SASRec and BERT4Rec can accurately predict interaction probabilities, putting them among the best models for accuracy-based ranking metrics. However, these models are not optimised for beyond-accuracy goals, such as recommendation diversity. Optimising for beyond-accuracy goals is specifically problematic, as most of the recommendation datasets do not provide the training data for optimising these goals, limiting the applications of traditional supervised learning methods. On the other hand, many real-life applications require the model to be aligned with goals beyond accuracy, and for these goals, Transformer-based models may exhibit poor effectiveness. We address this issue in Chapter 8 using a *Generative* model, which we align with beyond-accuracy goals using Reinforcement Learning.

These limitations define the scope of our work and provide a roadmap for the thesis. We now discuss other research directions related to Transformers for recommender systems that fall outside our scope.

2.3.6 Related Work on Transformers for Sequential Recommendation

Side Information. In this thesis, our focus is on collaborative filtering settings; this means that the user is represented as a sequence of items, and the only information the model has about an item is the user-item interactions. This setting is a simplification as, in reality, extra information about the item is usually available, including such information as item description, price, image, and. Integrating side information has been solved in many recent publications, including, most notably, the line of work by Jiacheng Li. Indeed, Li et al. proposed the methods to integrate time intervals [126], textual descriptions [125] and item categories [127] into Transformer-based Sequential Recommendation Models. Other important works on integrating side information into Transformers include Transformers4Rec [45] – an industrial framework by NVIDIA that allows deployments of Transformer-based recommender systems (including adding available side information) and Google's semantic ids approach [194, 222] that allows representing items through their content. Side information is an orthogonal research direction, and in most cases, the methodology presented in this thesis can be used together with the side-information-based methods.

Contrastive Learning. Contrastive learning [50, 189, 262, 283] methods augment the main training objective with an auxiliary contrastive objective to help the model learn more generic sequence representations. The idea is to generate several versions of the same sequence (e.g. crop, reverse, add noise, etc.) and add an auxiliary loss function that ensures that two versions of the same sequence have similar latent representations. In contrast, representations of different sequences are located far away from each other in the latent space. This allows the model to learn more robust representations of sequences and generalise new sequences better. However, these contrastive models still exhibit Limitations L2.1- L2.5. Indeed, a recent publication [284] has demonstrated that contrastive learning is, in fact, very similar to data augmentation and does not improve model effectiveness compared to the best data augmentation methods. Data augmentation is typically needed when the amount of training data available is limited [64, Ch 7.4]. On the other hand, most of the work in this thesis is on large-catalogue systems that are usually associated with very large amounts of available training data. For example, there are roughly 2 billion monthly active users in YouTube [84]; that is, there are 2 billion training sequences available for training the model. Arguably, at this scale, data augmentation is not required, as training data is practically unlimited for model training purposes. Hence, contrastive learning is an orthogonal direction, and auxiliary contrastive loss can be used with the methods described in this thesis.

This ends our overview of the existing Transformer-based Sequential Recommendation models. We now discuss how to evaluate the effectiveness of Sequential Recommenders.

2.4 Evaluation of Sequential Recommender Systems

In this Section, we describe the common evaluation methodology we use in this thesis, including the datasets, data-splitting strategies, evaluation measures, and implementation. We start with an overview of the datasets and splitting strategies.

2.4.1 Datasets and Data Splitting Strategies

An important issue in any Recommender Systems-related research is the selection of appropriate datasets. In Recommender Systems research, there are a number of conflicting recommendations with respect to the most appreciated dataset for the task. For example, recently, Hidasi et al. [79] argued against using Amazon Reviews datasets [73] and recommended using retail clickstream datasets, such as Retail Rocket, as they contain real-life implicit sequential data. In contrast, Klenitsky et al. [107] find that sequential patterns in the Retail Rocket dataset are rather weak, while Amazon Review datasets do exhibit sequential patterns.

Another popular choice for evaluating Sequential Recommender Systems is the MovieLens Dataset [71], which contains movie ratings. However, several researchers argue that the MovieLens dataset is problematic for Sequential Recommendation [51, 79] because it is an explicit rating dataset rather than implicit interactions dataset; nevertheless, it has been shown that strong sequential patterns are present in MovieLens even though the nature of these patterns may be artificial and caused by the peculiarities of the data gathering strategy. Moreover, MovieLens-1M is one of the most popular datasets used in the literature, and most of the researchers use the same version of the dataset, allowing them to compare the results between the papers. Therefore, we include experiments with MovieLens to ensure that our baselines align with prior work and that we can reproduce known results on a widely recognised dataset.

Table 2.1: Salient characteristics of the datasets used in this thesis

Dataset Name	Users	Items	Interactions	Mean Length	Median Length	Long Tail %	Genres
MovieLens-1M	6,040	3,416	999,611	165.50	96	0.00%	18
Steam	281,428	13,044	3,488,885	12.40	8	8.25%	
Beauty (Amazon Reviews)	40,226	54,542	353,962	8.80	6	68.76%	
MovieLens-20M	138,493	26,744	20,000,263	144.41	68	31.41%	
Booking.con	140,746	34,742	917,729	6.52	6	61.84%	
Gowalla	86,168	1,271,638	6,397,903	74.25	28	75.85%	
Tmall	473,376	2,194,464	34,850,828	73.62	29	57.05%	
Yelp	287,116	148,523	4,392,169	15.30	8	23.58%	
Steam-2M	201,963	1,000	2,198,260	10.88	7	0.00%	318

Overall, in the experiments in this thesis, we use a mixture of different datasets that allow us to evaluate our methods from different perspectives. In particular, we use MovieLens-1M [71], MovieLens-20M [71], Beauty [73] and Steam [100]. These datasets were used in the original BERT4Rec paper [230] and allow us to directly compare our results with the results reported by that paper (as well as many other papers that used the same datasets and the same preprocessing). For these datasets, the authors of BERT4Rec provided preprocessed versions in their repository[3], and to stay consistent, we use these datasets from their repository without any further preprocessing.

In addition, we use the following datasets in our experiments: Yelp [8] is one of the most popular datasets in the Sequential Recommendation research; Booking.com [63] is the trip destination dataset used in the 2021 WSDM challenge where several winning solutions were Transformer-based; Gowalla [32] is a check-in dataset which is useful for this thesis as the number of items in this dataset is over one million; Tmall [237] is another large-scale dataset with more than a million items; Steam-2M is our version of the Steam dataset [100] where we only leave 1000 most popular items (we create this version to iterate quickly in computational-heavy experiments). As we do not focus on the cold start problem in this thesis, we remove the users with less than 5 interactions at the preprocessing stage for all of our experiments.

Table 2.1 reports the salient characteristics of all experimental datasets. In addition to the standard characteristics of recommendation datasets, such as number of users, number of items and number of interactions, the table also reports sequence-related characteristics, such as mean sequence length and median sequence length. We also report the percentage of Long Tail items (items with less than 5 interactions), which is important for our embedding compression study

3. https://github.com/FeiSun/BERT4Rec/tree/master/data

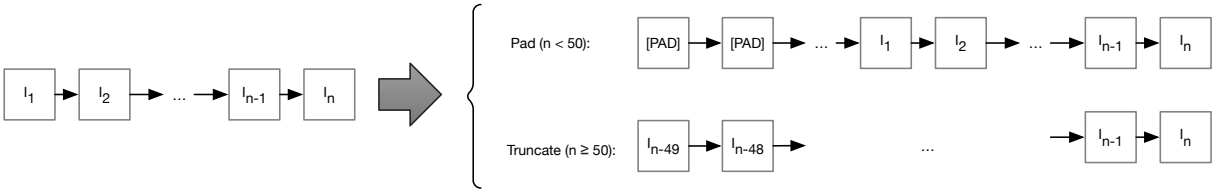


Figure 2.10: Padding/Truncation scheme. We use left padding in experiments, ensuring that the rightmost input to the model is always the most recent interaction.

(see Chapter 8). Finally, for two datasets (MovieLens-1M and Steam-2M), we report the total number of genres associated with items – this is important when we compute beyond-accuracy metrics in Chapter 8. The table shows that the datasets have a diverse range of characteristics, allowing us to focus on different aspects of Sequential Recommendation models.

Both BERT4Rec [230] and SASRec [100] original publications, as well as most of the other publications in the field of Sequential Recommendation, use the "Leave-One-Out" data splitting strategy. For every user sequence, the last interaction in the sequence is held out to the test set, and then these interactions are used as the test set. Following these publications, we use the "Leave-One-Out" splitting in this thesis. Moreover, we use the second-to-last interaction as the validation interaction, which we use to monitor model quality during the draining, as well as use validation data to select the best model in the early stopping mechanism.

Most Sequential Recommendation models, including SASRec and BERT4Rec, require input sequences to have a fixed length for efficient batch processing. In our experiments, we usually fix the sequence length between 50 and 200 (we report this length separately in every experiment). Consider we set the fixed input of the model to have n interactions. If a user has less than n interactions, we add special "*[pad]*" interactions at the beginning of the sequence. If the user has more than n interactions, we take the most recent n interactions from the sequence. Figure 2.10 illustrates this padding/truncation scheme visually. The scheme ensures that the user's most recent interaction is always located in the rightmost position (position with index 49 in the figure), and padding is added to the beginning of the sequence.

2.4.2 Evaluation and Metrics

Choosing the right set of metrics for evaluating recommender systems is one of the most important parameters of the experimental evaluation [67]. There is no "one-size-fits-all" metric, and different real-world systems optimise for different metrics.

The best way to analyse the quality of a recommender system is to bring the system in front of real users and analyse the user's response in *online experiments* (e.g. using A/B tests). However, within academic research, access to online experiments is limited. Hence, within the academic community, *offline experiments* are typically used as a proxy for online experiments. In these offline experiments, historical user interaction data is usually split into "train" and "test" partitions, and the goal of a recommender system is to predict the "test" interactions using the data from the "train" partition. While offline experiments are not ideal and do not always match the results of offline experiments, they still remain useful. Indeed, any recent advancements in recommender systems were achieved only using offline experiments. Hence, in this thesis, we also focus on offline measures.

When measuring the effectiveness of a recommender system, there are two types of effectiveness metrics: *accuracy-based metrics* and *beyond-accuracy metrics*. As most of this thesis focuses on accuracy-based metrics, we provide the common accuracy-based measures here and leave the description of the used beyond-accuracy measures to Chapter 8.

In particular, in most of the methodological chapters of this paper, we use two common ranking-based measures:

Normalised Discounted Cumulative Gain (NDCG). The Normalized Discounted Cumulative Gain (NDCG) at rank k [91] is defined as:

$$\text{NDCG@}k = \frac{\text{DCG}_k}{\text{IDCG@}k}, \tag{2.7}$$

where the Discounted Cumulative Gain (DCG) is given by

$$\text{DCG@}k = \sum_{i=1}^{k} \frac{2^{r_i} - 1}{\log_2(i+1)}, \tag{2.8}$$

and r_i is the relevance score of the result at position i. The Ideal DCG (IDCG) is the maximum possible DCG achievable with the ideal ordering of results.

In the Leave-One-Out evaluation scenario, there is exactly one relevant ground truth item i, for which we deem the relevance label r_i to be equal to one. For every other item in the catalogue, we set the relevance label to zero. Hence, for our scenario, NDCG@K can be simplified. Let p be the rank position of the ground truth relevant item. The Discounted Cumulative Gain (DCG)

at rank k is defined as:

$$\text{DCG@}k = \begin{cases} \dfrac{1}{\log_2(p+1)} & \text{if } p \leq k, \\ 0 & \text{if } p > k. \end{cases}$$

Since the ideal ranking places the relevant document at the top (i.e., $p = 1$), the Ideal DCG (IDCG) is:

$$\text{IDCG@}k = \frac{1}{\log_2(1+1)} = 1.$$

Therefore, the Normalized Discounted Cumulative Gain (NDCG) at rank k is:

$$\text{NDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k} = \begin{cases} \dfrac{1}{\log_2(p+1)} & \text{if } p \leq k, \\ 0 & \text{if } p > k. \end{cases}$$

Recall. The Recall at rank k [205, Chapter 7] is defined as the fraction of relevant items retrieved among all relevant items. In our setting, where there is exactly one relevant ground truth item, Recall@k can be expressed as:

$$\text{Recall@}k = \begin{cases} 1, & \text{if } p \leq k, \\ 0, & \text{if } p > k. \end{cases}$$

That is, Recall@k equals 1 if the ground truth relevant item appears in the top k positions and 0 otherwise. Since this metric has a binary outcome, it is often referred to as "Hit Rate" in Sequential Recommendation literature [100, 230]. However, we use the more conventional name, Recall, to ensure broader familiarity

Note that in most of our experiments, we use the full items collection as our candidates set when computing the metrics and do not employ any negative sampling at the inference time, as recommended by recent best practices [23, 117]. The only exceptions to this rule are some of the experiments in Chapter 3, where we use popularity-sampled metrics alongside the unsampled metrics (Chapter 3 contains results of a replicability study of BERT4Rec. Hence, we use the experimental methodology described in the original BERT4Rec paper in that chapter).

We also use several model efficiency measures across chapters, including training speed, required memory, and inference latency. As these measures are chapter-specific, we introduce them in the corresponding chapters of this thesis.

2.4.3 Implementation

We implement the code for all methodological chapters within the *aprec* recommendation platform[4]. We developed the early version of the platform as part of our solution [183] for the Booking.com challenge [63] just before starting the work on this thesis. The main components (including BERT4Rec and SASRec models) of the platform were developed and validated during the work on the Replicability study of BERT4Rec, which is described in Chapter 3.

Both BERT4Rec and SASRec were originally implemented using TensorFlow [1]. Hence, to stay as aligned as possible with the original versions of the models, we use Tensorflow as our main Deep Learning framework (however, we also ported some of the methods to Pytorch).

This finalises the background for this thesis. We now summarise the background and our identified research gaps.

2.5 Background Recap and Research Gaps

Summarising the background, we find that Sequential Recommendation is an important research direction that addresses a number of limitations of traditional Matrix Completion methods, such as using causal dependencies in the data or modelling evolving user interests. State-of-the-art models for Sequential Recommendation are based on the Transformer architecture, with SASRec and BERT4Rec being the most popular. Despite achieving high effectiveness on small-scale academic datasets, these models have limitations on large-scale real-world datasets. These limitations, outlined in Section 2.3.5, include slow training (Limitation L2.1), expensive scores computing during training (Limitation L2.1), large Item Embedding Tensor (Limitation L2.3), expensive scores computing during inference (Limitation L2.3), and complex beyond accuracy goals with little training data (Limitation L2.5)).

Figure 2.9 also illustrates these limitations and provides the roadmap for the rest of the thesis. Indeed, the main focus of the remaining chapters in this thesis is on addressing these limitations. However, when we started our work on the first limitation, we realised that there were many inconsistencies in the reported results in the literature for sequential recommender systems. Hence,

4. https://github.com/asash/bert4rec_repro

in order to establish a solid ground for our research, in the next chapter, we perform the *Systematic Literature review* of SASRec and BERT4Rec and do a *Replicability Study* of these models. This study helps us to make sure that any of our conclusions are validated against strong and well-optimised baselines.

Chapter 3

A Systematic Review and Replicability Study of BERT4Rec for Sequential Recommendation

In this chapter, we analyse the effectiveness and training efficiency of both SASRec (Section 2.3.3) and BERT4Rec (Section 2.3.4) through both a systematic literature review and a replicability study of the originally reported results. In Section 3.1, we motivate the need for a systematic review and replicability study. Section 3.2 describes our systematic review methodology and finds that the results reported in the literature are not consistent. Section 3.3 describes the available BERT4Rec implementations that we use in our replicability study. Section 3.4 describes the methodology of the replicability study. Section 3.5 analyses the results of the study. Section 3.6 put this work in context wrt. related work and performances observed in recent publications; Section 3.7 summarises the results of the replicability study and contains final remarks.

The material of this chapter is based on our paper [174], which was published as a full paper at the reproducibility track of the ACM RecSys'22 conference.

3.1 Motivation: Examining the Need for a Systematic Review and Replicability Study on BERT4Rec

As discussed in Chapter 2, BERT4Rec [230] and SASRec [100] are highly-cited Transformer-based Sequential Recommendation models. In the original BERT4Rec publication, Sun et al. [230] argued that one of the main advantages of BERT4Rec compared to other Transformer-based models, such as SASRec, is that it uses an item masking training task [46]. Applied to Sequential Recommender Systems, the main idea of this training task is to replace random items in the training sequences with a special *[mask]* token and train the model to recover these masked tokens; we describe details of this task in Section 2.3.4. In the original publication [230], BERT4Rec was claimed to achieve significant superiority over existing neural and traditional approaches; however, subsequent publications by a number of different authors (e.g. [53, 271, 283]) did not confirm this superiority. Our goal in this chapter, therefore, is to understand the reasons behind this discrepancy. In particular, we analyse three open-source implementations of BERT4Rec with the aim of reproducing the originally reported results using these implementations. We find that there is a marked discrepancy in both the effectiveness and efficiency of these implementations. We also demonstrate that training the original implementation of BERT4Rec, with its default configuration, results in an underfitted model, and it requires up to 30$\times$ more training time in order to replicate the originally reported results.

Ultimately, we show that, with proper configuration, BERT4Rec achieves better performance than earlier models such as SASRec and that some results reported in subsequent papers are based on underfitted versions of BERT4Rec. Moreover, we show that an appropriately trained BERT4Rec can match or outperform later models (e.g. DuoRec [189], LightSANs [52] & NOVA-BERT [140]) and therefore may still be used as a state-of-the-art sequential recommendation model.

In addition, we propose our own implementation of BERT4Rec that is based on a popular Hugging Face Transformers library [255]. Hugging Face Transformers is a popular Machine Learning library with a large supporting community. For instance, its GitHub repository[1] has almost 62K stars and 15K forks, which makes it the second most popular machine learning library on GitHub.[2] It has become a de facto standard for publishing Transformer-based models and therefore contains well optimised and efficient versions of Transformer-based models. Based on this and the fact that it was already successfully applied for recommendations [45], we expect that using it as a backbone for building Transformer-based recommenders should be both effective and efficient. Indeed, we show that our implementation with default configuration replicates the results published by Sun et al. [230] and requires up to 95% less time to achieve these results. Moreover, the BERT architecture in our implementation can be easily replaced with other models available in the Hugging Face Transformers library. We demonstrate by using two examples (DeBERTa [72] and ALBERT [120]) that such replacement can lead to further improved effectiveness.

Figure 3.1 summarises the results of our findings on the MovieLens-1M dataset. Each point on the figure corresponds to a default configuration of a recommendation model, and the dashed line represents the performance of the original BERT4Rec code when trained for different amounts of time. The figure shows that, depending on the chosen BERT4Rec implementation, the observed NDCG@10 effectiveness can vary from 0.0546 for RecBole implementation to 0.156 (our implementation, 3x difference). It also shows that the required training time also depends on the implementation. For example, the BERT4Rec-VAE implementation requires 18 minutes to train with default configuration, whereas the original implementation requires 44 minutes while achieving poorer NDCG@10 than BERT4Rec-VAE. The figure also shows that the original implementation can reach the same level of performance as the best implementation. However, it requires much more training than what is specified in the default configuration. The figure

1. https://github.com/huggingface/transformers
2. After Tensorflow, according to https://github.com/EvanLi/Github-Ranking, as of 25/03/2022

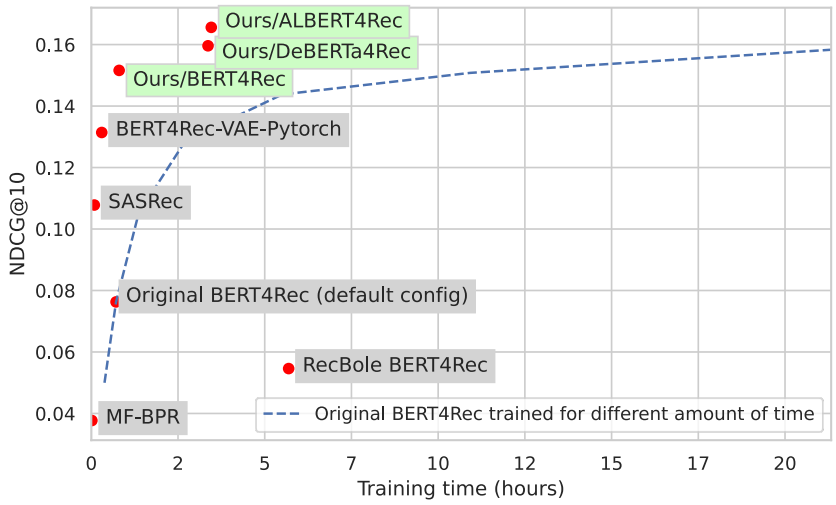


Figure 3.1: Comparison of four implementations of BERT4Rec (Original, RecBole, BERT4Rec-VAE-Pytorch, Ours/BERT4Rec) with two baseline models (MF-BPR, SASRec) and two more advanced Transformer-based models (Ours/ALBERT4Rec, Ours/DeBERTa4Rec) on the MovieLens-1M dataset. All points represent models with their default configuration. The dashed line represents NDCG@10 of the original BERT4Rec implementation trained for different amounts of training time. Note that these results are not comparable with the results reported in the original BERT4Rec paper [230], as we do not use negative sampling for metrics. For comparison in BERT4Rec's original setup see Section 3.5.2.

also portrays results of our implementation based on DeBERTa [72] and ALBERT [120] Transformers and shows that these models can outperform BERT4Rec, however they require more time than our BERT4Rec implementation. Nevertheless, their training times are still 85% smaller than what is required for the full convergence of the original BERT4Rec implementation.

In short, the contributions of this chapter are:

1. A systematic review of the papers comparing BERT4Rec and SASRec, which shows that the results of such comparisons are not consistent;
2. An analysis of the available BERT4Rec implementations, which shows that frequently these implementations fail to reproduce results reported in [230] when trained with their default parameters,
3. An analysis of the impact of the training time on the recommendation performance of the original BERT4Rec implementation;
4. A new Hugging Face Transformers-based implementation of BERT4Rec that successfully replicates the results reported in [230], while being faster to train;

5. Two new recommendation models, DeBERTa4Rec and ALBERT4Rec, that improve the quality of our implementation by replacing the BERT model with DeBERTa and ALBERT, respectively. The code and the documentation for this chapter can be found in our GitHub repository.[3]

We now turn to the systematic review of the literature that compares BERT4Rec's and SASRec's effectiveness.

3.2 Systematic review of SASRec and BERT4Rec

According to the original publication [230], BERT4Rec outperformed the second-best model, SASRec, on four different datasets (Beauty [74], Steam [100], MovieLens-1M [71] and MovieLens-20M [71]) and across six different evaluation metrics (Recall@{1, 5, 10}, NDCG@{5, 10} and MRR). In order to validate this result and estimate its reproducibility, we perform a systematic review of the papers that compare SASRec and BERT4Rec. We formulate two hypotheses that we test in our systematic review:

Systematic Review Hypothesis

H3.1: Overall, BERT4Rec is not systematically better than SASRec in the literature.

H3.2: Even when the experimental setup is similar (e.g. experiment on the same datasets), the outcome of the comparison of BERT4Rec and SASRec may be different, which indicates poor replicability of BERT4Rec results.

Hypothesis H3.1 addresses a general question of whether BERT4Rec is indeed a superior model when compared to SASRec; i.e. it outperforms SASRec across a large number of different datasets and metrics. Hypothesis H3.2 addresses the question of reproducibility; we look at particular datasets and metrics and check if the experiment results are consistent within this experimental setup.

3. https://github.com/asash/bert4rec_repro

3.2.1 Review Methodology

To test Hypotheses H3.1 and H3.2 we conducted a review of all papers citing the original BERT4Rec publication [230] according to the Google Scholar website.[4] From these papers, we chose to include only those papers that used both BERT4Rec and SASRec models for their experiments and examined their observed performances. We further excluded papers that were not peer-reviewed to minimise the chances of relying on non-verified experiments.[5] We considered performing a *meta analysis*, by aggregating improvements across multiple papers (e.g. "average improvement by BERT4Rec in NDCG@10 over SASRec"). However, we observe that the identified papers applied various evaluation measures (e.g. varying rank cutoffs) and methodology (e.g. different negative sampling strategies, see Section 3.4.4). Instead, we considered three outcomes of the experiments: "BERT4Rec wins", "SASRec wins" and "Tie", and we relied on counting the outcomes. In particular, we counted that a particular model wins a comparison if it was better according to all metrics used in the experiment. If a model was better according to one subset of metrics and worse according to another, we counted a tie.

To test the overall superiority of BERT4Rec over SASRec (Hypothesis H3.1), we determine the total numbers of each possible outcome (BERT4Rec wins; SASRec wins; Tie) and check whether or not there is any considerable amount of situations when SASRec wins over BERT4Rec. To test the results' reproducibility on different datasets (Hypothesis H3.2), we aggregate the results by datasets and count experiment outcomes for each dataset independently.

3.2.2 Systematic Review Results

We reviewed the 370 papers citing BERT4Rec according to Google Scholar on 25/03/2022.[6] Following the inclusion criteria (comparison with SASRec), we found 58 publications containing such comparisons and, after that, excluded 18 (not peer-reviewed, etc.). This left a total of 40 publications that compare BERT4Rec and SASRec, making a total of 134 comparisons on 46 different datasets (3.35 comparisons per paper on average, 2.91 comparisons per dataset on average).

---

4. `http://scholar.gooogle.com`
5. Other examples of excluded publications include a paper withdrawn by authors – and only available in the web archive – and an MSc student thesis.
6. The spreadsheet of analysed papers is also included in our Github repository.

Table 3.1: Results of BERT4Rec vs SASRec comparisons in the peer-reviewed publications. Bold denotes a model with more wins on a dataset. Asterisk (*) denotes datasets used in the original BERT4Rec paper [230]. Only the datasets appearing in at least five papers are presented (8 datasets out of 46). However, the comparison results on all other 38 datasets are included in the "Total" numbers.

dataset	total	BERT4Rec wins	SASRec wins	Ties	BERT4Rec wins papers	SASRec wins papers	Ties papers
Beauty* [74]	19	**12 (63%)**	5 (26%)	2 (11%)	[230], [145], [282], [143], [124], [33], [275], [242], [271], [76], [258], [231]	[283], [130], [279], [248], [189]	[14], [40]
ML-1M* [71]	18	**13 (72%)**	3 (17%)	2 (11%)	[230], [145], [282], [124], [33], [98], [242], [76], [258], [224], [167], [186], [123]	[52], [271], [189]	[275], [40]
Yelp [8]	10	**6 (60%)**	4 (40%)	0 (0%)	[283], [5], [14], [248], [167], [189]	[52], [130], [279], [132]	
Steam* [170]	8	**7 (88%)**	1 (12%)	0 (0%)	[230], [145], [282], [124], [275], [258], [40]	[271]	
ML-20M* [71]	8	**7 (88%)**	0 (0%)	1 (12%)	[230], [145], [280], [124], [33], [258], [186]		[40]
Sports [74]	6	1 (17%)	**4 (67%)**	1 (17%)	[123]	[283], [130], [279], [189]	[14]
LastFM [24]	6	**4 (67%)**	2 (33%)	0 (0%)	[283], [98], [242], [123]	[5], [279]	
Toys [74]	5	0 (0%)	**5 (100%)**	0 (0%)		[283], [53], [130], [279], [14]	
Total	134	**86 (64%)**	32 (23 %)	16 (12 %)			

3.2.2.1 H3.1. Overall results consistency

Table 3.1 summarises the results of our systematic review. As we can see from the "Total" row of the table, BERT4Rec indeed wins the majority of the comparisons (86 out of 134, 64%). However, the number of comparisons in which SASRec won (32 out of 134, 24%) or at least achieved a tie (16 out of 134, 12%) is not negligible: 36% overall. Therefore, we can conclude that BERT4Rec is not consistently superior compared to SASRec in the published literature, which validates Hypothesis H3.1 and is not consistent with the original BERT4Rec paper. We now investigate if the differences in datasets can explain this inconsistency, i.e. if the reason is that BERT4Rec is better on some datasets, but SASRec is better on others.

3.2.2.2 H3.2. Poor replicability of BERT4Rec

We turn again to Table 3.1 and analyse the results of the systematic review aggregated by dataset. The table includes all datasets from our review appearing in at least five papers. From the table, we observe that the proportion of outcomes is indeed highly dependent on the dataset. For example, on both the ML-20M and Steam datasets, BERT4Rec won 7 out of 8 comparisons (88%), whereas on Sport, it won just 1 out of 6 (17%), and on Toys, it won 0 out of 5 (0%) comparisons. Therefore, it appears that the disparity in the overall results can be explained by some salient characteristics of the datasets.

However, there are still some inconsistencies within datasets. For example, on the two most popular datasets, Beauty and ML-1M, the proportion of experiments which BERT4Rec did not win roughly matches the overall result: 7 out of 19 for Beauty (37%) and 5 out of 13 for ML-1M (28%). Importantly, the original BERT4Rec paper [230] found that BERT4Rec was superior by a statistically significant margin on both of these datasets, however we can see a large number of papers failed to replicate this result, which validates our Hypothesis H3.2.

Overall, we argue that the lack of agreement in the observed results is problematic: it means that papers have used BERT4Rec with different configurations. Ultimately, many of these papers may be using poor configurations of BERT4Rec as their baselines. This leads us to investigate the available open-source implementations of BERT4Rec and their hyperparameter settings.

Table 3.2: BERT4Rec implementations used in our experiments. The GitHub stars are as of 25/03/2022.

Implementation	GitHub URL	Framework	GitHub stars	Example papers
Original	FeiSun/BERT4Rec	Tensorflow v1	390	[85, 87, 142]
RecBole	RUCAIBox/RecBole	PyTorch	1,800	[14, 52, 131]
BERT4Rec-VAE	jaywonchung/BERT4Rec-VAE-Pytorch	PyTorch	183	[195, 244, 272]
Ours/HuggingFace	asash/bert4rec_repro	Tensorflow v2	N/A	N/A

3.3 BERT4Rec implementations

To understand the reasons for the poor replicability of BERT4Rec results, demonstrated by our systematic review, we analyse the available BERT4Rec implementations. During our systematic review, we identified three available BERT4Rec implementations cited in the papers:

1. The original implementation, provided by the authors of the original paper [230];
2. RecBole [281] - a library that contains a large number of recommendation models, including BERT4Rec;
3. BERT4Rec-VAE - a project that implements BERT4Rec and Variational Autoencoder [134] models using PyTorch.[7]

In addition to these three existing implementations, we note the Transformers4Rec project [45], which is based on the popular HuggingFace Transformers natural language processing library [255]. Transformers4Rec does not include an implementation of BERT4Rec; however, inspired by this project, we implement our version of BERT4Rec that uses HuggingFace's version of BERT as a backbone. Indeed, given the popularity of the HuggingFace Transformers library, we expect that models implemented using it will be highly efficient and may outperform other available implementations.

Salient characteristics of all four BERT4Rec implementations are listed in Table 3.2. We now turn to the experiments with these four implementations, which help us to understand the poor replicability of BERT4Rec results.

7. https://github.com/jaywonchung/BERT4Rec-VAE-Pytorch

3.4 Experimental Setup

3.4.1 Research Questions

We aim to address the following research questions with our experiments:

BERT4Rec Replicability Research Questions

RQ3.1: Can we replicate the state-of-the-art results reported in [230] using all of the available BERT4Rec implementations, applying their default configurations?

RQ3.2: What is the effect of the training time on the performance of the original BERT4Rec implementation?

RQ3.3: Can our HuggingFace-based implementation of BERT4Rec benefit from replacing BERT with another language model available in the HuggingFace Transformers library?

3.4.2 Datasets

Because our goal is to examine replicability of the BERT4Rec, we use same four datasets as used in the original publication [230], namely ML-1M [71], Beauty [74], Steam [100] and ML-20M [71]. These datasets also represent 4 of the 5 most popular datasets in our systematic review. The data preprocessing and splitting strategy is as described in Section 2.4.1.

3.4.3 Models

For our replicability experiment (RQ3.1, we use four BERT4Rec implementations, as described in Section 3.3. For all four implementations, we apply their default data prepossessing pipelines and default model parameters. Table 3.3 lists the default parameter values. The original BERT4Rec version includes slightly different parameters tuned for each different dataset, and we use the ones specified for the ML-20M dataset; however, according to the original publication [230], these changes in parameters within the limits specified in the configuration files have only a limited effect and therefore should not change overall model performance. For the ML-1M and ML-

20M datasets, we also experiment with a so-called "longer seq" version, where we increase the maximum sequence length because the average number of items per user is much larger in these two datasets. Overall, the configuration of our implementation of BERT4Rec is similar to the original model, with a notable difference in the training stopping criteria: to ensure that the models are fully trained, we use an early stopping mechanism; we measure the value of loss function on the validation data and stop training if validation loss did not improve for 200 epochs.

We also use two baselines in our replicability experiments:

1. MF-BPR [199] - a classic matrix factorisation-based approach with a pairwise BPR loss. We use the implementation of this model from the LightFM library [119] and set the number of latent components to 128;
2. SASRec [100] - a Transformer-based model, described in Section 2.3.3. We use an adaptation of the original code for this model.[8] For SASRec we set sequence length to 50, embedding size to 50 and use two transformer blocks; according to the experiments conducted by Kang et al. [100], these parameters are within the range where SASRec shows reasonable performance.

In the extra training experiment (RQ3.2), we use the default configuration of the original BERT4Rec implementation with the exception of a number of training steps, which vary between 200,000 and 12,800,000.

Finally, to compare BERT with other models available in the HuggingFace Transformers library (RQ3.3), we experiment with two recent Transformer-based architectures: (1) DeBERTa [72] - a model that improves BERT with a disentangled attention mechanism [72] where each word is encoded using two vectors (a vector for content and a vector for position); (2) ALBERT [120] - a model that improves BERT via separating the size of the hidden state of the vocabulary embedding from the number of the hidden layers. It also introduces cross-layer parameter sharing, which allows for a decrease in the overall number of parameters in the network. Following the BERT4Rec naming convention, we call these two models DeBERTa4Rec and ALBERT4Rec. For these two models, we use the same model configuration parameters as for BERT4Rec with the exception of longer sequence length, for which we considered values from {50, 100, 200} and report the results for the best-performing value (200) according to NDCG@10 metric on the validation set.

8. https://github.com/kang205/SASRec

Table 3.3: Default parameters of the BERT4Rec implementations.

Implementation	Original	RecBole	BERT4Rec-VAE	Ours	Ours (longer seq)
Sequence length	200	50	100	50	100
Training stopping criteria	400,000 steps	300 epochs	200 epochs	Early stopping: 200 epochs	Early stopping: 200 epochs
Item masking probability	0.2	0.2	0.15	0.2	0.2
Embedding size	64	64	256	64	64
Transformer blocks	2	2	2	2	2
Attention heads	2	2	4	2	2

3.4.4 Metrics

Following Sun et al. [230], we evaluate the models using the Leave-One-Out strategy and focus on two ranking-based metrics: NDCG and Recall@K on the test data.[9] The authors report results using *sampled metrics*: for each positive item, they sample 100 negative items - these 101 items are then ranked for evaluation. In particular, they use a popularity-based sampling of negatives: the probability of sampling an item as a negative is proportional to its overall popularity in the dataset. However, as we discussed in Section 2.4.2, sampled metrics are known to be problematic. Indeed, it has been shown in a number of recent publications [23, 40, 117] that sampled metrics are not always consistent with full, unsampled versions (where all items are ranked for each user) and can lead to incorrect model comparisons.

Nevertheless, because our goal is to examine the reproducibility of the BERT4Rec model, we apply sampled metrics to compare our results with the results reported in the original paper. In addition, following the recommendations in [117] and [23], we report results on *unsampled metrics*, where we rank all items from the dataset at evaluation time. However, in experiments where we do not need to compare with the results reported in the original publication [230] we only report unsampled metrics.

9. The original BERT4Rec publication uses the name Hit Rate (HR) instead of Recall. The Hit Rate represent the probability of successful prediction of one single interaction in the next-item prediction task. In the case of sequential recommendation, when there is only one true positive item per user, this metric is equivalent to Recall, and we prefer the more conventional name.

3.4.5 Criteria for a Successful Replication

There is some level of randomness involved in deep neural network training. Indeed, even when the same model is trained multiple times, random weights initialisation and random data shuffling can lead to small changes in the resulting effectiveness metrics. BERT4Rec also randomly masks items during training, which also slightly increases the randomness of the process. Furthermore, as mentioned above, the evaluation metrics used by the original BERT4Rec publication [230] involve sampling random items and, therefore, may change even when evaluating exactly the same model.

Thus, it is almost impossible to replicate the reported results exactly. Therefore, we need to define some interval around the values reported in [230] within which we can say that our model replicates originally reported results. Because we only know a single measurement per dataset, we can not rely on hypothesis testing methods, such as paired t-tests, so we need a heuristic for defining the tolerance interval.

According to Madhyastha and Jain [150], the standard deviation in classification metrics for instances of the same deep neural model trained with different random seeds can reach 2.65%. If we assume that ranking metrics, such as NDCG and Recall, exhibit similar variance under different model instances, this suggests that a larger tolerance is needed for defining equivalence. Hence, in this chapter, we define the successful replicability criteria as follows:

Successful Replication Criteria

A model replicates the results reported in the original publication on a metric if the metric value is equal to the originally reported value within a relative tolerance of $\pm 5\%$.

Table 3.4: Replicability of the originally reported BERT4Rec results. **Bold** denotes a successful replication of a metric reported in [230], percentages show the difference with the reported metric, underlined denotes the best model by a metric, † denotes a statistically significant difference with the Bonferroni multiple testing correction ($pvalue < 0.05$).

(a) ML-1M Dataset

	Model	Popularity-sampled		Unsampled		Training
		Recall@10	NDCG@10	Recall@10	NDCG@10	Time
Baselines	MF-BPR	0.5134 (-26.34%)†	0.2736 (-43.21%)†	0.0740†	0.0377†	58
	SASRec	0.6370 (-8.61%)†	0.4033 (-16.29%)†	0.1993†	0.1078†	316
BERT4Rec versions	Original	0.5215 (-25,18%)†	0.3042 (-36.86%)†	0.1518†	0.0806†	2,665
	RecBole	0.4562 (-34.55%)†	0.2589† (-46.26%)†	0.1061†	0.0546†	20,499
	BERT4Rec-VAE	**0.6698 (-3.90%)**†	0.4533 (-5.29%)†	0.2394†	0.1314†	1,085
	Ours	**0.6865 (-1.51%)**	**0.4602 (-4.48%)**	0.2584	0.1392	3,679
	Ours (longer seq)	**0.6975 (+0.07%)**	**0.4751 (-1.39%)**	0.2821	0.1516	2,889
Reported [230]	BERT4Rec	0.6970	0.4818	N/A	N/A	N/A

(b) Steam Dataset

	Model	Popularity-sampled		Unsampled		Training
		Recall@10	NDCG@10	Recall@10	NDCG@10	Time
Baselines	MF-BPR	0.3466 (-13.63%)†	0.1842 (-18.53%)†	0.0398†	0.0207†	162
	SASRec	0.3744 (-6.70%)†	0.2052 (-9.24%)†	0.1198†	0.0482†	3,614
BERT4Rec versions	Original	0.2148 (-46.47%)†	0.1064 (-52.94%)†	0.0737†	0.0375†	4,847
	RecBole	0.2325 (-42.06%)†	0.1177 (-47.94)†	0.0744†	0.0377†	83,816
	BERT4Rec-VAE	0.3520 (-12.29%)†	0.1941 (-14.15%)†	0.1237†	0.0526†	65,303
	Ours	**0.3978 (-0.87%)**	**0.2219 (-1.86%)**	0.1361	0.0734	117,651
Reported [230]	BERT4Rec	0.4013	0.2261	N/A	N/A	N/A

(c) Beauty Dataset

	Model	Popularity-sampled		Unsampled		Training
		Recall@10	NDCG@10	Recall@10	NDCG@10	Time
Baselines	MF-BPR	0.2090 (-30.91%)†	0.1089 (-41.47%)†	0.0185†	0.0090†	58
	SASRec	0.1111 (-63.27%)†	0.0524 (-71.83%)†	0.0079†	0.0036†	316
BERT4Rec versions	Original	0.1099 (-63.67%)†	0.0567 (-69.55%)†	0.0163†	0.0079†	3,249
	RecBole	0.1996 (-34.02%)†	0.1103 (-40.76%)†	0.0158†	0.0079†	11,024
	BERT4Rec-VAE	0.2339 (-22.68%)	0.1407 (-24.44%)	0.0331	0.0188	21,426
	Ours	0.1891 (-37.49)†	0.0919 (-50.64%)†	0.0166†	0.0080†	14,497
Reported [230]	BERT4Rec	0.3025	0.1862	N/A	N/A	N/A

(d) ML-20M Dataset

	Model	Popularity-sampled		Unsampled		Training
		Recall@10	NDCG@10	Recall@10	NDCG@10	Time
Baselines	MF-BPR	0.6126 (-18.02%)†	0.3424 (-35.88%)†	0.0807†	0.0407†	197
	SASRec	0.6582 (-11.92%)†	0.4002 (-25,06%)†	0.1439†	0.0724†	3635
BERT4Rec versions	Original	0.4027 (-46.11%)†	0.2193† (-58.93%)†	0.0939†	0.0474†	6,029
	RecBole	0.4611 (-38.30%)†	0.2589 (-51.52%)†	0.0906†	0.0753†	519,666
	BERT4Rec-VAE	**0.7409 (-0.86%)**	**0.5259 (-1.52%)**	0.2886	0.1732	23,030
	Ours	**0.7127 (-4.63%)**†	0.4805 (-10.02%)†	0.2393†	0.1310†	44,610
	Ours (longer seq)	**0.7268 (-2.74)**†	0.4980 (-6.74%)†	0.2514†	0.1456†	39,632
Reported [230]	BERT4Rec	0.7473	0.5340	N/A	N/A	N/A

3.5 Experimental Results

3.5.1 RQ3.1: Replicability with default configurations.

We first analyse if the available implementations of BERT4Rec are able to replicate the originally reported results when trained with their default configurations. Table 3.4 compares the results achieved by the models trained with their default configuration with the results reported in the original BERT4Rec paper [230]. As we can see from the table, both versions of our HuggingFace-based implementation replicate the reported results on 3 datasets out of 4 (all except Beauty) for the popularity-sampled Recall@10 metric and on 2 datasets out of 4 for the popularity-sampled NDCG@10 metric. Moreover, BERT4Rec-VAE replicates the originally reported results on 2 datasets for the popularity-sampled Recall@10 metrics (both versions of MovieLens) and one dataset for the NDCG@10 metric (ML-1M). However, the original and RecBole implementations of BERT4Rec fail to replicate the originally reported results on all four datasets by a large margin (e.g. the original implementation is 46.47% worse according to Recall@10 metric on Steam).

The table also reports a comparison of the BERT4Rec implementations with the two baseline recommender models. As we can see, the best version of BERT4Rec always statistically significantly (according to the 2-tailed t-test with Bonferroni multiple test correction, $pvalue < 0.05$) outperforms both MF-BPR and SASRec on all four datasets and on both sampled and unsampled metrics. BERT4Rec-VAE performs better than the baselines on all four datasets, and the Ours and Ours (longer seq) versions of BERT4Rec perform better than baselines on three out of four datasets (except Beauty). SASRec performs better than the original and RecBole implementations on three datasets out of four datasets (except Beauty), which echoes the inconsistencies identified in our systematic review. Further, comparisons of the RecBole and original implementations with MF-BPR on Steam and ML-20M can lead to different conclusions if we look at sampled or unsampled metrics. For example, according to sampled Recall@10, the RecBole implementation is 33% worse than MF-BPR on the Steam dataset; however, at the same time, it is 86% better than MF-BPR according to full Recall@10. This discrepancy is in line with the results reported in [23, 40, 117] and shows the importance of using unsampled metrics in the research.

From the table, we also see that there is a big discrepancy in training times.[10] For example, on the ML-1M dataset, training BERT4Rec-VAE takes only 18 minutes, whereas training the RecBole version takes 5.6 hours. At the same time, BERT4Rec-VAE achieves 2.3 times better performance than RecBole, according to unsampled Recall@10.

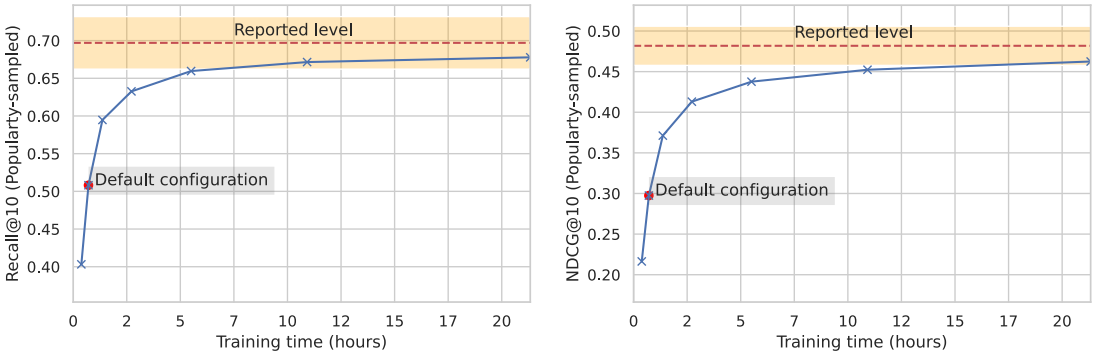
Unfortunately, we failed to replicate results reported in [230] for the Beauty dataset. In attempts to achieve originally reported results, we also ran experiments with the original implementation with a configuration tuned for this specific dataset, tried training the original model for up to 16x more training time, and tried using the evaluation framework and metrics from the original codebase. None of these attempts allowed us to reach metric values significantly better than we obtained from the BERT4Rec-VAE implementation, which is 22.68% worse than the reported values.

Overall, the answer to the RQ3.1 depends on the implementation and on the dataset. According to popularity-sampled Recall@10, we can replicate the originally reported results with our implementation in 3/4 cases, in 2/4 cases using BERT4Rec-VAE and in 0/4 cases using original and RecBole implementations. Furthermore, for unsampled metrics, we can observe similar conclusions: BERT4Rec models that fail to replicate the originally reported (sampled) results also performed poorly on unsampled metrics.

3.5.2 RQ3.2: Effects of training time on the original BERT4Rec implementation performance

Sun et al. [230] did not report the amount of training they needed to reach the state-of-the-art metrics. The amount of training specified in the original BERT4Rec code is controlled by the parameter *training steps*, which is set to 400,000 by default. To understand the effect of training time, we vary the number of training steps from slightly less than the default (200,000, 0.5x compared to the default) to much larger (1,280,000, 32x compared to the default), which results in a variation of training time from 23 minutes to 21.3 hours. Figure 3.2 portrays the relationship between training time and test performance of the original BERT4Rec implementation on the ML-1M dataset using popularity-sampled metrics.[11] As can be observed from the figure, it is

10. We report training times on our hardware configuration: 16 (out of 32) cores of an AMD Ryzen 3975WX CPU; NVIDIA A6000 GPU; 128GB memory.
11. We use popularity-sampled metrics because we need to compare results with the values reported by Sun et al. [230]



(a) Effect on popularity-sampled Recall@10 **(b)** Effect on popularity-sampled NDCG@10

Figure 3.2: Effect of the training time on the performance of the original implementation of BERT4Rec. The dashed line represents the metric value reported in [230], and the filled area represents $\pm$5% interval around the reported value, within which we count the result as a "replication" of the originally reported result. Default configuration corresponds to the results with the parameters specified in the original repository for the ML-1M dataset.

possible to reproduce the originally reported results using the original BERT4Rec implementation. However, it requires much more training time than the default configuration. Indeed, the model needs to be trained for almost 11 hours instead of 42 minutes to replicate (within $\pm$5% tolerance interval) results on sampled Recall@10 (15$\times$) and 21 hours to replicate results on sampled NDCG@10 (30x times). Looking back at Figure 3.1, the effect of training time on the unsampled version of NDCG follows the same general trend: the performance that the model reaches with the default configuration is 51.8% lower compared to what can be reached with a 30x increased amount of training.

Overall, in answer to RQ3.2, we find that the number of training steps set in the default configuration of the original BERT4Rec code is too small, and to reach performance levels reported by Sun et al. [230], the training time has to be increased up to 30$\times$ (from less than an hour to almost a day of training on our hardware). This change makes the model much harder to train and, for example, makes hyperparameter tuning much less feasible, specifically with a limited amount of hardware. It also makes it very easy to use an underfitted version of BERT4Rec in the experiments mistakenly. This likely explains the inconsistencies observed in our systematic review.

Table 3.5: Comparison of the Transformer models from the HuggingFace library. All results are reported on full (unsampled) metrics. The percentage shows the relative difference with BERT4Rec. **Bold** denotes the best model by a metric,† denotes statistically significant difference compared with BERT4Rec according to a paired t-test with Bonferroni multiple testing correction ($pvalue < 0.05$).

Model	Recall@10	NDCG@10	Training Time
BERT4Rec	0.282	0.151	2,889
DeBERTa4Rec	0.290 (+3.0%)	0.159 (+2.3%)	12,114
ALBERT4Rec	**0.300 (+6.4%)**†	**0.165 (+9.2%)**†	12,453

3.5.3 RQ3.3: Other Transformers from HuggingFace for Sequential Recommendation

To answer our final research question, we compare our BERT4Rec implementation with two other models: DeBERTa4Rec (based on the DeBERTa [120] architecture) and ALBERT4Rec (based on the ALBERT [72] architecture). Both architectures propose improvements to the BERT architecture (some of the improvements we describe in Section 3.4.3), and both models were shown to outperform BERT on the GLUE benchmark in their native natural language processing domain. Our goal is to experiment if these improvements hold in the domain of sequential recommendations. Our Hugging Face-based implementation makes such an experiment very simple, as in the Transformers library, these models have compatible interfaces with BERT. We also apply the same early stopping mechanism as we use in our version of BERT4Rec (see Section 3.4.3).

The results for these new models on the ML-1M dataset are listed in Table 3.5. From the table, we observe that both models are better to outperform BERT4Rec, however only for ALBERT4Rec are the results statistically significant (+6.48% Recall@10, +9.23% NDCG@10). These improvements allow us to conclude that the answer to RQ3.3 is positive - our implementation of BERT4Rec can further other models available in the Hugging Face Transformers library. Overall, there are more than 100 model architectures available in the Transformers library, and many of these models can be used seamlessly instead of BERT with our BERT4Rec implementations.

3.6 Related Work & Discussion

As observed in Section 4, there has been some difficulty in replicating the BERT4Rec reported results in [230]. In terms of related work, we highlight Dacrema et al. [55], who showed that deep learning-based recommendation papers frequently used weak configurations of baselines. They demonstrated this on an example of simpler matrix factorisation baselines; however, in this work, we show that this problem also exists with more complicated (and effective) models. Chin et al. [31] showed that the dataset selection could also influence the conclusion about model performance, which echoes the results of our systematic review. Krichene et al. [117] showed that conclusions about model performances may change when sampled or unsampled metrics are used – this was recently specifically confirmed for sequential recommendation [40]. Our experiments also confirm these results: for example, on the MovieLens-1M dataset, a Matrix Factorisation model performs 12% better than the RecBole implementation of BERT4Rec according to the sampled version of Recall@10, but at the same time it is 30% worse according to the unsampled version of Recall@10. However, none of the above-mentioned works performed a systematic review of one popular technique nor examined the reasons why that technique could be difficult to replicate. Nevertheless, the answers to our research questions hold in both sampled and unsampled versions of the metrics.

Our results for RQ3.1 and RQ3.2 in Section 3.5 above suggest that available BERT4Rec implementations are the cause of this difficulty, probably due to underfitting. Indeed, some of the recent papers that seemingly have used an underfitted BERT4Rec as a baseline for evaluation on the ML-1M dataset are listed in Table 3.6. All these papers used unsampled metrics for evaluation so that we can compare their reported values with fully-trained BERT4Rec results. As we can see from the table, the difference with our fully fitted version of BERT4Rec ranges from -10% to -53%, the results that are in line with the ones we observe when training BERT4Rec implementations with default configurations. Moreover, the results reported for the best models in these papers are rather worse than those we observe for a fully-trained BERT4Rec (GRU4Rec+ -20.59%, LightSANs -19.03%) or only marginally better (NOVA-BERT +1.55%, DuoRec +4.43%) models. Furthermore, as shown in Section 3.5.3, similar or even better improvements can be achieved by a simple replacement of one Transformer with another. According to this table, ALBERT4Rec improves the state-of-the-art result (+2% Recall@10 compared to DuoRec). However, this improvement needs to be verified via direct comparison with statistical significance testing rather than comparing reported numbers. We leave these experiments, as well as experiments on more datasets and with other Transformer architectures available in the HuggingFace Transformers Library, for future research.

Table 3.6: BERT4Rec results and best model results reported in the literature for the ML-1M dataset. The results are copied from the respective publications. All results are reported on full (unsampled) metrics. The percentage shows the difference with our implementation of BERT4Rec. **Bold** denotes the best reported BERT4Rec result and the best result overall.

Publication	BERT4Rec Recall@10	Best model	Best model Recall@10
Our Implementation	**0.282 (+0.0%)**	ALBERT4Rec	**0.300 (+6.4%)**
Dallmann et al. [40]	0.160 (-43.2%)	GRU4Rec+	0.224 (-20.5%)
Qiu et al. [189]	0.132 (-53.1%)	DuoRec	0.294 (+4.4%)
Fan et al. [52]	0.221 (-21.6%)	LightSANs	0.228 (-19.0%)
Liu et al. [140]	0.252 (-10.5%)	NOVA-BERT	0.286 (+1.5%)

3.7 Conclusions

In this chapter, we conducted a systematic review of the papers comparing BERT4Rec and SASRec and found that the reported results were not consistent. To understand the reasons for the inconsistency, we analysed the available BERT4Rec implementations. We found that, in many cases, they fail to replicate originally reported results when trained with their default parameters. Furthermore, we showed that the original implementation requires much more training time compared to the default configuration in order to replicate the originally reported results. This gives weight to the argument that, in some cases, papers have used underfitted versions of BERT4Rec as baselines.

We also proposed our own implementation of BERT4Rec based on the HuggingFace Transformers library, which, in most cases, replicates the originally reported results with the default configuration parameters. We showed that our implementation achieves similar results to the most recent sequential recommendation models, such as NOVA-BERT and DuoRec. We also showed that our implementation could be further improved by using other architectures available in the HuggingFace Transformers library. As our implementation is more efficient than the original BERT4Rec code, we use it in the rest of the thesis. We believe that this chapter, as well as our openly available code, will help the researchers use appropriately trained baselines and will move the science in the right direction.

Our analysis in this paper also raises an important question *why* BERT4Rec is more effective than SASRec. In the original BERT4Rec publication, Sun et al. argued [230] that the effectiveness gains are because (i) the bidirectional architecture of BERT4Rec is better suited for sequential recommendation than SASRec's unidirectional architecture and (ii) the use of the item masking task instead of sequence shifting. However, in the coming chapters, we show that the differences in effectiveness can be mostly attributed to *how* these models are trained. Indeed, in Chapter 4, we show that while BERT4Rec's item masking task has advantages over SASRec's item masking, it is possible to design another training task that is suitable for SASRec while allowing to train models faster. In Chapter 5, we further show that most of the effectiveness gains of BERT4Rec can be attributed to the absence of negative sampling, and when controlled for negative sampling, the difference between model's effectiveness is not always in favour of BERT4Rec.

We now turn to Chapter 4, where we address the training efficiency of Transformer-based Sequential Recommendation models.

Chapter 4

Effective and Efficient Training for Sequential Recommendation using Recency Sampling

In Chapter 3, we showed that existing Transformer-based Sequential Recommendation Systems exhibit a tradeoff between effectiveness and training efficiency. Indeed, for example, on the ML-1M dataset, our version of BERT4Rec is 41% more effective according to the Recall@10 metric than SASRec but requires 9.14× more training time. On the other hand, as we state in Limitation L2.1 in Section 2.3.5, training time is critical in real-world deployments of Transformer-based Sequential Recommender Systems. Indeed, unlike language modelling, where different tasks are based on the same language, recommendation datasets contain different sets of items; hence, the pre-training/fine-tuning paradigm is not available, and the models have to be trained from scratch every time, making training efficiency a critical factor for ensuring that Transformer-based Sequential Recommender Systems can be effectively deployed in real-world applications.

Hence, in this chapter, we analyse whether or not it is possible to train an effective model if the training time budget is limited. To achieve the goal of effective and efficient training, we propose a novel training objective, *Recency Sampling of Sequences* (RSS). RSS, allows us to achieve BERT4Rec-level effectiveness within an hour of training – comparable to SASRec in efficiency but matching BERT4Rec in effectiveness.

This chapter is organised as follows: Section 4.1 motivates the need for an efficient and effective training objective. Section 4.2 provides an overview of existing training objectives and identifies their limitations. In Section 4.3, we design the RSS training objective. Section 4.4 outlines the experimental setup for this chapter. Section 4.5 analyses the results of the experiments. Section 4.6 summarises the chapter and provides concluding remarks.

The material of this chapter is based on our full research paper [176], which was published in the proceedings of the ACM RecSys'22 conference, and its extension, journal paper [181], which was published in the special issue "Highlights of RecSys'22" of the ACM Transactions on Recommender Systems journal.

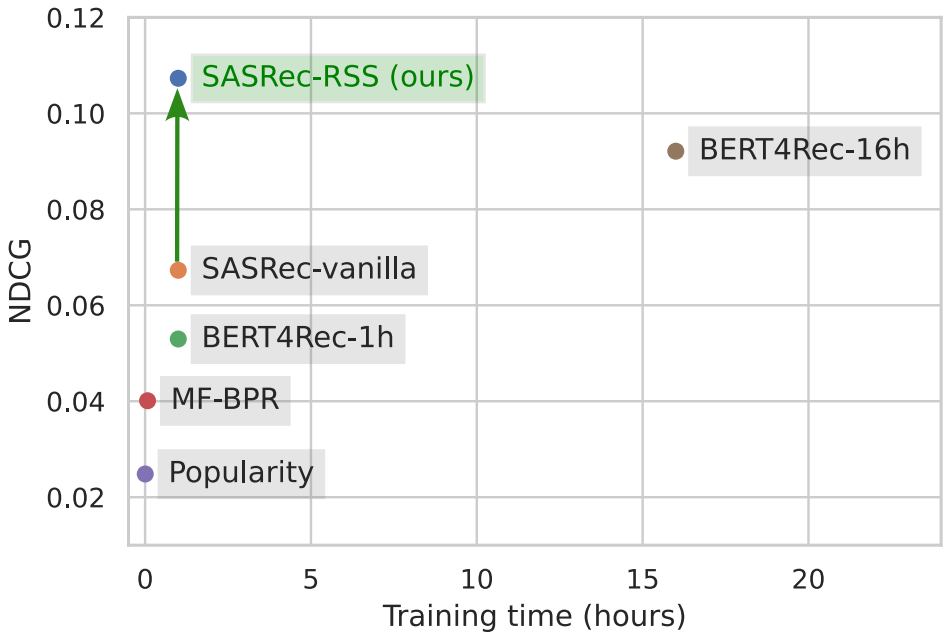


Figure 4.1: The SASRec [100] model trained with our proposed training method outperforms BERT4Rec [230] on the MovieLens-20M dataset [71] and requires much less training time. SASRec-vanilla corresponds to the original version of SASRec; BERT4Rec-1h and BERT4Rec-1h are versions of original BERT4Rec implementations that have been trained for 1 hour and 16 hours, respectively.

4.1 Need for Effective and Efficient Training

As we discussed in Chapter 3, the most advanced sequential models, such as BERT4Rec, suffer from a slow training problem. Indeed, our experiments in Chapter 3 show that in order to reproduce the result reported in the original publication, BERT4Rec requires more than 10 hours of training using modern hardware. This is also illustrated in Figure 4.1, which portrays the NDCG@10 of MF-BPR [199], SASRec [100] and BERT4Rec [230] models for different training durations on the MovieLens-20M dataset [71]. Indeed, as shown in Figure 4.1, on MovieLens-20M dataset BERT4Rec trained for 16 hours achieves higher NDCG@10 than SASRec (0.092 vs. 0.067), but at a significantly higher training cost (16 hours vs 1 hour). On the other hand, when we limit training time of BERT4Rec by 1 hour, it becomes less effective than SASRec (NDCG10 0.053 vs. 0.67). Hence, BERT4Rec is an effective model, but its training is slow.

Slow training is a problem in both research and production environments. For research, slow training limits the number of experiments that researchers can run using available computational resources. In production, slow training increases the costs of using recommender systems due to the high running costs of GPU or TPU accelerators. Furthermore, slow training hinders how quickly the model can be retrained to adapt to changing user interests. For example, when a new

episode of a popular TV show is released, the recommender system might still be recommending the old episode because it has not finished retraining yet. Hence, in this chapter, we focus on the time-limited training of models. The main question we address in this chapter is *can the training of existing sequential recommendation models be improved so that they attain state-of-the-art performance in limited training time?*

The primary components of model training can be characterised as follows: (i) the model architecture that is being trained, (ii) the training objective that defines what the model is being trained to reconstruct, and (iii) the loss function used to measure its success. All three components have a marked impact on training efficiency. For example, Hidasi et al. [81] showed that changing only the loss function can dramatically change the model performance. Raffel et al. [193] made similar findings for the model architecture and training objective for related tasks in language modelling. However, in this work, we focus on the training objective, identifying two key limitations in existing approaches, as well as an appropriate loss function for the objective.

Among the training objectives in the literature, there are two popular objectives for training sequential recommenders: *Sequence Continuation* and *Item Masking*. In this chapter, we show that both of these objectives have limitations.

First, *Sequence Continuation* [80, 81, 236] is probably the most intuitive and popular. Sequence Continuation splits the sequence into a prefix and a suffix, and then a model is trained to recover the suffix given the prefix. This objective closely aligned with our goal (predict the next interaction of the user). Given this alignment, Sequence Continuation objective arguably should result in optimal effectiveness when both training data and computational resources are unlimited. However, this objective never uses the beginning of the sequence as a training target. Hence, it discards potentially valuable knowledge and limits the number of training samples it can generate from a single sequence, which is specifically problematic when the datasets contain a relatively small number of sequences.

Second, in the *Item Masking* approach – which is used by BERT4Rec (see Section 2.3.4) – the task of the model is to recover masked items at any position in the sequence, which is a much more general and complex task than the next item prediction. We argue that this training objective is only weakly related to the end goal of sequential recommendation. Indeed, we will show that, despite leading to better effectiveness, this more general training task requires considerable training time.

These limitations of the existing approaches motivate us to design a new *Recency-based Sampling of Sequences (RSS)* approach that probabilistically selects positives from the sequence to build training samples. In our approach, more recent interactions have more chances of being sampled as positives; however, due to the sampling process' probabilistic nature, even the oldest interactions have a non-zero probability of being selected as positives. The sampling probability distribution in RSS is controlled by the *recency importance function*, which may have different shapes, for example exhibiting exponential or power distributions. Depending on the shape, we say that the function belongs to the *exponential family* or the *power family* of functions.

Our experiments are conducted on four large sequential recommender datasets, and demonstrate the application of the proposed RSS approach upon three recent sequential recommendation model architectures (GRU4Rec, Caser and SASRec), when combined with both pointwise and listwise loss functions. We find that RSS improves the effectiveness of all three model architectures. Moreover, on all four experimental datasets, versions of RSS-enhanced SASRec trained for one hour can markedly outperform state-of-the-art baselines. Indeed, RSS applied to the SASRec model can result in a 60% improvement in NDCG over a vanilla SASRec and a 16% improvement over a fully-trained BERT4Rec model, despite taking 93% less training time than BERT4Rec (see also Figure 4.1). We also find that both exponential and power importance functions result in similar optimal sampling probability distribution after fine-tuning their shape to best fit the MovieLens-20M dataset.

Moreover, we run experiments to understand better *how* RSS changes the resulting learned recommendation model. In particular, RSS is based on the idea that recent interactions are more important than earlier ones. To check whether or not RSS enables models to learn this difference in practice, we analyse how it changes the learned models with respect to interaction recency. Fortunately, Transformer [247]-based Sequential Recommendation models, such as SASRec and BERT4Rec, encode positional information explicitly in the form of *positional embeddings* (recall Section 2.3.2). To understand how RSS changes the model, we perform a novel analysis by comparing the positional embeddings learned by the original and RSS-enhanced versions of the SASRec model. We show that compared to the original SASRec, the RSS-enhanced version successfully learns to distinguish recent and earlier positions.

Overall, this chapter makes the following contributions:

1. We identify limitations in the existing training objectives used by sequential recommendation models;

2. We propose Recency-based Sampling of Sequences (RSS), which emphasises the importance of more recent interactions during training;
3. We perform extensive empirical evaluations on four sequential recommendation datasets, demonstrating significant improvements over existing state-of-the-art approaches.
4. We propose a novel methodology for analysing position embeddings and utilise this methodology to empirically demonstrate that an RSS-enhanced version of SASRec learns to differentiate between recent and earlier positions. In contrast, the original SASRec fails to identify any differences.
5. We experiment with both exponential and power families of importance functions and show that despite differences, the optimal functions from both families are mostly indistinguishable;

We now discuss the existing training objectives and their limitations.

4.2 Training Sequential Recommendation Models

As discussed in Section 2.2, Sequential Recommendation is usually cast as the Next Item Prediction task. Consider a set of users U and items I. Each user $u \in U$ has a sequence of interactions $s_u = \{i_{u_1}, i_{u_2}, i_{u_3}...i_{u_n}\}$ where items $i_{u_\tau} \in I$ are ordered by the interaction time. The next item prediction task is defined as follows: given a sequence s_u, rank the items from I, according to their likelihood of being the Sequence Continuation $i_{u_{n+1}}$. This task corresponds to *Leave One Out* evaluation - hold out the last element from each user's interaction sequence and then evaluate how well a model can predict each held-out element.

As mentioned in Section 2.2.1, the best models for the next item prediction task are based on deep neural networks. Generally speaking, their training procedure consists of iterations of the following steps:

Steps to train a Sequential Recommendation Model

While the model is not converged:

1. Generate a batch of training samples, each with positive and negative items;
2. Generate predictions using the model;
3. Compute the value of the loss function;
4. Update model parameters using backpropagation.

We aim to improve the training of existing models, so step 2 is not within the scope of this chapter. Backpropagation (step 4) – e.g. through stochastic gradient descent – is a very general and well-studied procedure, and we follow the best practices used by the deep learning models, details of which we describe in Section 4.4. This leaves us with two essential parts of model optimisation - the generation of the training samples and the loss function. These two parts are not independent: a loss function designed for one training task does not always fit into another. For example, BPR-max loss (used by GRU4Rec$^+$ [80]) has an assumption of only one positive item per training sample and, therefore, is not applicable to a Sequence Continuation task with multiple positives, as used by Caser. Hence, a new training task requires the selection of an appropriate loss function. We further discuss some possible choices of the loss functions for our proposed method later in Section 4.3.3. In the following, we describe existing approaches to generate training samples and identify their limitations, a summary of which we provide in Section 4.2.2.

4.2.1 Generation of Training Samples

A training sample for a sequential model consists of three parts - the input sequence, positive items, and negative sampled items. Sequential recommender models [80, 81, 100, 236] treat ground truth relevance as a binary function; by definition, every non-positive item is negative. In practice, to make the training more tractable, many models only consider samples of negative items, identified using techniques such as random sampling [100, 199], in-batch negatives [81], or the negatives with the highest scores [269]. This chapter focuses on constructing positive samples. Negative sampling approaches are orthogonal to positive sampling and can be applied independently. Indeed, we do not use negative sampling in this chapter; instead, we discuss negative sampling in detail in Chapter 5. In the remainder of this section, we describe positive sampling strategies for sequential recommendations. Figure 4.2 illustrates Sequence Continuation (including its variant, Sequence Continuation with Sliding Window) and Item Masking, the most commonly used strategies, which we discuss in turn below.

Matrix factorisation methods (Section 2.1.2) use a straightforward *Matrix Reconstruction* training objective: for each user u and item i, the goal of the model is to estimate whether the user interacted with the item. This goal leads to a simple procedure for generating training samples - the training algorithm samples (user, item) pairs as inputs and assigns labels for the pairs

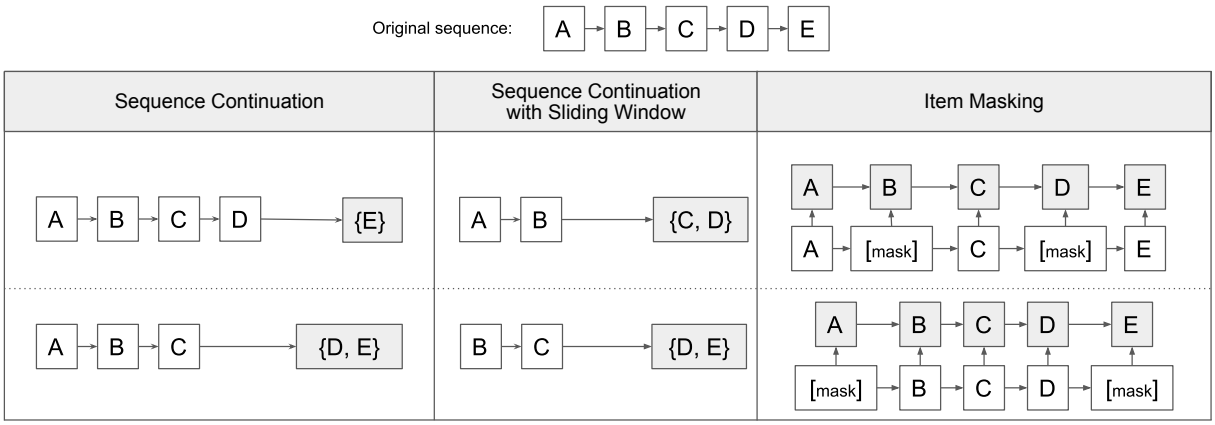


Figure 4.2: Training sample generation strategies used in existing models. White boxes represent model inputs, and filled boxes represent model outputs. In Sequence Continuation, the sequence is split into two parts, with the aim of predicting whether or not an item belongs to the second part based on the sequence of elements in the first part. In Sequence Continuation with a sliding window, we first generate shorter sub-sequences from the original sequence and then apply the Sequence Continuation method. In Item Masking, some elements are removed and replaced with a special "[mask]" value, with the aim of correctly reconstructing these masked items.

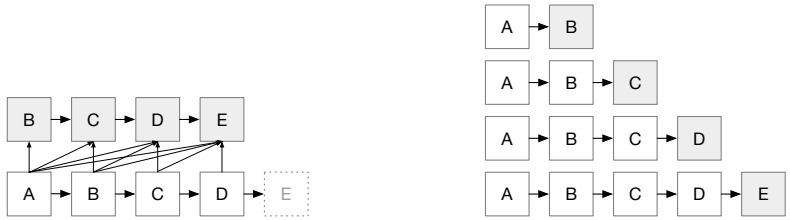
based on interactions. A classic model that uses matrix reconstruction is Bayesian Personalized Rank (BPR) [199], which we use as one of our baselines. The main disadvantage of matrix reconstruction is that it does not consider the order of the interactions, and therefore, Sequential Recommendation models can not use it.

In the *Sequence Continuation* training objective, training samples are generated by splitting the sequence of interactions into two consequent parts:

$$s = \{i_1, i_2, i_3..i_n\} \mapsto \begin{cases} s_{input} = \{i_1, i_2, i_3, \dots i_{n-k}\}; \\ s_{target} = \{i_{n-k+1}, i_{n-k+2}, \dots i_n\} \end{cases} \tag{4.1}$$

Where k is a hyperparameter. The model uses s_{input} as the input sequence and assigns label 1 to the positive items from $i_+ \in s_{target}$ and label 0 to the negative items $i_- \notin s_{target}$. If k is equal to 1, the Sequence Continuation task turns into the next item prediction task, which matches the end goal of sequential recommender systems.

Using Sequence Continuation in its basic form, we can produce precisely one training sample out of a single sequence of interactions. Some models (e.g. Caser [236]) use the sliding window approach to generate more than one sequence - which generates shorter subsequences out of a whole sequence and then creates training samples out of these shorter subsequences. The sliding window approach allows the model to generate up to $n - 1$ training samples from a se-



(a) Shifted Sequence training sample

(b) Sequence Continuation training samples equivalent to using Shifted Sequences

Figure 4.3: Equivalence of Shifted Sequence and Sequence Continuation training tasks. In Shifted Sequence, the model is trained to predict its output shifted by one element to the left. The models that use this training task only use elements in positions $0..i$ to predict i^{th} output. This is equivalent to n Sequence Continuation tasks, where n is the length of the sequence.

quence of n interactions. However, shorter sequences only allow modelling the short-term user preferences, and researchers have to find a balance between the number of generated samples and the maximum length of the sequence [236]. GRU4rec, GRU4rec$^+$, and Caser models use variations of the Sequence Continuation task for training. The *Sequence Shifting* training task used by SASRec and NextItNet [270] is essentially a version of Sequence Continuation: it trains the model to predict the target sequence, which is shifted by one element compared to the input. they These models predict the second element of the input sequence by the first, third by the first two, etc. When these models predict the j^{th} item in the output, they only have access to the first $(j-1)$ elements of the input so that this shifted sequence prediction task essentially is n independent Sequence Continuation tasks. Figure 4.3 graphically illustrates shifted sequence task and equivalent Sequence Continuation training samples. Thus, the main limitation of Sequence Continuation is that it only generates a small number of training samples out of a single sequence, and the items in the first part of the user's sequence never have a chance to be selected as a target, which means that the recommender system is unlikely to learn how to recommend these items, even though they may be relevant for some users. We refer to this limitation as Limitation L4.1.

In contrast to earlier neural sequential models, BERT4Rec [230] uses an *Item Masking* training objective, which it inherited from the original BERT [46] language model. In BERT, the idea is to hide some terms from the sentence and then ask the model to reconstruct these hidden elements. Similarly, in BERT4Rec, some items in the sequence are masked, and the model is retrained to recover these items. The target sequence, in this case, exactly matches the original sequence (without masking):

$$s = \{i_1, i_2, i_3, i_4, ..i_n\} \mapsto \begin{cases} s_{input} = \{i_1, [mask], i_3, [mask], \dots i_n\}; \\ s_{target} = \{i_1, i_2, i_3, i_4, \dots i_n\} \end{cases} \tag{4.2}$$

This approach generates up to 2^n training samples out of a single training sequence of length n. BERT4Rec does not mask more than $\gamma\%$ of items in a sequence, where γ is a hyperparameter (a typical value is $\gamma = 20\%$, see Table 3.3); however, it still generates many more training samples compared to the single training sample generated from a sequence under Sequence Continuation. As Sun et al. [230] showed, more training ensures to avoid overfitting and achieves better performance compared to other models with similar architecture.

However, we argue that the main disadvantage of the Item Masking approach is that it is weakly related to the next item prediction task. To make a prediction, BERT4Rec adds the [mask] element to the end of the input sequence and tries to reconstruct it so that training and evaluation samples have a different distribution. The model must learn how to solve the evaluation task (reconstruct the last item in the sequence) as part of a much more general and more complicated task (reconstruct any item in the sequence). BERT4Rec adds a small proportion of training samples with only the last element masked to address this mismatch. However, the consequence is still a substantially more complicated training task and a longer time to converge compared to the models that use sequence continuation. We refer to this problem of weak correspondence to the original task as Limitation L4.2.

4.2.2 Summary of Limitations

We described the two main training objectives used by sequential recommendations approaches: Sequence Continuation (including its variations, Sequence Continuation with sliding window, and Sequence Shifting) and Item Masking. Indeed, as argued above, both of these training objectives have their limitations, which we summarise as follows:

L4.1: Sequence Continuation can only generate a small number of training samples from a single training sequence. This allows training to be performed relatively quickly, but performance of these models is lower compared to a state-of-the-art model such as BERT4Rec.

L4.2: Reconstruction of masked items is a very general task, which is loosely connected to the sequential recommendation task. Using this task, models can reach state-of-the-art performance, but model training can take markedly longer than other training objectives.

In the next section, we design *Recency-based Sampling of Sequences*, a novel training task that addresses these limitations; the section also discusses possible choices of the loss function for this training task.

4.3 RSS: Recency-based Sampling of Sequences

As shown in Section 4.2, existing training objectives are rather ineffective (Sequence Continuation) or inefficient (Item Masking). To close the gap between the effectiveness and efficiency, in this section we design the Recency-based Sampling of Sequences (RSS) training objective, that probabilistically select targets from sequence using an recency importance function.

In this section, we introduce the RSS training objective (Section 4.3.1), two families of recency importance functions (Section 4.3.2) and choice of loss function. (Section 4.3.3). Later, in Section 4.3.4, we introduce a concept of position similarity matrix and describe how it can be used to analyse the effect of RSS on trained models. Section 4.3.5 provides a summary of the salient characteristics of RSS.

4.3.1 The RSS Training Objective

Recency-based Sampling of Sequences (RSS) is a training objective that is closely related to the sequential recommendations and allows the model to generate many training samples out of a single user sequence simultaneously. To address the limitations of existing training objectives described in Section 4.2.2, we first outline the principles used to design our training task:

P4.1: Each element in a sequence can be selected as the target; multiple items can be selected as a target in each training sample. Using this principle, we match the main advantage of the Item Masking approach - generating up to 2^n training samples out of each user sequence. This principle addresses Limitation L4.1.

P4.2: More recent training interactions in a sequence better indicate the user's interests, and hence, these are more realistic targets. User interests change over time, and one of the main advantages of sequential recommender systems is taking these changes into account. Therefore, the methods that rely on this principle will retain a close connection to sequential recommendations. This principle addresses Limitation L4.2.

In our proposed training objective, to follow these two principles, we use a *recency importance function*, $f(k)$, that is defined for each position $0 \,..\, n-1$ in the sequence of the length n and indicates chances of each position to be selected as a target: the probability of an item at position k of being selected as a positive is proportional to the value of $f(k)$. $f(k)$ must exhibit the following properties:

1. $f(k)$ is positive:

$$f(k) > 0 \tag{4.3}$$

2. $f(k)$ is monotonically growing:

$$f(k) \leq f(k+1) \tag{4.4}$$

This first property corresponds to Principle P4.1 and defines the likelihood of each item being selected as a target as positive. The second property corresponds to Principle P4.2 and ensures that more recent items have higher or equal chances of being selected as a target.

To generate a training sample, we first calculate c - how many target items we want to sample. Following BERT4Rec, we define a parameter γ that controls the maximum percentage of items that can be used as targets and then calculate c via multiplying γ by the length of the sequence. We then randomly sample, with replacement, c targets from the sequence, with the probability of being sampled, $p(i)$, proportional to the value of a recency importance function, $f(i)$:

$$p(i) = \frac{f(i)}{\sum_{j=0}^{n-1} f(j)} \tag{4.5}$$

Algorithm 1 Recency-based Sampling of Sequences

Input: $sequence$ - a sequence of interactions; γ - maximum percent of target items; f - recency importance function

Output: $input$ is a generated input sequence for the model; $target$ is a set of sampled positive items

function RECENCYSEQUENCESAMPLING($sequence$, γ, f)
$sampledIdx \leftarrow set()$
$n \leftarrow length(sequence)$
$c \leftarrow max(1, int(n * \gamma)))$
$prob \leftarrow Array[n]$
$prob[i] \leftarrow \frac{f(i)}{\sum_{j=0}^{n-1} f(j)}$ **for** i **in** [0,$n-1$]
$sampledIdx \leftarrow random.choice(range(0..n-1), c, prob)$
$input \leftarrow list()$
$target \leftarrow set()$
for $i \leftarrow 0, n-1$ **do**
if $i \in sampledIdx$ **then** $target.add(sequence[i])$ **else** $input.append(sequence[i])$
end for
return $input, target$
end function

We assume that function $random.choice(a, c, p)$ is an equivalent of the $numpy.random.choice$ function from the numpy python package. It iteratively samples c samples from collection a, where the probability of each item i of being sampled equals $p[i]$ at each stage, with replacement.

We generate the input sequence to the model by removing targets from the original sequence. The full procedure is described in Algorithm 1. We now describe two families of recency importance functions that have the required properties.

4.3.2 Recency Importance Functions

In this Section, we describe the properties of two families of recency importance functions, which can be used with RSS (i. e. they satisfy the requirements described by Equations (4.3) and (4.4)): *exponential importance* and *power importance*

4.3.2.1 Exponential Importance

Our first proposal for a recency importance function that has the required properties is the exponential function:

$$f_{exp}(k) = \alpha^{n-k} \tag{4.6}$$

Where $0 < \alpha \leq 1$ is a parameter that controls the importance of the recent items in the sequence, and n is the sequence length. If $\alpha = 1$, then each item has an equal chance of being sampled as a target, and Recency-based Sampling of Sequences becomes similar to the Item Masking approach (but without providing the positions of masked items) or to the matrix reconstruction approach, where items are sampled uniformly from the sequence. If α is close to zero, items from the end of the sequence have a much higher chance of being sampled, and therefore, RSS becomes equivalent to the Sequence Continuation task. Figure 4.4 provides an example of the recency importance (for $\alpha = 0.8$) and the generated samples.

Koren et al. [115] recently used a similar exponential function for modelling temporal dynamics in neighbourhood-based recommendation methods. Indeed, the authors successfully model a user-item interaction as a weighted sum of the other interactions of the same user, with the weight proportional to the exponential function:

$$e^{-\beta_u \Delta t} \tag{4.7}$$

where Δt is the time interval between the modelled and a known interaction, and β_u is a user-specific learnable parameter. In contrast to Koren et al. [115], we use the recency importance function for the target items selection instead of the explicit interaction similarity modelling. Nevertheless, inspired by their work, we use the exponential importance as the main approach throughout this chapter.

Figure 4.5a illustrates the sampling probability distributions generated by the exponential importance function for different values of the importance parameter α. As the figure shows, with the exception of the situation when $\alpha \approxeq 1$ (which corresponds to uniform sampling), exponential importance produces probability distributions that are very strongly skewed towards the most recent items. For example, even with $\alpha = 0.9$, the probability of sampling the item at position 10 in a sequence of 50 interactions is less than 0.002. Overall, the exponential nature of the function means that the probability of sampling always decays faster than linearly (wrt. to position k) for nearly all values of α. On the other hand, Ludewig and Jannach [144] success-

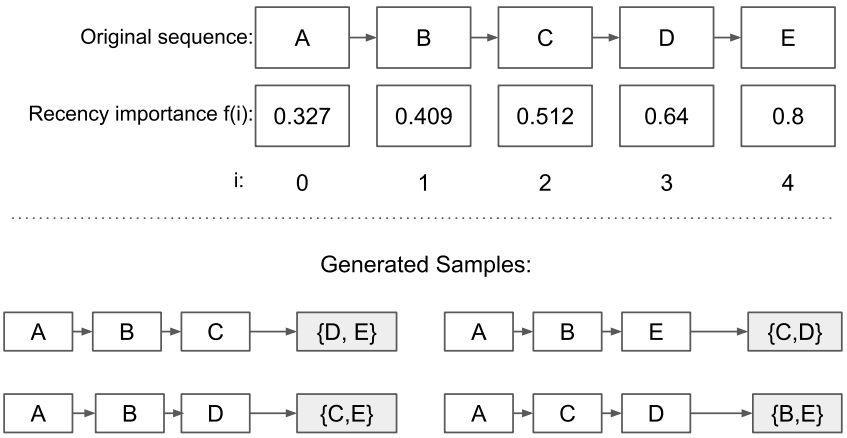
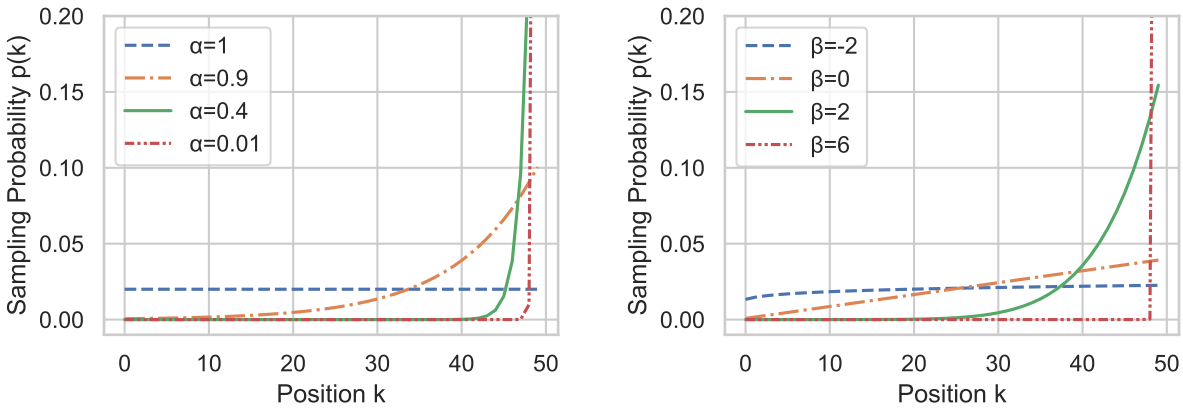


Figure 4.4: Recency-based Sampling of Sequences. The beginnings of the sequences remain largely unchanged, whereas elements from the end of the sequence are chosen as positive samples more frequently.



(a) Exponential Importance
$f_{exp}(k) = \alpha^{n-k}$

(b) Power Importance
$f_{pow}(k) = \left(\frac{k+1}{n+1}\right)^{e^{\beta}}$

Figure 4.5: Sampling probability distributions produced by exponential and power recency importance functions. Sequence length n is set to 50.

fully deployed a linear decay of position importance weights in the V-SKNN model. Similarity to the method proposed by Koren et al. [115] (mentioned above), V-SKNN uses these weights for explicit item-item similarity modelling rather than for computing target item sampling probabilities; therefore, it differs from RSS. However, the fact that V-SKNN used a linear decay of position similarity motivates us to investigate broader set recency importance functions, which are capable of generating linear and sub-linear decay of recency importance. Hence, we also experiment with a *power* importance function, which we describe in the next section.

4.3.2.2 Power importance

Another family of the importance functions we use for our experiments is the *power importance* function:

$$f_{pow}(k) = \left(\frac{k+1}{n+1}\right)^{e^{\beta}} \tag{4.8}$$

where n is the sequence length, k is the position and β is a hyperparameter. In this function, the position of the item in the sequence is raised to a constant power (controlled by the hyperparameter β). For example, when $\beta = 0$, this results in a linear function, when $\beta = -\infty$ the importance becomes constant (meaning that there is an equal chance of sampling any item in the sequence).

Figure 4.5b illustrates sampling probability distributions generated by the power importance functions with different values of the parameter β. Comparing the figure with Figure 4.5a, we can see that in contrast with the exponential importance, power importance can generate linear (when $\beta = 0$) and sub-linear (when $\beta < 0$) shapes of probability distributions.

Note that, because n and β are constants, the set of sampling probability distributions produced by this family is equal to the one produced by a simpler function:

$$\hat{f}_{pow}(k) = (k+1)^{\tau} \tag{4.9}$$

where $\tau = e^{\beta}$. Note that this simpler form omits the normalization coefficient $\frac{1}{n+1}$, because when computing a probability distribution in Equation (4.5), this coefficient is included in both numerator and denominator; hence it can be reduced. However, this normalisation is important in practice: without the normalisation, $\hat{f}_{pow}(k)$ becomes very large and numerically unstable even when k and τ have modest values. For example, for $k = 50$ and $\tau = 400$ (realistic numbers we use in our experiments), $\hat{f}_{pow}(k) = 51^{400} \approx 10^{683}$. This number is larger than the maximum value that can be represented with a standard *float32* data type and, therefore, causes errors during computations. By keeping the normalisation, we bring the value of the function into the $[0..1]$ interval and, therefore, help to avoid problems with infinities during computations.

The substitution of the parameter $\tau = e^{\beta}$ in Equation (4.9) helps to squeeze the area of the interesting hyperparameter values to a more symmetric and smaller interval. In particular, as we mention in Section 4.3.2.1, we are specifically interested in the cases when recency importance decays sub-linearly because faster-than-linear cases are covered by the exponential importance function. Without the substitution, we have the following cases for the behaviour of $\hat{f}_{pow}(k)$:

$$\begin{cases} \tau < 0 \rightarrow \text{out of the scope (violates the monotonic growth requirement (Equation 4.4))} \\ \tau = 0 \rightarrow \text{uniform sampling probability} \\ \tau \in (0, 1) \rightarrow \textbf{sub-linear decay of sampling probability} \\ \tau = 1 \rightarrow \text{linear decay of sampling probability} \\ \tau > 1 \rightarrow \text{faster-than-linear decay of sampling probability} \end{cases} \tag{4.10}$$

From these cases we can see that uniformly sampling of τ from the allowed range $\tau \in [0..+\infty]$ will almost certainly be sampled from the less interesting area of the faster-than-linear growth (with a probability of 1). Instead, by making the substitution $e^{\beta} = \tau$, we obtain a simpler and symmetric set of cases for the behaviour of $f_{pow}(k)$:

$$\begin{cases} \beta < 0 \rightarrow \textbf{sub-linear decay of sampling probability} \\ \beta = 0 \rightarrow \text{linear decay of sampling probability} \\ \beta > 0 \rightarrow \text{faster-than-linear decay of sampling probability} \end{cases} \tag{4.11}$$

Thus, a randomly sampled β has equal chances of producing faster-than-linear and sub-linear probability decay. In practice we chose β from the interval (-2..6): as Figure 4.5b shows, for sequences of length 50, $\beta = -2$ produces a sampling probability distribution close to uniform, whereas for $\beta = 6$ the probability of sampling the last item becomes close to one, and the probability of sampling earlier items is close to zero; RSS, therefore, becomes similar to Sequence Continuation.

In summary, we have proposed *exponential importance* and *power importance* families of importance functions, which can produce a broad set of shapes, including exponential, linear, and sub-linear shapes. Sections 4.5.3 and 4.5.4 cover our experiments with these two families of importance functions. In particular, in Section 4.5.4, we show that optimal shapes generated by both these families are very similar to each other.

4.3.3 Loss Functions for RSS

The second important component of the training procedure is the loss function. Loss functions for recommender systems can be generally divided into three categories - *pointwise* (optimise the relevance estimation of each item independently), *pairwise* (optimise a partial ordering between pairs of items) and *listwise* (optimise the recommendations list as a whole) losses [141]. RSS works with all types of loss functions that support multiple positive samples within each training sample.

GRU4rec$^+$ [80] showed the advantages of applying a listwise loss function above pointwise and pairwise methods; however, the Top-1-max and BPR-max losses introduced in that paper have an assumption that there is only one positive item within each training sample. Instead, we use LambdaRank [19] (denoted λRank), another listwise optimisation loss function. λRank has been widely deployed in training learning-to-rank scenarios [27, 188] for web search. Similarly, λRank has been shown to be advantageous for recommender tasks [128], for example, when applied to Factorization Machines [269] or Transformer-based sequential models [183].

λRank [19] uses λ-*gradients* instead of objective function gradients in the gradient descent method. According to Burges [19], the λ-gradient for an item $i \in I$ is defined as follows:

$$\lambda_i = \sum_{j \in I} |\Delta NDCG_{ij}| \frac{-\sigma}{1 + e^{\sigma(s_i - s_j)}} \tag{4.12}$$

where s_i and s_j are predicted scores, $\Delta NDCG_{ij}$ is the change that would be observed in an NDCG metric if items i & j were swapped, and σ is a hyperparameter defining the shape of the sigmoid, typically set to 1. Burges [19] used λRank to build one of the most successful learning-to-rank algorithms *LambdaMART*, which is based on gradient boosting trees. Versions of LambdaMART still produce state-of-the-art results for the learning-to-rank task [86, 188].

There are multiple available implementations of LambdaMART. In particular, Qin et al. [188] have shown that a version of LambdaMART implemented in a popular LightGBM library [104] performs better than other available implementations. Therefore, we matched our implementation of λRank with LightGBM's implementation. On analysis of the LightGBM source code, we found that it uses a normalised version of the λ-gradients (compared to Equation (4.12)). To keep our version consistent with the best available implementation, we include these modifications into our version of λRank. We provide more details on these modifications in Appendix A in the journal version of this work [181].

In addition to λRank, we also experiment with the pointwise Binary Cross-Entropy (BCE), following [100, 236], to investigate the effect of the listwise loss and the necessity of both the training objective and the loss function in our solution.

4.3.4 RSS and Positional Embeddings in Transformer-based models.

In this section, we describe the methodology for analysing the effects of RSS on the learned model using positional embeddings.

The main idea of RSS is that recent interactions are more important to the model compared to the older ones, and therefore, the model should treat them differently. To analyse whether or not this happens in practice, we need a mechanism that allows us to understand the learned differences between different positions in a sequence. Fortunately, the Transformer architecture (as used by SASRec) provides this mechanism in the form of positional embeddings (recall Section 2.3).

Our goal is to examine whether or not RSS helps the model distinguish between recent and earlier positions. To achieve this goal, we need to measure *position similarity* between different positions learned by the model. Intuitively, we would like the learned similarities to exhibit the following properties:

Pr.4.1: Similarities between nearby positions are **high**, as the nearby interactions are likely to be related to each other.

Pr.4.2: Similarities between distant positions are **low**; as distant interactions are unlikely to be related to each other.

Pr.4.3: For recent positions, similarity decays faster with the distance between positions than for the earlier positions. This means that the relative order of recent interactions is more important than the relative order of earlier interactions.

Property Pr.4.3 can be written as

$$similarity(i_r, i_r + \Delta_i) < similarity(i_e, i_e + \Delta_i) \tag{4.13}$$

where i_r corresponds to recent positions and i_e correspond to earlier positions.

SASRec explicitly separates interaction representations into item embeddings and position embeddings (see Equation (2.5)); this allows us to focus solely on the position representations. Indeed, as the item embeddings are independent of the positions, we can, therefore, exclude them from our analysis of position similarities.

After summation with the item embeddings, SASRec employs the position embeddings as input for linear projections. Therefore, it is reasonable to utilise the cosine similarity of positional embeddings as a means to measure the learned similarity between two positions in the sequence.

Formally, we define the learned position similarity matrix S, where the elements $s_{i,j}$ correspond to the learned similarity between positions i and j, and are defined as:

$$s_{i,j} = \frac{e_i \cdot e_j}{\|e_i\|\|e_j\|} \tag{4.14}$$

where e_i and e_j are positional embeddings learned by SASRec for positions i and j. Based on the definition, matrix S is symmetric with respect to the main diagonal, i.e.:

$$s_{i,j} = s_{j,i} \tag{4.15}$$

However, the symmetry with respect to the secondary diagonal is an undesirable property, as it violates Property Pr.4.3. Indeed, to show by contradiction that symmetry with respect to the secondary diagonal is undesirable, we assume it holds and demonstrate how this leads to a violation of Property Pr.4.3. Assuming the symmetry:

$$s_{i,j} = s_{n-j,n-i} \tag{4.16}$$

Let's denote $\Delta_i = j - i$, so

$$s_{i,i+\Delta_i} = s_{n-i-\Delta_i,n-i} \tag{4.17}$$

Equation (4.17) can be also written as

$$s_{i,i+\Delta_i} = s_{i',i'+\Delta i} \tag{4.18}$$

where $i' = n - i - \Delta_i$. If i is a recent position, ($i \approx n$), then i' corresponds to a early position ($i' \approx 0$). In that case Equation (4.17) contradicts Equation (4.13), which requires $s_{i,i+\Delta_i} < s_{i',i'+\Delta i}$. This means that a symmetric similarity matrix with respect to the secondary diagonal violates our desirable Property Pr.4.3. Ideally, we would like to avoid this kind of symmetry.

Overall, the position similarity matrix S allows us to analyse how the learned model treats interactions in different positions. For example, the original SASRec model is trained to predict original input shifted by one element. In this training task, every position in the sequence has equal importance, and therefore, we hypothetise that the similarity between positions only depends on the distance between positions, i.e.:

$$s_{i,j} \approx g(|i - j|) \tag{4.19}$$

which will lead to symmetry with respect to the secondary diagonal:

$$s_{i,j} \approx g(|i - j|) = g(|(n - j) - (n - i)|) \approx s_{n-j,n-i} \tag{4.20}$$

We also hypothesise that the RSS training objective helps the model avoid this undesirable symmetry. Indeed, by the design of the training objective, recent positions and their relative order are more important earlier positions in the sequence and their order, and therefore, we expect that the similarity $s_{i,j}$ between positions i and j depends on both absolute values of position and their relative distance between them:

$$s_{i,j} \approx g(i, |i - j|) \tag{4.21}$$

In summary, we hypothetise that SASRec trained with the RSS training objective exhibits all three desirable properties, while regular SASRec violates Property Pr.4.2. We experimentally analyse these hypothesis in Section 4.5.6

4.3.5 Summary

Overall, RSS is a novel training objective based on a probabilistic sampling of target items, with more probability assigned to recent items. Our motivation for the probabilistic target sampling objective encodes two principled intuitions for the sequential recommendation. RSS is model- and loss-agnostic: it can be used with various model architectures and loss functions. RSS is parametrised by the recency importance function, which defines the probability decay. Two examples of recency function families include exponential importance and power importance.

RSS only affects model training and leaves other model characteristics, such as the number of parameters or model throughput, unchanged. Hence, in the next section, we investigate the effects of RSS on model training through detailed experimentation.

4.4 Experimental Setup for Recency-Based Sampling of Sequences

In the following, we list our research questions (Section 4.4.1), our experimental datasets (Section 4.4.2), the recommender models on which we build, and our comparative baselines (Section 4.4.3), and finally, evaluation details (Section 4.4.4).

4.4.1 Research Questions

Our experiments aim to address the following research questions:

Resency-based Sampling of Sequences Research Questions

RQ4.1: Does Recency-based Sampling of Sequences (RSS) help for training sequential recommendation models compared to Sequence Continuation?

RQ4.2: Does a listwise λRank loss function benefit RSS training?

RQ4.3: What is the impact of the recency importance parameter α in the exponential recency importance function (Equation (4.6)) of RSS?

RQ4.4: What is the effect of the recency important function shape in RSS?

RQ4.5: How do RSS-enhanced models compare with state-of-the-art baselines?

RQ4.6: What is the effect of RSS on the positional embeddings learned by the SASRec model?

4.4.2 Datasets

Our experiments are performed on four large-scale datasets for sequential recommendation: MovieLens-20M [71], Yelp [8], Gowalla [32] and Booking.com [63]. We preprocess and split the datasets as described in Section 2.4.1.

These experimental datasets are quite distinct in terms of sequence length (see Table 2.1 in Section 2.4.1 for the salient characteristics of the datasets): median sequence length varies from 6 in the Booking.com dataset to 68 in the MovieLens-20M dataset. Indeed, our choice of datasets allows for the testing of RSS performance in settings with different sequence lengths.

We note that the Booking.com dataset contains sequences of cities within user's trips, which is a special case for the RSS approach. Indeed, in contrast to other types of recommendations, such as movies or books, multi-city trips have a strong sequential nature. Indeed, for example, if a user is making a road trip by car, there could be only one or two neighbouring cities where the user can stop, and hence all other more distant items are non-relevant. This strong sequential nature could be problematic for RSS, as it contradicts Principle P4.1, which says that any item in the sequence can be selected as a relevant target for the preceding items.

4.4.3 Models

4.4.3.1 Experimental Architectures

RSS is a training objective that can be used with a large variety of model architectures. We identify three large groups of model architectures that can be used with RSS: Recurrent Neural Network-based models, Convolutional Neural Network-based models, and Attention-based models. In each group, we select a well-cited representative architecture. Overall, we experiment using RSS with three recent model architectures for sequence recommendation:

1. **Recurrent Neural Networks:** *GRU4Rec* [81] is a sequential recommender architecture based on recurrent networks;
2. **Convolutional Neural Network:** *Caser* [236] applies a convolutional neural network structure for sequential recommendation. For our experiments, we use the basic architecture described in [270];
3. **Transformer (Attention network):** *SASRec* [100] is a sequential recommendation architecture based on Transformer (Section 2.3.2). The original implementation of SASRec is trained as a sequence-to-sequence model; however, only the final element from the target sequence is used at inference time. In order to match our common training framework and train the model with the RSS training objective, we ignore all outputs of the architecture except the final one. This is a notable change in the training process because the original SASRec computes its loss over all outputs. To make sure that this change does not lead to significant quality degradation, we include the original version of SASRec as a baseline (see Section 4.4.3.2).

We implement[1] these architectures using TensorFlow [1]. Note that for our experiments, we reuse only the architectures of these models and not the training methods or hyperparameters of these methods described in their original papers. Indeed, because our goal is to research the impact of the training task, the appropriate training parameters may differ from the original implementation.

---

1. The code for this Chapter is available at `https://github.com/asash/bert4rec_repro`

We implement RSS and Sequence Continuation training objectives on each of the three experimental architectures. We do not apply the Item Masking training objective with these architectures: Item Masking assumes that a model produces a score distribution per masked item, which is not compatible with those architectures; however, as discussed below, we include BERT4Rec as an Item Masking baseline.

For our experiments, we set common training parameters for all model architectures, following the settings in [100]. In particular, we set the size of the item embeddings to 64, we use the Adam optimiser, applying the default learning rate of 0.001 and following SASRec [100], we set the β_2 parameter, which controls the decay rate of the second moment in Adam, to 0.98[2].

Following the SASRec paper [100], set the maximum sequence length to 50 and apply the padding/truncation of sequences as described in Section 2.4.1. We select target items for both RSS and Sequence Continuation before applying padding/truncation.

Following BERT4Rec [230], we set the maximum percentage of a sequence to mask, γ, to 20%. Except where otherwise noted, we deploy the exponential recency function and set the recency parameter α to 0.8. Finally, in order to estimate the performance of the models under limited training time, we fix the training time of all models to 1 hour (we provide details on the amount of training data used for training of each model within the 1-hour limit in Appendix B of the journal version of this work [181]). Experiments are conducted using 16 cores of an AMD Ryzen 3975WX CPU, 128GB of memory, and an NVIDIA A6000 GPU.

4.4.3.2 Baselines

In order to validate that using RSS makes it possible to achieve performance comparable to state-of-the-art recommender models, we compare it with a selection of popular and state-of-the-art recommenders. We use the following non-neural models as baselines: (i) *Popularity* - the most popular items in the dataset; (ii) *MF-BPR* — Matrix factorisation with BPR Loss [199]. We use the implementation of this recommendation model from the popular LightFM library [119].

2. Note that the datasets used in the SASRec paper [100] are different from the ones we use in our experiments. However, the datasets used in [100] represent a wide range of data types, including e-commerce, games, and movie recommendations, and therefore we use these hyperparameters without changing them.

We also use two Transformer-based models as state-of-the-art baselines: (i) *SASRec-vanilla* — the original version of SASRec recommender [100], a Transformer-based model that uses a shifted sequence task, described in Section 4.2.1. To make the comparison fair with the RSS-enhanced variant, we limit the training time of this model to 1 hour; (ii) *BERT4Rec* is another Transformer-based model based on the BERT [46] architecture, which, as we have shown in Chapter 3, exhibits state-of-the-art effectiveness.

We use two versions of BERT4Rec: *BERT4Rec-1h* denotes where the training time of BERT-4Rec is limited to 1 hour to allow a fair comparison in a limited-time setting; *BERT4Rec-16h*, where training time is limited to 16 hours in order to compare the performance of our approach with the state-of-the-art model (recall in Chapter 3 we find empirically that reproducing the reported BERT4Rec results takes around 16 hours on our hardware [174]). We set the other parameters of BERT4Rec following the original paper [230].

In contrast with other baselines, BERT4Rec calculates a score distribution across all items in the catalogue for each element in the sequence. In contrast, other baselines calculate a single distribution of scores per sequence. This means that BERT4Rec requires $O(N)$ more memory per training sequence for storing output scores and ground truth labels compared to other baseline models. This makes training the original implementation of BERT4Rec infeasible when a dataset has too many items. Indeed, the original BERT4Rec publication [230] only reports results on relatively small datasets with no more than 55000 items, and our attempts to train BERT4Rec on a large Gowalla dataset with more than 1 million items failed because of memory and storage issues (see also Section 4.5.5). Hence, in this chapter, we do not use BERT4Rec for Gowalla (we will develop techniques to train BERT4Rec with large catalogues in Chapters 5 and 6).

4.4.4 Evaluation Measures

Following our common experimental setup (recall Section 2.4), we evaluate the models using the Leave-One-Out experimental strategy. We use the NDCG@10 and Recall@10 as our evaluation measures. To measure the significance of performance differences, we apply the paired t-test with Bonferroni multiple testing correction, following recommended practices in IR [56]. Following the recent guidance for the evaluation of recommender systems [67], our evaluation unit is a *user* (i.e. we measure statistical significance with respect to per-user results), and we use a significance level (or $pvalue$) of 0.05.

4.5 RSS Evaluation Results

We now analyse our experimental results for each of the four research questions stated in Section 4.4.1.

4.5.1 RQ4.1. The benefit of Recency Sampling

To address our first research question, we compare our experimental architectures (GRU4Rec, Caser, SASRec) trained with either Sequence Continuation or RSS objectives. Table 4.1 reports the effectiveness results in terms of Recall@10 and NDCG@10, of the three architectures, trained with both Sequence Continuation (denoted Cont) or RSS and applying two different loss functions (Binary Cross-Entropy – BCE – and λRank) on four datasets (MovieLens-20M, Yelp, Gowalla, Booking.com). Statistically significant differences – according to a paired t-test with Bonferroni multiple testing correction ($pvalue < 0.05$) – among the training objectives for a given architecture, model, and loss function are shown. On first inspection of Table 4.1, we note that the general magnitudes of the reported effectiveness results are smaller than those reported in [230] — indeed, as stated in Section 4.4.4, in contrast to [230], we do not use negative sampled metrics, instead preferring the more accurate unsampled metrics (see Section 2.4.2). The magnitudes of effectiveness reported for MovieLens-20M are in line with those reported by [40] (e.g. a Recall@10 of 0.137 for SASRec-vanilla is reported in [40] when also using a Leave-One-Out evaluation scheme and unsampled metrics).

We now turn to the comparison of training objectives. In particular, we note from the table that, on the MovieLens-20M, Yelp, and Gowalla datasets, RSS results in improved NDCG@10 in 17 out of 18 cases – 15 of which are by a statistically significant margin – and also improved Recall@10 in 16 out of 18 cases (15 statistically significant). For instance, on MovieLens-20M, SASRec is the strongest performing architecture (in line with the previous findings [100, 230]); however, applying RSS significantly improves its Recall@10, both when using BCE (0.153→0.188) and when using λRank (0.105→0.196). Similarly and interestingly, SASRec with the RSS objective and λRank loss outperformed other models by a very large margin on the Gowalla dataset (e.g. Recall@10 0.102 compared to 0.071 when using Sequence Continuation). We postulate that the large number of items in the dataset makes the training task very hard, and only the combination of RSS with λRank allows training the model with reasonable quality in the given time limit. The only instance when RSS is worse than Continuation on these three

Table 4.1: Comparing Sequence Continuation with Recency-based Sampling of Sequences training objectives under limited training for various model architectures. **Bold** denotes a more effective training objective for an (Architecture, Loss, Dataset) triplet. We use * to denote statistically significant differences compared to the other training objective (left vs. right), and † to denote significant differences in the change of loss function (upper vs. lower). All tests apply a paired t-test with Bonferroni multiple testing correction ($pvalue < 0.05$). The training time of all models is limited to 1 hour.

(a) Recall@10

		MovieLens-20M		Yelp		Gowalla		Booking.com	
Architecture	Loss	Cont	RSS	Cont	RSS	Cont	RSS	Cont	RSS
GRU4Rec	BCE	0.0221†	**0.0354***	0.0075†	**0.0100*†**	**0.0026***	0.0005	0.4621	**0.4962***
	λRank	0.0082	**0.1544*†**	0.0009	**0.0045***	0.0068†	**0.0119*†**	**0.4780†**	**0.5084*†**
Caser	BCE	0.1424†	**0.1866***	0.0046†	**0.0099*†**	0.0076	**0.0081**	**0.5600*†**	0.5454†
	λRank	0.0330	**0.1496*†**	0.0009	**0.0017***	0.0087†	**0.0157*†**	0.4968	**0.5273***
SASRec	BCE	0.1537†	**0.1888***	0.0146†	**0.0269*†**	0.0089	0.0089	**0.5845*†**	0.5178
	λRank	0.1050	**0.1968*†**	0.0045	**0.0052***	0.0715	**0.1020*†**	**0.5662***	0.52464†

(b) NDCG@10

		MovieLens-20M		Yelp		Gowalla		Booking.com	
Architecture	Loss	Cont	RSS	Cont	RSS	Cont	RSS	Cont	RSS
GRU4Rec	BCE	0.0115†	**0.0183***	0.0035†	**0.0049*†**	**0.0017***	0.0002	0.2829	**0.2899***
	λRank	0.0040	**0.0839*†**	0.0004	**0.0014***	0.0033†	**0.0067*†**	**0.3132*†**	**0.3093†**
Caser	BCE	0.0784†	**0.0995***	0.0021†	**0.0049*†**	0.0039	**0.0040**	**0.3665*†**	0.3311†
	λRank	0.0177	**0.0814*†**	0.0003	**0.0007***	0.0055†	**0.0100*†**	0.3181	**0.3226***
SASRec	BCE	0.0850†	**0.1002***	0.0076†	**0.0136*†**	0.0044	**0.0044**	**0.3633*†**	0.2966
	λRank	0.0579	**0.1073*†**	0.0021	**0.0025***	0.0478†	**0.0749*†**	**0.3623***	0.3122†

datasets is for the GRU4Rec architecture on Gowalla with BCE loss. This is also likely to be explained by the difficulty of the task on this dataset due to the large number of items. This is also reinforced by the difficulty of training GRU4Rec architecture in general (see also Section 4.5.3 and Figure 4.6).

We also note the three datasets where RSS performs well (MovieLens-20M, Yelp, and Gowalla) are very different in terms of sequence length (see Table 2.1 in Chapter 2), varying from median length eight on Gowalla to median length 68 on MovieLens-20M. This means that RSS can be effective with both short and long sequences. On the other hand, for the Booking.com dataset, we observe that in 3 out of 6 cases, RSS is less effective. This is not an unexpected result: as we argued in Section 4.4.2, this dataset violates the underlying assumption encoded in Principle P4.1. Indeed, due to the geographical distance between items in this multi-city trip dataset, items cannot be considered out-of-order. Hence, RSS does not improve the stronger models on this dataset.

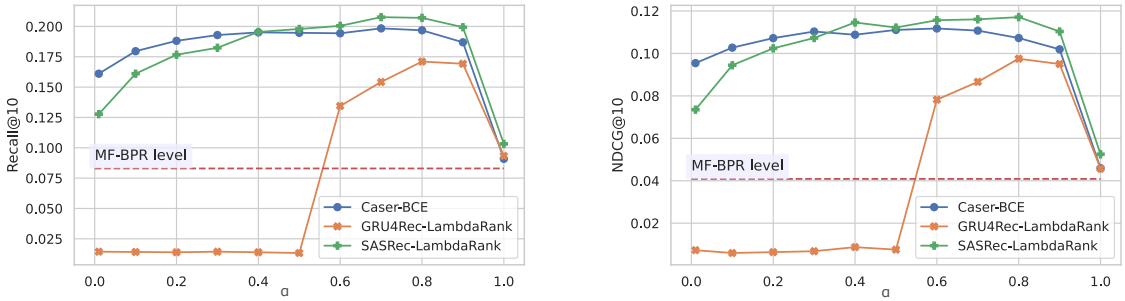
Table 4.2: Comparing RSS-enhanced SASRec with baseline models under limited training. **Bold** denotes the best model for a dataset by the metric in the main group, underlined the second best. Symbols * and † denote a statistically significant difference compared with SASRec-RSS-BCE and SASRec-RSS-λRank, respectively, according to a paired t-test with Bonferroni multiple testing correction ($pvalue < 0.05$).
[1] We do not report results for BERT4Rec models for the Gowalla dataset: due to the large number of items in this dataset, we were not able to train the model. [2] We report results for BERT4rec-16h separately due to its larger training time.

		MovieLens-20M		Yelp		Gowalla		Booking.com	
Model	Train time	Recall @10	NDCG @10	Recall @10	NDCG @10	Recall @10	NDCG @10	Recall @10	NDCG @10
Popularity	1h	0.049†*	0.025†*	0.006†	0.003†*	0.008*	0.004*	0.097†*	0.043†*
MF-BPR	1h	0.079†*	0.040†*	0.019†*	0.009†*	0.029†*	0.018†*	0.449†*	0.279†*
SASRec-vanilla	1h	0.136†*	0.067†*	0.022†*	0.011†*	0.010*	0.005†*	0.463†*	0.270†*
BERT4rec-1h	1h	0.107†*	0.053†*	0.014†*	0.007†*	N/A[1]	N/A[1]	0.479†*	0.288†*
SASRec-RSS-BCE	1h	0.189*	0.100*	**0.027***	**0.014***	0.009*	0.004*	0.518*	0.297*
SASRec-RSS-λRank	1h	**0.197†**	**0.107†**	0.005†	0.003†	**0.102†**	**0.075†**	**0.525†**	**0.312†**
BERT4Rec-16h[2]	16h	0.173†*	0.092†*	0.028*	0.014*	N/A[1]	N/A[1]	0.565†*	0.354†*

Overall, in response to RQ1, we conclude that Recency-based Sampling of Sequences improves model training if the items earlier in the user sequence can be treated as positives (properties exhibited by the MovieLens-20M and Gowalla datasets).

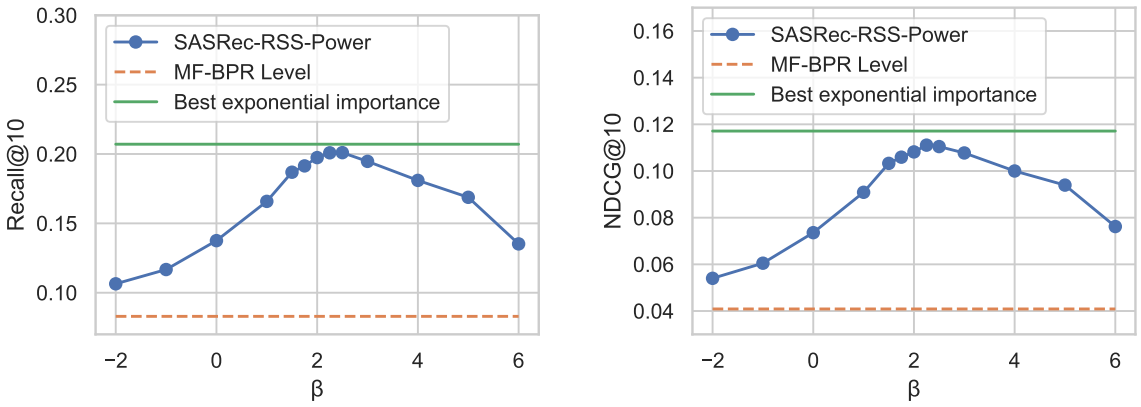
4.5.2 RQ4.2. Comparison of Different Loss Functions

Next, we address the choice of the loss function, as per RQ4.1. We again turn to Table 4.1, but make comparisons of the upper vs. lower performances in each group. For instance, for RSS, we observe that applying the listwise λRank loss function on the GRU4Rec architecture on MovieLens-20M dataset results in a significant increase (0.035→0.154), as denoted by the † symbol. Indeed, across all of Table 4.1, we observe that when used with RSS training task, λRank improves NDCG@10 in 8 cases out of 12 (all 8 significantly) as well as Recall@10 in 8 cases out of 12 (8 significantly). In contrast, λRank only improves over BCE in 7 out of 24 cases for the Sequence Continuation training objective (all by a significant margin). Overall, and in answer to RQ2, we find that λRank usually improves (except Yelp) the effectiveness of our proposed RSS training objective, while it does not offer the same level of improvement for Sequence Continuation. We explain this finding as follows: In Sequence Continuation, there is only one relevant item per sequence, and hence, the benefit of a listwise loss function is limited. In contrast, RSS selects multiple relevant items for each sequence, and in this case, a listwise loss function can benefit the model by training it to rank these items nearer the top of the ranking. However, as λRank did not improve RSS results on Yelp, we can not say that the improvements are consistent, and the question of the loss function selection requires further research.



(a) Effect on Recall **(b)** Effect on NDCG

Figure 4.6: SASRec, GRU4rec and Caser performance on the MovieLens-20M dataset, when trained with Recency-based Sampling of Sequences with the exponential importance function $f_{exp}(k) = \alpha^{(n-k)}$, where n is the sequence length. Position recency importance parameter α is plotted on the x-axis. When $\alpha = 0$, the training objective turns into Sequence Continuation, and when $\alpha = 1$, the task becomes similar to Item Masking or matrix reconstruction. The training time of all models is fixed at 1 hour.



(a) Effect on Recall **(b)** Effect on NDCG

Figure 4.7: SASRec-RSS performance when trained on MovieLens-20M dataset with power recency importance function: $f_{pow}(k) = \left(\frac{k+1}{n+1}\right)^{e^{\beta}}$ and variable parameter β.

4.5.3 RQ4.3. Impact of Position Recency Importance in the Exponential Importance Function

This research question is concerned with the importance of sampling recent items in training sequences. To address this question, we train every experimental architecture with the best-performing loss from Table 4.1 on the MovieLens-20M dataset. We vary the recency importance parameter α in the exponential recency importance function (Equation (4.6)), to investigate its effect on effectiveness. In particular, as $\alpha \to 0$, the training task turns to Sequence Continuation, while with a large α, the training task loses its sequential nature and becomes similar to matrix factorisation.

Figure 4.6 summarises the impact of α on the model effectiveness. We also present the performance of the MF-BPR [199] baseline. From the figures, we observe that when we set α close to zero, the results match those we report in Table 4.1 for the Sequence Continuation task, illustrating that under small α, RSS only samples the last element in each sequence. Similarly, for $\alpha = 1$, we observe that the effectiveness of all models drops almost to that of the matrix factorisation baseline, as target items are sampled from sequences without any ordering preference. Note that in this case, we sample target items uniformly, which is similar to BERT4Rec's Item Masking. However, BERT4Rec also has access to the positions of masked items (through the position embeddings), whereas in the case of $\alpha = 1$, the positional information is completely lost, and the model can not learn to predict the *next* item and predicts *some* item instead. Overall, the general trends visible in Figure 4.6 suggest that RSS allows the training of effective models across a wide range of the α parameter values: for Caser and SASRec, large improvements over Sequence Continuation training are achieved for $0.2 \leq \alpha \leq 0.9$; for GRU4Rec, strong performance is obtained $0.6 \leq \alpha \leq 0.9$. Indeed, for small α, the number of positive items is limited, and hence the lambda gradients in λRank are also small. This provides little evidence to the GRUs in GRU4Rec, which, therefore, struggles with the vanishing gradient problem (a problem faced by many such recurrent architectures).

Overall, in response to RQ3, we find that the higher values of the recency importance parameter $\alpha \leq 0.9$ results in effective performance for all three model architectures.

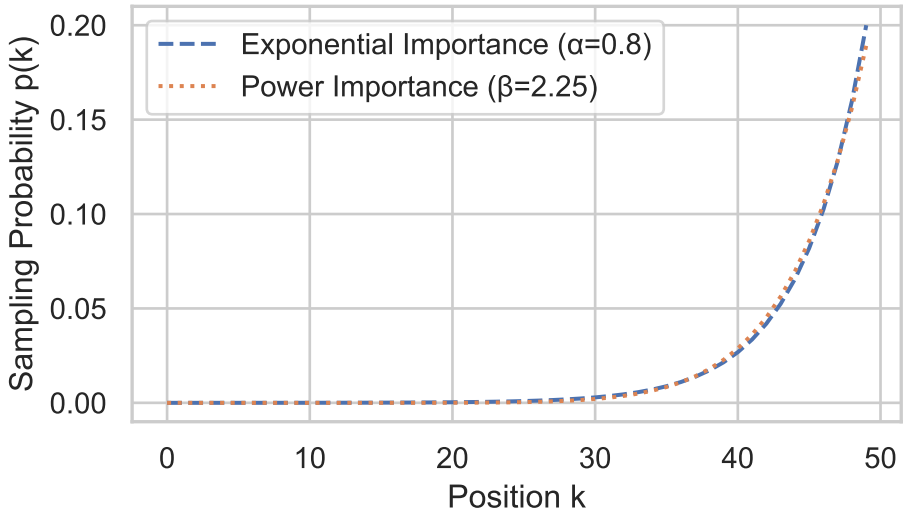


Figure 4.8: Optimal sampling probability distributions for SASRec-RSS model generated by exponential ($f_{exp}(k) = 0.8^{n-k}$) and power ($f_{pow}(k) = \left(\frac{k+1}{n+1}\right)^{e^{2.25}}$) recency importance functions. The plotted curves are mostly superimposed.

4.5.4 RQ4.4. Recency Importance Function Shape

We now analyse the effects of replacing the exponential importance function (defined in Equation (4.6)) with the power importance (defined in Equation (4.8)).

To understand whether or not this different family of distributions may improve overall recommendations quality compared to exponential recency importance, we train a SASRec-RSS model on the MovieLens-20M dataset with the Binary Cross-Entropy loss and power importance while varying the power hyperparameter β in the range from -2 to 6. Figure 4.7 shows the effect of this variation on the model performance in terms of NDCG and Recall metrics.

From the figure, we note that the model achieves the best performance when $\beta = 2.25$. Interestingly, the best-achieved Recall@10 of 0.2008 and NDCG@10 of 0.1110 are similar to the best results achieved by the model with exponential importance: it obtains Recall@10 of 0.2070 and NDCG@10 of 0.1171.

These similarities are not surprising when we look to Figure 4.8, which plots the sampling probability distributions corresponding to the optimal power recency function (with $\beta = 2.25$) and optimal exponential recency function (with $\alpha = 0.8$). As we can see from the figure, the shapes of the distributions are almost identical, which leads to similar model performance.

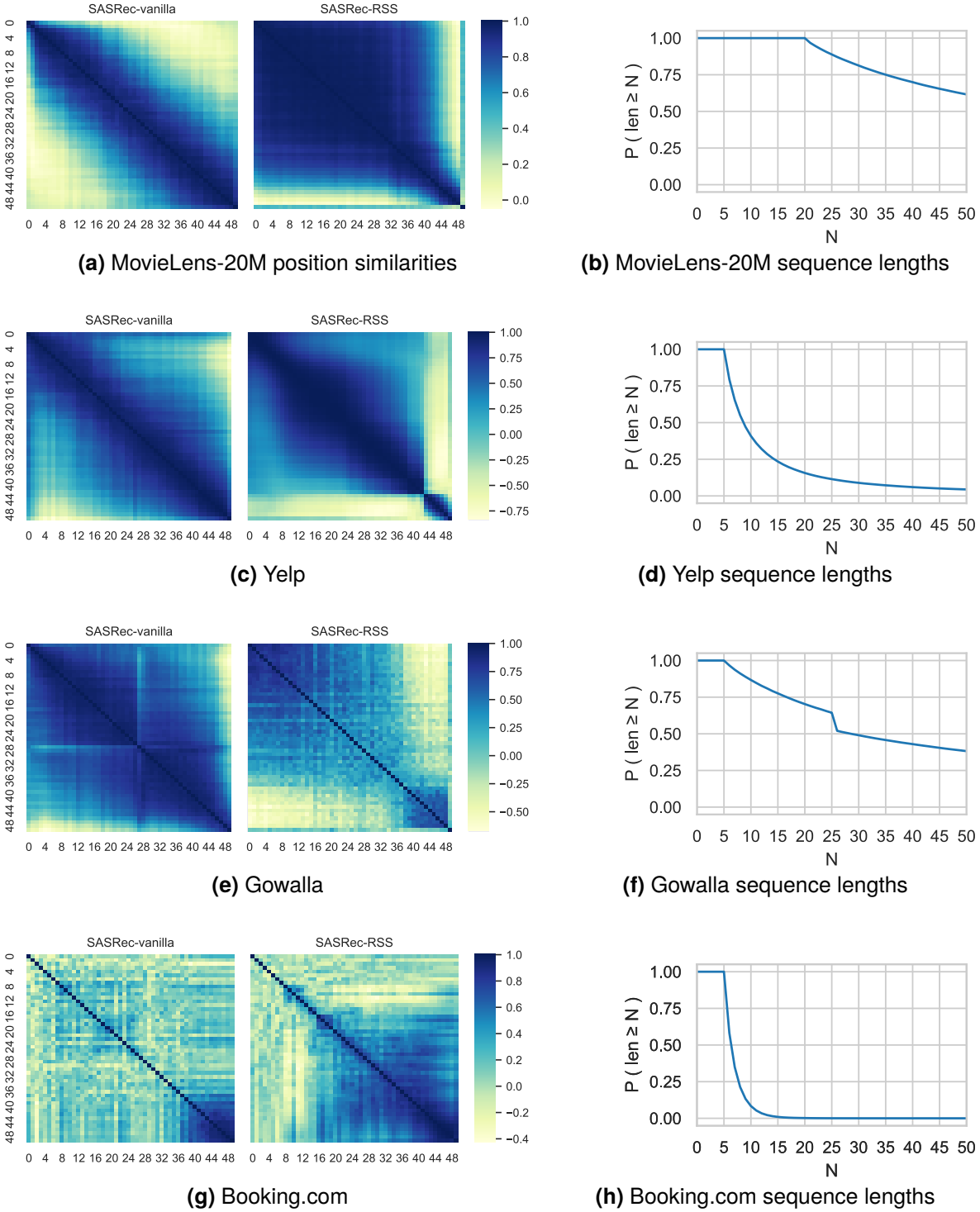
Overall, this analysis suggests that both power and exponential functions are viable alternatives, and if they are tuned properly, they are likely to produce similar sampling probability distributions. However, we find that the exponential importance function is easier to tune, as it exhibits high performance in a wide range of hyperparameter values. Overall, we recommend using the exponential function as a default importance function in RSS.

4.5.5 RQ4.5. Comparison with Baselines

To address our fifth research question concerning the comparison with baseline models, we compare the best-performing RSS-enhanced model, SASRec-RSS, using both λRank and Binary Cross-Entropy losses, with the 5 baseline models described in Section 4.4.3. Table 4.2 summarizes the results of this comparison, reporting effectiveness metrics as well as training time duration. In particular, recall that all models are trained for less than 1 hour, except for BERT4Rec-16h (a full training of BERT4Rec). Moreover, we did not train BERT4Rec on the Gowalla dataset because the preprocessing code for BERT4Rec does not scale to its large number of items (indeed, Gowalla has more items than users; see Table 2.1). Indeed, the preprocessing code to generate masked training sequences requires 14GB of storage for MovieLens-20M, but 548GB for Gowalla.

On analyzing Table 4.2, we observe that SASRec-RSS (λRank or BCE) achieves the most effective performance on all four datasets among the time-limited recommendation models. For instance, on the MovieLens-20M dataset, compared to the original formulation of SASRec (denoted SASRec-vanilla), the RSS adaptation significantly improves NDCG@10 (by the margin of 60%) for the same training duration. Moreover, compared to the 16 hour training of BERT4-Rec, SASRec-RSS exhibits 16% higher NDCG@10 (a significant improvement), despite needing only 6% of the training time (16h$\rightarrow$ 1h). For Booking.com, where RSS was less effective, SASRec-RSS with λRank objective obtains the NDCG@10 12% less than that obtained by the expensive BERT4Rec-16h model, and the Recall that is 7% less. Interestingly, on the Yelp dataset, the λRank version of SASRec is not effective (same performance as popularity baseline). Still, the BCE version of the model significantly outperforms all other models in the main group and achieves performance on par with BERT4Rec-16h. Furthermore, we see that in all cases where we are able to train BERT4Rec under limited training time, BERT4Rec underperforms compared to the SASRec-RSS (λRank or BCE version).

Overall, in answer to RQ4.5, we find that SASRec-RSS can achieve significantly higher effectiveness than the state-of-the-art SASRec and BERT4Rec approaches when trained for a comparable time. Furthermore, we can achieve performances exceeding or very close to a fully-trained BERT4Rec, but with much less training time. This highlights the importance of an appropriate training objective in general and the benefits of our proposed RSS training objective in particular.



(a) MovieLens-20M position similarities

(b) MovieLens-20M sequence lengths

(c) Yelp

(d) Yelp sequence lengths

(e) Gowalla

(f) Gowalla sequence lengths

(g) Booking.com

(h) Booking.com sequence lengths

Figure 4.9: Similarity matrices of positional embeddings

4.5.6 RQ4.6. Effect on Positional Embeddings

To better understand the impact of RSS on the resulting learned models, we analyse the similarities between the positional embeddings learned by SASRec when trained with both the shifted sequence training objective (SASRec-vanilla) and RSS. We compute the cosine similarity matrix S as defined by Equation (4.14) between the embeddings of each pair of positions and then use these matrices to validate desirable properties of positional embeddings, as described in Section 4.3.4.

The heatmaps shown in Figure 4.9 (left) graphically visualise these similarity matrices for all four experimental datasets. The darker colour on the figure corresponds to the higher similarity between positions, whereas the lighter colour corresponds to the lower similarity. Figure 4.9 (right) also shows the distributions of sequence lengths for these datasets, as these distributions help to explain some of the properties of the matrices: For example, in Figure 4.9e, we see that for the Gowalla dataset, artefact lines are appearing at position 26. As we can see from the corresponding sequence length distribution (Figure 4.9f), This corresponds to a sudden drop in the sequence length distribution in the dataset. Indeed, for the Gowalla dataset, only 52% of sequences are 26 items or longer, a drop from 64% for the sequences of length ≥ 25 - this important change in distribution is reflected in the learned embeddings. Similarly, for the Booking.com dataset, both SASRec-vanilla and SASRec-RSS mostly contain noise for early positions, with little correspondence between nearby positions. This can be explained by the fact that the Booking.com dataset has only a very small number of sequences longer than 15 items, as can be seen from the corresponding sequence distribution plot.

The overall general trends that can be observed in Figure 4.9 are as expected in that all matrices are symmetric with respect to the main diagonal (because of the symmetry of cosine similarity). The figures also show that the similarity is high in all cases, close to the main diagonal and lower further away from it. This means that both SASRec-Vanilla and SASRec-RSS successfully learned our desirable Properties Pr.4.1 and Pr.4.2 (close positions have similar positional embeddings; embeddings of distant positions are less similar) – as defined in Section 4.3.4.

We now look to the symmetry with respect to the secondary diagonal, which, as we argue in Section 4.3.4, is an undesirable property, as it violates Property Pr.4.3 (earlier positions are more similar to each other than the recent ones). As we expect, for each dataset except Booking.com, in SASRec-vanilla the similarity between a pair of positional embeddings is mostly defined by the distance between their respective positions (see also Equation (4.21)). As we showed in Section 4.3.4, this makes the matrix symmetric with respect to the secondary diagonal; therefore, SASRec-vanilla violates Property Pr.4.3.

In contrast, the similarity matrices for SASRec trained with the RSS training objective are not symmetric with respect to the secondary diagonal. Instead, for each dataset except Booking.com, we observe two groups of positions: a large group of earlier positions and a small group of recent positions. For example, on the MovieLens-20M dataset, we can say that positions 0–40 comprise the earlier group, whereas positions 41–49 comprise the recent group. Positions within each group tend to be similar to each other and different compared to the positions from the other group. In practice, this means that in contrast to SASRec-vanilla, the similarity matrices of SASRec-RSS do not violate Property Pr.4.3. This confirms that RSS helps models to make better distinctions between recent and earlier positions, as we hypothesised in Section 4.3.4.

Furthermore, from Figure 4.9g we can also see that position similarity matrices for the Booking.com dataset do not look like other similarity matrices. For example, similarity matrices for SASRec-vanilla trained on this dataset are not symmetric with respect to the secondary diagonal. The shorter sequence lengths in the Booking.com dataset can explain this. As we can see from the figure, there are virtually no sequences with more than 20 interactions in this dataset, which differs from the other datasets in our experiments (even for the Yelp dataset, where the average sequence length is relatively short, approximately 4% of all sequences have at least 50 interactions). According to our applied padding scheme (see Section 2.4.1 and Figure 2.10), this means that positions on the left side of the sequence will be padded and ignored by the model. As a result, the positional embeddings for earlier positions remain mostly unchanged after random initialisation in the model, which explains the differences in similarity matrices with the other datasets.

In summary, we conclude that while both SASRec-vanilla and SASRec-RSS models successfully learn Properties Pr.4.1 and Pr.4.2, however, only an RSS-enhanced model successfully learned Property Pr.4.3 on three out of four datasets.

4.6 Conclusions

In this chapter, we identified two limitations in existing training objectives for sequential recommender models. To address these two limitations, we proposed a refined training objective, called Recency-based Sampling of Sequences (RSS). Through experimentation on four datasets, we found that this relatively simple change in the training objective can bring significant improvements in the overall effectiveness of state-of-the-art sequential recommendation models, such as SASRec and Caser. Furthermore, we showed that the λRank loss function brought further effectiveness benefits to training under RSS not otherwise observed under a more traditional Sequence Continuation task. Indeed, on the large MovieLens-20M dataset, we observed that RSS applied to the SASRec model can result in a 60% improvement in NDCG@10 over the vanilla SASRec model (0.107 vs. 0.067 on MovieLens-20M) (Table 4.2), and a 16% improvement over a fully-trained BERT4Rec model (0.107 vs. 0.092), despite taking 93% less training time than BERT4Rec (1 hour vs 16 hours, see also Figure 4.1). Moreover, on the Yelp and Gowalla datasets, which both have geographic and strong sequential characteristics, RSS applied to SASRec brought significant benefits in both NDCG and Recall metrics (Table 4.2). We further experimented with two families of recency importance functions (power importance and exponential importance) and found that when their parameters are tuned properly, these functions are likely to produce similar sampling probability distributions and consequently achieve similar performance (this is shown in Figures 4.7 and 4.8). We also analysed the effect of RSS on the learned positional embeddings of the SASRec model and have shown that in contrast to the original version of SASRec, the RSS-enhanced version successfully learns to distinguish recent and earlier positions (see also Figure 4.9). While we did not apply RSS to BERT4Rec due to its usage of the Item Masking training objective, which is harder to adapt to RSS however, we believe that BERT4Rec could be adapted in future work to benefit from RSS.

Finally, in this chapter, we were not able to train BERT4Rec on the Gowalla dataset, as it computes full score distribution for every position in the sequence, which is not feasible for large-scale datasets. Indeed, BERT4Rec does not use negative sampling, which is typically needed to train models with large catalogues. In the next chapter, we will show that the absence of negative sampling is one of the key components in BERT4Rec's high effectiveness. We will also design a methodology that allows us to train both SASRec and BERT4Rec on large catalogues with negative sampling without hindering their effectiveness.

Chapter 5

Reducing Overconfidence in Sequential Recommendation Trained with Negative Sampling

As we discussed in the previous chapter (Section 4.2), each training step of Transformer-based recommendation models requires generating a batch of training samples with positive and negative items. In Chapter 4, we focused on how to select *positive* interactions for training while using *all* non-interacted items as negatives. However, in Section 2.3.5, we argued that computing all item scores during training may be too expensive due to the large item catalogue, as stated by Limitation L2.2. Hence, in practice, recommender systems are frequently trained using all positive interactions but only a small subsample of negative interactions; this technique is known as *negative sampling*.

As we show in this chapter, negative sampling increases the proportion of positive interactions in training data; hence, models tend to overestimate the probabilities of items being positive – we call this phenomenon *overconfidence*. We also show that overconfidence may lead to poor model effectiveness. To address the overconfidence problem, we propose our novel *Generalised Binary Cross-Entropy (gBCE)* loss function, which counters the negative effects of negative sampling, and gSASRec — a version of SASRec that uses gBCE. We show that by using gBCE, it is possible to train highly effective Sequential Recommender Systems while retaining the negative sampling required in a Large-Catalogue Scenario.

This chapter is organised as follows: Section 5.1 describes the need for negative sampling in a large-catalogue setup and discusses why the loss function needs to be adjusted in such a setup; Section 5.2 provides the necessary background for work; Section 5.3 formalises Sequential Recommendation using probabilistic framework and discusses the typically used loss functions; we describe the problem of overconfidence in Section 5.4; in Section 5.5 we introduce gBCE and theoretically analyse its properties, before defining gSASRec; Section 5.6 experimentally analyses the impact of negative sampling in SASRec, BERT4Rec and gSASRec; Section 5.7 provides concluding remarks for this chapter;

The material of this chapter is based on our full research paper [178], which was published in ACM RecSys 2023 proceedings, as well as the extended paper [184], which was published in the ACM Transactions on Recommender systems journal.

We now discuss the need for negative sampling in large-catalogue setups and why loss functions need to be adjusted when negative sampling is used.

5.1 Introduction

As discussed in Section 2.3.1, the success of language model architectures for Sequential Recommendation is explained by the similarities between modelling sequences of words in texts and modelling sequences of user-item interactions. However, a direct adaptation of language model architectures for Sequential Recommendation can be problematic because the number of items in the system catalogue can be much larger than the corresponding vocabulary size of the language models. For example, in 2024, YouTube had a catalogue of more than 800 million videos [84]. In practice, a direct adaptation of language models with catalogue sizes exceeding 1 million items is computationally prohibitive [176, 181]. Indeed, compared with traditional matrix factorisation models that compute one score distribution per user, Sequential Recommendation models are usually trained to predict scores for each position in the sequence, meaning that the model has to generate $S * N$ scores per sequence, where S is the sequence length, and N is the size of the catalogue. For example, to train a Sequential Recommendation model with 64 sequences of 200 items per batch, having 1M items would require 51 GB GPU memory, without accounting for the training gradients. Factoring in the gradients and model weights increases this to more than 100 GB, thereby exceeding consumer-grade GPU capacities.

A typical solution for this training problem is *negative sampling*: the models are trained on all *positive* interactions (the user-item interactions present in the training set) but only sample a very small fraction of *negative* interactions (all other possible but unseen user-item interactions). Negative sampling is known to be one of the central challenges [197] in training recommender systems: it increases the proportion of positive interactions in the training data distribution, and therefore, models learn to overestimate the probabilities of future user-item interactions. We describe this phenomenon as *overconfidence*. While the magnitude of retrieval scores is typically unimportant for the *ranking* of items, overconfidence is problematic because models frequently fail to focus on nuanced variations in the highly-scored items and focus on distinguishing top vs. bottom items instead. Another problem caused by overconfidence is more specific to models trained with the popular Binary Cross-Entropy loss: if an item i with high predicted probability p_i is sampled as a negative, $\log(1 - p_i)$ calculated by the loss function tends to $-\infty$, causing unstable training. Overall, we argue that overconfidence hinders model effectiveness and makes model training hard.

Although overconfidence is a general problem applicable to any recommender system trained with negative sampling (e.g. including Matrix Factorisation-based, see Section 2.1.2), in this thesis, we focus specifically on Sequential Recommender Systems, for which negative sampling is specifically important due to the large GPU memory requirement discussed above. Indeed, as we show in this chapter, the use of negative sampling leads to overconfidence in SASRec. Existing solutions that can address overconfidence induced by negative sampling in recommender systems (e.g. [198, 269]) are hard to adapt to deep learning-based Sequential Recommender models (see also Section 5.2.1). Hence, the overconfidence issue present in negatively sampled Sequential Recommendation models remains largely unsolved. Indeed, the state-of-the-art BERT4Rec model does not use negative sampling and, therefore, cannot be applied to datasets with large catalogues.[1]

Hence, to address the overconfidence issue in the Sequential Recommendation, we introduce a novel Generalised Binary Cross-Entropy loss (gBCE) – a generalisation of BCE loss using a generalised logistic sigmoid function [185, 204]. We further propose the Generalised SASRec model (gSASRec) – an enhanced version of SASRec [100] trained with more negative samples and gBCE. Theoretically, we prove that gSASRec can avoid overconfidence even when trained with negative sampling (see Theorem 5.5.1). Our theoretical analysis aligns with an empirical evaluation of gSASRec on three datasets (Steam, MovieLens-1M, and Gowalla), demonstrating the benefits of having more negatives and the gBCE loss during training. On smaller datasets (Steam and MovieLens-1M), the combination of these improvements significantly outperforms BERT4Rec's performance on MovieLens-1M (+9.47% NDCG@10); it also achieves comparable results on Steam (-1.46% NDCG@10, not significant), while requiring much less time to converge. Additionally, gBCE shows benefits when used with BERT4Rec trained with negative samples (+7.2% NDCG@10 compared with BCE Loss on MovieLens-1M with 4 negatives). On the Gowalla dataset, where BERT4Rec training is infeasible due to large catalogue size (see Section 4.4.3.2), we obtain substantial improvements over the regular SASRec model (+47% NDCG@10, statistically significant). Although this chapter focuses on Sequential Recommendation, our proposed methods and theory could be applicable to other research areas, such as recommender systems (beyond Sequential Recommendation), search systems, or natural language processing.

---

1. By BERT4Rec, we refer to the model architecture, the training task and the loss function. As we show in Section 5.6.2.1, while it is possible to train BERT4Rec's architecture while using negative sampling, doing so negatively impacts the model's effectiveness.

We also show that the overconfidence problem is related to the concept of *model calibration* [68] – the ability of the model to predict *actual* user-item interaction probabilities, which is usually quantified using the *Expected Calibration Error*. We show that SASRec has a very high expected calibration error of (ECE=0.966; the maximum possible ECE is 1.0). By applying gBCE, it is possible to reduce ECE below 0.01, allowing us to interpret the model's output as a probability. Low calibration error is useful, for example, to estimate revenue in the e-commerce scenario: when multiplying the predicted probability of purchase by the item price, we can estimate the expected revenue from the item.

In short, our contributions can be summarised as follows:

1. We define overconfidence through a probabilistic interpretation of Sequential Recommendation;
2. We show (theoretically and empirically) that SASRec is prone to overconfidence due to its negative sampling;
3. We propose gBCE loss and theoretically prove that it can mitigate the overconfidence problem;
4. We use gBCE to train gSASRec and show that it exhibits better (on MovieLens-1M) or similar (on Steam) effectiveness to BERT4Rec, while both requiring less training time, and also being suitable for training on large datasets;
5. We show that gBCE can also reduce the models' calibration error, and therefore, outputs of the models trained with gBCE can be used as probabilities (e.g. to compute expectations of revenue in the advertisement).

5.2 Negative sampling and the Large Vocabulary Bottleneck

We now describe the necessary background for this chapter. Section 5.2.1 describes existing heuristics for negative sampling. Section 5.2.2 describe how large vocabulary problems have been addressed in language models.

5.2.1 Negative Sampling Heuristics: Hard Negatives, Informative Samples, Popularity Sampling

One of the first attempts to train recommender systems with negative sampling was Bayesian Personalised Rank (BPR) [199]. The authors of BPR observed that models tend to predict scores close to exactly one for positive items in the training data (a form of overconfidence) and proposed to sample one negative item for each positive item and optimise the relative order of these items, instead of the absolute probability of each item to be positive. However, as Rendle (the first author of BPR) has recently shown [197], BPR optimises the Area Under Curve (AUC) metric, which is not top-heavy and is therefore not most effective for a ranking task. Hence, several improvements over BPR, such as WARP [254], LambdaRank [19], LambdaFM [269], and adaptive item sampling [198] have since been proposed to make negatively-sampled recommender models more suitable for top-heavy ranking tasks. These approaches usually try to mine the most informative (or *hard*) negative samples that erroneously have high scores and, therefore, are ranked high. Unfortunately, these approaches mostly rely on iterative or sorting-based sampling techniques that are not well-suited for neural network-based approaches used by Sequential Recommendation models: neural models are usually trained on GPUs, which allow efficient parallelised computing but perform poorly with such iterative methods. Indeed, Chen et al. [29] recently proposed an iterative sampling procedure for Sequential Recommendation but only experimented with smaller datasets (<30k items) where state-of-the-art results can be achieved without sampling at all (see also Section 5.6.2.1). Instead, Sequential Recommenders typically rely on simple heuristics such as uniform random sampling (used by Caser [236] and SASRec [100]) or do not use negative sampling at all (e.g. BERT4Rec [230]). Pellegrini et al. [171] recently proposed to sample negatives according to their popularity and showed this to be beneficial when the evaluation metrics are also popularity-sampled. Our initial experiments have shown that popularity-based sampling is indeed beneficial with popularity-based evaluation metrics but not with the full (unsampled) metrics. However, several recent publications [23, 40, 117, 176, 181] recommend against using sampled metrics, and therefore we avoid popularity sampling for evaluation.

Another heuristic that is popular for search tasks is *in-batch* sampling [135, Ch. 5] (e.g. used by GRU4Rec [81]). According to [197], in-batch sampling is equivalent to popularity-based negative sampling, and hence we avoid it for the same reason stated above. Indeed, we focus on uniform sampling – as used by many Sequential Recommender systems – and design a solution that helps to counter the overconfidence of such models caused by uniform sampling.

5.2.2 Large Vocabularies in Language Models

In Natural Language Processing, the problem aligned to a large catalogue size is known as the *large vocabulary bottleneck*. Indeed, according to Heap's Law [77], the number of different words in a text corpus grows with the size of the corpus, reaching hundreds of billions of words in recent corpora [49], and making computing scores over all possible words in a corpus problematic. A typical solution employed by modern deep learning language models is to use *tokenisation* (e.g. Word Piece [259] or BPE [58]), which splits infrequent words into (more frequent) sub-word groups of characters. This allows the use of a vocabulary of relatively small size (e.g. $\sim$30,000 tokens in BERT [46]) whilst being capable of modelling millions of words by the contextualisation of the embedded word piece representations. While decomposing item ids into sub-items can be used to reduce the item vocabulary of a recommender, the decomposition requires a more complex two-stage learning process to assign sub-items, which we will discuss in detail in Chapter 6. Other techniques have also been proposed to reduce the vocabulary size by pruning some tokens. For example, some classification models remove non-discriminating words [2, 228], which in the context of recommender systems means removing popular items (e.g. if a movie was watched by most of the users, it is not-discriminating). However, removing popular items is a bad idea as users are prone to interact with popular items and recommending popular items is a strong baseline [95].

Perhaps the most related work to ours is the Sampled Softmax loss [92], which proposes a mechanism to approximate the value of a Softmax function using a few negatives. However, Softmax loss is known to be prone to overconfidence [252]. Indeed, Sampled Softmax loss has recently been shown to incorrectly estimate the magnitudes of the scores in the case of recommender systems [257]. Our experiments with Sampled Softmax loss are aligned with these findings. We discuss Sampled Softmax loss in detail in Section 5.3.2 and experimentally evaluate it in Section 5.6.2.4. In summary, among the related work, there is no solution to the overconfidence problem in Sequential Recommender systems. Hence, we aim to close this gap and design a solution for this overconfidence that is suitable for sequential models. In the next section, we cover the necessary required preliminaries and then in Section 5.5, we show that the problem can be solved with the help of Generalised Binary Cross-Entropy loss.

5.3 Sequential Recommendation & loss Functions

In the following, Section 5.3.1, we more formally set the Sequential Recommendation task as a probabilistic problem and in Section 5.3.2 discuss loss functions used for training sequential models.

We now discuss a probabilistic view of Sequential Recommendation, which we use for improving SASRec in Section 5.5.

5.3.1 Probabilistic View of Sequential Recommendation

The goal of a Sequential Recommender system is to predict the next item in a sequence of user-item interactions. Formally, given a sequence of user-item interactions $u = \langle i_0, i_1, i_2, ...i_n \rangle$, where $i_k \in I$, the goal of the model is to predict the next user's interaction i_{n+1}. Sequential recommendation is usually cast as a *ranking problem*, so *predict* means to rank items in the catalogue according to their estimated probability of appearing next in the sequence. We denote this conditional probability distribution over all items appearing next in the sequence after u as $P(i|u)$. However, because we only consider probability distributions conditional on a given sequence of interaction, we omit this conditioning notation and refer to this distribution as $P(i)$ for simplicity. $P(i)$ is not directly observable: the training data only contains the user's actual interactions and does not contain information about the probabilities of any alternative items not interacted with.

Learning to estimate the distribution $P(i)$ is a hard task because the model doesn't have access to it, even during training. Instead, the model learns to estimate these probabilities, i.e. vector $\hat{p}$ = $\langle \hat{p}_1, \hat{p}_2, ..., \hat{p}_{|I|} \rangle$, by using observed interactions. After knowing the outcome of the interaction, the probability of a user interacting with the actually interacted item i^+ turns to 1, and the probabilities of all other interactions collapse to zero; therefore, the observed probability distribution can be written as $y(i) = \mathbb{I}[i = i^+]$, where $i^+ \in I$ is a positive interaction selected according to the training objective (as discussed in Section 4.2). $y(i)$ is measured *after* the user selected the item, so it always equals 1 for the positive item i^+ and equals 0 for all other items.

Note that to rank items, models do not have to compute the modelled probabilities $\hat{p}$ explicitly. Instead, models frequently compute item *scores* $s = \langle s_1, s_2, ..., s_{|I|} \rangle$ and assume that if item i is scored higher than item j ($s_i > s_j$) then item i is more likely to appear next in the sequence than item j ($\hat{p_i} > \hat{p_j}$). Whether or not it is possible to recover modelled item probabilities $\hat{p} = \langle \hat{p}_1, \hat{p}_2, .., \hat{p}_{|I|} \rangle$ from the scores s depends on the loss function used for model training.

We say that a loss function $\mathcal{L}$ *directly models probabilities* $\hat{p}$, if there exists a function f, which converts scores to probabilities ($\hat{p_i} = f(s_i)$) and when the model is trained with $\mathcal{L}$, $\hat{p}$ approximates the distribution P (e.g. a model trained with $\mathcal{L}$ minimises the KL divergence between P and $\hat{p}$). In the next section, we discuss the loss functions used by sequential models that directly model probabilities.

5.3.2 BCE Loss and Softmax Loss

Two popular loss functions, which directly model probabilities are *Binary Cross-Entropy (BCE)* (used by Caser and SASRec) and *Softmax loss* (used by BERT4Rec and ALBERT4Rec).

Binary Cross-Entropy is a *pointwise* loss, which treats the ranking problem as a set of independent binary classification problems. It models the probability with the help of the *logistic sigmoid function* $\sigma(s)$:

$$\hat{p_i} = \sigma(s_i) = \frac{1}{1 + e^{-s_i}} \tag{5.1}$$

The value of BCE loss is then computed as:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{|I|} \sum_{i \in I} (y(i) \log(\hat{p_i}) + (1 - y(i)) \log(1 - \hat{p_i})) \tag{5.2}$$

BCE minimises the KL divergence [161, Ch. 5] between the observed interactions and the modelled distributions, $D_{KL}(y(i)||p_i)$, where each of the probability distributions is treated as a distribution with two outcomes (i.e. interaction/no interaction). BCE considers each item probability independently, so the sum of the probabilities over the entire catalogue does not have to add up to 1. Indeed, as we show in Section 5.5 when BCE is used with negative sampling, the model learns to predict probabilities close to 1 for the most highly-ranked items.

In contrast, Softmax loss treats the ranking problem as a multi-class classification problem, thereby considering the probability distribution across all items, obtained by using a softmax$(\cdot)$ operation:

$$\hat{p}_i = \text{softmax}(s_i) = \frac{e^{s_i}}{\sum_{j \in I} e^{s_j}} \tag{5.3}$$

The value of Softmax loss is then computed as:

$$\mathcal{L}_{softmax} = -\sum_{i \in I} y(i) \log(\hat{p}_i) = -\log(\text{softmax}(s_{i^+})) \tag{5.4}$$

Softmax loss minimises KL divergence [161, Ch. 5] between actual and modelled distributions $D_{KL}(y||p)$, where each y and p are multi-class probability distributions. In contrast to BCE, the item probabilities $\hat{p}_i$ modelled by Softmax loss add up to 1, meaning that overconfidence is less prevalent (however, it is still known to overestimate probabilities of the top-ranked items [252]).

Unfortunately, the softmax$(\cdot)$ operation used by Softmax loss requires access to *all* item scores to compute the probabilities (which makes it more of a *listwise* loss). In contrast, if the model is trained with negative sampling, the scores are only computed for the *sampled* items. This makes Softmax loss unsuitable for training with negative sampling. In particular, this means that BERT4Rec, which uses the Softmax loss, cannot be trained with sampled negatives (without changing the loss function).

To use Softmax loss with sampled negatives, Jean et al. [92] proposed *Sampled Softmax Loss (SSM)*.[2] SSM approximates probability $\hat{p}_i$ from Equation (5.3) using a subset of k negatives $I_k^- \subset I^-$. This approximation is then used to derive the loss:

$$\hat{p}_i = \text{SSM}(s_i, I_k^-) = \frac{e^{s_i}}{e^{s_i} + \sum_{j \in I_k^-} e^{s_j}} \tag{5.5}$$

$$\mathcal{L}_{SSM} = -\sum_{i \in \{I_k^- \cup i^+\}} y(i) \log(\hat{p}_i) = -\log(\text{SSM}(s_{i^+}, I_k^-)) \tag{5.6}$$

2. Here we consider a simplified version of the Sampled Softmax Loss as is typically used in recommender systems literature [106, 257]; sometimes the same loss is also referred as the "Noise Contrastive Estimation loss" in recommender systems litetrature [189]. A full version of the Sampled Softmax Loss proposed by Jean et al. [92] additionally includes the correction term $-\log(Q)$, where Q is the *proposal distribution* from which negatives are sampled. The correction term is designed to solve a similar problem of probability distribution shift caused by negative sampling. It may be possible that careful design of the proposal distribution Q and applying the correction term to sampled Softmax will have a similar effect on the effectiveness of Sequential Recommender systems as the methodology proposed in this chapter; we leave this analysis for future research.

The estimated probability value computed with Sampled Softmax is higher than the probability estimated using full Softmax, as the denominator in Equation (5.3) is larger than the denominator in Equation (5.5). However, if all high-scored items are included in the sample I_k^-, the approximation becomes close. To achieve this, Jean et al. originally proposed a heuristic approach specific to textual data (they segmented texts into chunks of related text, where each chunk had only a limited vocabulary size). In the context of Sequential Recommender systems, some prior works [171, 270] used variations of SSM loss with more straightforward sampling strategies, such as popularity-based or uniform sampling. In this chapter, we focus on the simplest scenario of uniform sampling, and therefore, in our experiments, we use Sampled Softmax loss with uniform sampling. Note that Sampled Softmax Loss normalises probabilities differently compared to the full Softmax loss, and therefore, the Sampled Softmax loss and the full Softmax loss are different loss functions. Indeed, as Sampled Softmax uses only a sample of items in the denominator of Equation (5.5), the estimated probability of the positive item $\hat{p}_i$ is an overestimation of the actual probability, a form of overconfidence. Indeed, as mentioned above, Sampled Softmax loss fails to estimate probabilities accurately for recommender systems [257]. Nevertheless, as variations of Sampled Softmax have been used in Sequential Recommendations [171, 270], we use Sampled Softmax loss as a baseline in our experiments (see Section 5.6.2.4).

In contrast, it is possible to calculate BCE loss over a set of sampled negatives I_k^- without modifying the loss itself (except for a normalisation constant, which does not depend on the item score and therefore can be omitted), as follows:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{|I_k^-| + 1} \left(\log(\sigma(s_{i^+})) + \sum_{i \in I_k^-} \log(1 - \sigma(s_i)) \right) \tag{5.7}$$

Using BCE with sampled negatives is a popular approach, applied by models such as SASRec [100] (which uses 1 negative per positive), and Caser [236] (which uses 3 negatives).

Unfortunately, negative sampling changes the balance of the positive/negative samples in training data. As we discuss in the next section, this balance change causes model effectiveness to degrade. To counter this effect, the loss function can be adjusted by changing the weights of positive/negative instances. Related work on adjusting class weights in the loss function has been done in adjacent domains, most notably in computer vision [38, 137, 164]. However, in the domain of Sequential Recommender Systems, the adjustment of class weights in loss functions hasn't been systematically studied. Indeed, some of the most cited [100, 236] as well as the most recent [261, 266, 277] publications use negative sampling coupled with the BCE loss without

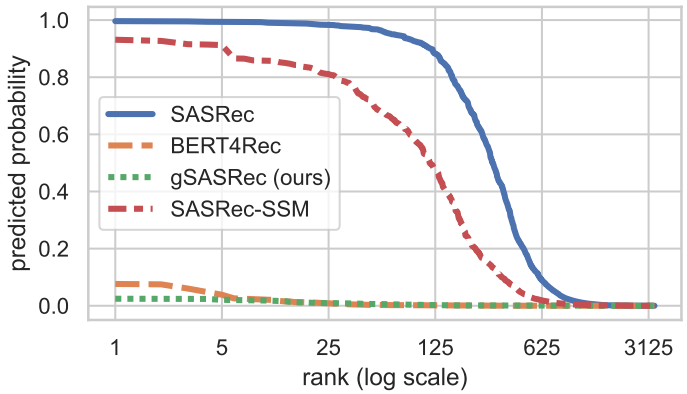


Figure 5.1: Predicted probability at different ranks for user 963 in MovieLens-1M. SASRec-SSM is a SASRec model trained with Sampled Softmax loss with 16 negatives.

any class weight adjustments. In the next section, we show that in Sequential Recommender systems, negative sampling coupled with the BCE loss also causes a problem we call *overconfidence* that leads to effectiveness degradation. In Section 5.5, we show how this problem can be mitigated with class-weighting-based gBCE loss.

5.4 Model Overconfidence

5.4.1 Overconfidence Phenomenon

We say that a model is *overconfident* in its predictions if its predicted probabilities $\hat{p}_i$ for highly-scored items are much larger compared to actual probabilities $P(i)$, i.e., $\hat{p}_i \gg P(i)$. In general, the magnitude of the relevance estimates are rank-invariant, i.e. do not affect the ordering of items, and hence, they are rarely considered important when formulating a ranking model. In contrast, overconfidence is problematic only for the loss functions used to train the models, particularly when they directly model the interaction probability. Indeed, for some loss functions (such as pairwise BPR [199] or listwise LambdaRank [19]), only the difference between the scores of paired items ($s_i - s_j$) is important. Therefore, we cannot define overconfidence for these losses. However, these losses usually require algorithms that iteratively select "informative" negative samples, which are hard to apply with deep learning methods (see also Section 5.2.1). As discussed in Section 5.3.1, the probability distribution $P(i)$ cannot be directly observed, and therefore overconfidence may be hard to detect. However, in some cases, overconfidence may be obvious. For example, Figure 5.1 shows predicted probabilities by four different models for a

sample user in the MovieLens-1M dataset. As can be seen from the figure, SASRec's predicted probabilities for items at rank positions between 1 and 25 are almost indistinguishable from 1. This is a clear sign of overconfidence: only one of these items can be the correct prediction, and therefore, we expect the *sum* of probabilities to be approximately equal to 1, and not each individual probability. In fact, in this figure, the sum of all probabilities predicted by SASRec equals 338.03. In contrast, for BERT4Rec, the sum of probabilities equals exactly 1 (as the probabilities are computed using Softmax) and for our gSASRec (see Section 5.5.5) it is equal to 1.06. From the figure, we also see that a SASRec model trained with Sampled Softmax loss is also prone to overconfidence (sum of all probabilities equals 152.3).

We argue that overconfidence for highly-ranked items is problematic: the model does not learn to distinguish these items from each other (all their predicted probabilities are approximately equal) and focuses on distinguishing top items from the bottom ones. The lack of focus on the top items contradicts our goal: we want the correct order of highly-ranked items and are not interested in score differences beyond a certain cutoff. Moreover, overconfidence is specifically problematic for the BCE loss: if an item with high probability $\hat{p}_i \approx 1$ is selected as a negative (the chances of such an event are high when there are many high-scored items), $\log(1 - p_i)$ computed by the loss function tends to $-\infty$. In practice, even for small numbers, $\log(\cdot)$ returns values that have reasonable magnitude. For example, in Figure 5.1 for the highest ranked false-positive, the interaction probability predicted by SASRec is $p_i = 0.99601$. If that item by chance is sampled as a negative, the value of the logarithm computed by Binary Cross-Entropy loss equals to $\log(1 - 0.99601) = -5.52$. While on the surface, this number is far away from $-\infty$, in practice, it still dominates the loss value: the logarithm for the positive interaction, in this case, is -0.0037. This makes the gradient of the loss function large, causing training instability; we also show experimentally that BCE's gradient remains large throughout the model training in Section 5.6.2.6.

5.4.2 Model Calibration

Overconfidence is closely related to model *calibration* [68, 168] – the ability of a model to accurately predict interaction probabilities $P(i)$. Calibration is not important for accuracy-based ranking metrics, such as NDCG@10, as these metrics only depend on the order of predicted scores. However, calibration is important in some scenarios when we actually need to be able to interpret the model's output as a probability. One such example is when we want to order items by expected revenue in e-commerce recommendations – to compute expected revenue, we need to multiply the item's price by the probability of the user buying the item.

To measure models' calibration, Naeini et al. [168] propose to use *Expected Calibration Error* (ECE) metric. Suppose we have n model's predictions, for which we know ground truth positive/negative class. To measure ECE, we group the predictions with roughly equal predicted probabilities into M *bins*, $\langle B_1, B_2 \dots, B_M \rangle$, and then compare the magnitude of predicted probabilities within these bins with the fraction of positive interactions in the same bins:

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} \left| \frac{\text{Positives}(B_m)}{|B_m|} - P(B_m) \right| \tag{5.8}$$

where $\frac{|B_m|}{n}$ is the proportion of all samples that fall into bin B_m, $\text{Positives}(B_m)$ is the number of positives within bin B_m and $P(B_m)$ is probability predicted by the model to samples within B_m (recall that we group samples in such a way that all samples in B_m have approximately same predicted probability so that we can use for example mean predicted probability for samples in B_m as $P(B_m)$).

Our intuition is that for a calibrated model, we expect that among samples with predicted probability ρ, the proportion of positives within the bin should also be approximately equal to ρ, and the discrepancy between the two can be used as a measure of model calibration. As we describe in Section 5.4.1, in overconfident recommendation models, the predicted interaction probability is much higher compared to the actual interaction probability, and, therefore, these models have high expected calibration error.

We now introduce gBCE loss, a generalisation over Binary Cross-Entropy. We then apply it for a theoretical analysis of BCE's overconfidence and show how gBCE mitigates the overconfidence problem and improves model calibration.

5.5 Generalised Binary Cross Entropy and its Properties

In this section, we design gBCE and theoretically show that it can mitigate the overconfidence problem. In Section 5.5.1, we introduce gBCE and analyse its properties; in Section 5.5.2, we show that gBCE may be replaced with regular BCE loss with transformed positive scores, which may be more convenient in practice; in Section 5.5.3 we show how to re-parametrise gBCE to make it independent of the chosen sampling rate; finally, in Section 5.5.5 we introduce gSASRec – an improved version of SASRec, which uses gBCE.

5.5.1 Generalised Binary Cross Entropy

We now introduce *Generalised Binary Cross Entropy (gBCE)* loss, which we use to analyse and mitigate overconfidence induced by negative sampling. We define gBCE, parameterised by β as:

$$\mathcal{L}^{\beta}_{\text{gBCE}} = -\frac{1}{|I^-_k| + 1}\left(\log(\sigma^{\beta}(s_{i^+})) + \sum_{i \in I^-_k} \log(1 - \sigma(s_i))\right). \tag{5.9}$$

gBCE differs from regular BCE loss in that it uses the *generalised logistic sigmoid function* [185, 204] for the positive interaction (sigmoid raised to the power of β). The power parameter $\beta \geq 0$ controls the shape of the generalised sigmoid[3]. For example, when $\beta \approx 0$, the output of the generalised sigmoid becomes closer to 1 for all input scores. On the other hand, when $\beta = 1$, BCE and gBCE are equal:

$$\mathcal{L}^{1}_{gBCE} = \mathcal{L}_{\text{BCE}} \tag{5.10}$$

Similarly to BCE, gBCE is also a pointwise loss, and it considers the probability of interaction as a sigmoid transformation of the model score (Equation (5.1)). We now show the exact form of the relation between the actual interaction probability $P(i)$ (which we desire to estimate, as we discuss in Section 5.3.1) and the modelled probabilities $\hat{p}_i = \sigma(s_i)$, learned by a model trained with gBCE.

Theorem 5.5.1. *For every user in the dataset, let $P(i)$ be the interaction probability distribution of the user interacting with item $i \in I$, $s = \langle s_1, ...s_{|I|}\rangle$ are scores predicted by the model, i^+ is a positive item selected by the user, $I^-_k = \langle i^-_1, i^-_2, ..., i^-_k\rangle$ are k randomly (uniformly, with replacement) sampled negatives, $\alpha = \frac{k}{|I^-|}$ is the negative sampling rate. Then a recommender model, trained on a sufficiently large number of training samples using gradient descent and $\mathcal{L}^{\beta}_{\text{gBCE}}$ loss, will converge to predict score distribution s, so that*

$$\sigma(s_i) = \frac{\beta P(i)}{\alpha - \alpha P(i) + \beta P(i)}; \forall i \in I \tag{5.11}$$

---

3. Note that β can be extracted from the logarithm in Equation (5.9); an equivalent form for the equation is $\mathcal{L}^{\beta}_{\text{gBCE}} = -\frac{1}{|I^-_k|+1}\left(\beta \log(\sigma(s_{i^+})) + \sum_{i \in I^-_k} \log(1 - \sigma(s_i))\right)$. As β is a number between 0 and 1, it can be interpreted as the down-weighting coefficient for the positive component of the loss. This interpretation makes gBCE similar to other extensions of BCE that change weights of positive and negative classes; see, for example [137]. However, to the best of our knowledge, these extensions have not been used to solve the overconfidence problem in recommender systems

Proof. With a sufficiently large number of training samples, gradient descent converges to minimise the expectation of the loss function [64, Ch. 4] (assuming the expectation has no local minima). Therefore, the predicted score distribution converges to the minimum of the expectation $\mathbb{E}\left[\mathcal{L}^{\beta}_{\text{gBCE}}\right]$:

$$s = \arg\min_{s} \mathbb{E}\left[\mathcal{L}^{\beta}_{\text{gBCE}}\right] \tag{5.12}$$

Hence, our goal is to show that Theorem 5.5.1 is true if and only if the expectation $\mathbb{E}\left[\mathcal{L}^{\beta}_{\text{gBCE}}\right]$ is minimised.

To show that, we first rewrite the definition of $\mathcal{L}^{\beta}_{\text{gBCE}}$ (Equation (5.9)) as a sum of contributions for each individual item in I:

$$\mathcal{L}^{\beta}_{\text{gBCE}} = \frac{1}{|I^{-}_{k}| + 1} \sum_{i \in I} \mathcal{L}_i \tag{5.13}$$

where the contribution of each item, $\mathcal{L}_i$, is defined as follows:

$$\mathcal{L}_i = -(\mathbb{I}[i = i^{+}] \log(\sigma^{\beta}(s_i)) + \sum_{j=1}^{k} \mathbb{I}[i = i^{-}_{j}] \log(1 - \sigma(s_i))) \tag{5.14}$$

The probability of an item being selected as a positive is defined by the interaction probability distribution:

$$P(\mathbb{I}[i = i^{+}]) = P(i) \tag{5.15}$$

whereas the probability of an item being selected as j^{th} negative is equal to the product of the probability of an item being negative and the negative sampling probability. Suppose we apply a uniform sampling with a replacement for identifying negatives. In that case, the sampling probability is always equal to $\frac{1}{|I^{-}|}$, so overall, the probability of selecting an item i as the j^{th} negative can be written as:

$$P(\mathbb{I}[i = i^{-}_{j}]) = \frac{1}{|I^{-}|}(1 - P(i)) \tag{5.16}$$

We can now calculate the expectations of each individual loss contribution $\mathbb{E}[\mathcal{L}_i]$:

$$\mathbb{E}[\mathcal{L}_i] = -(P(\mathbb{I}[i = i^+]) \log(\sigma^\beta(s_i)) + \sum_{j=1}^{k} P(\mathbb{I}[i = i_j^-]) \log(1 - \sigma(s_i)))$$

(By the definition of expectation)

$$= -(P(i) \log(\sigma^\beta(s_i)) + \sum_{j=1}^{k} \frac{1}{|I^-|}(1 - P(i)) \log(1 - \sigma(s_i)))$$

(Substituting Equations (5.15) and (5.16))

$$= -(P(i) \log(\sigma^\beta(s_i)) + \frac{k}{|I^-|}(1 - P(i)) \log(1 - \sigma(s_i)))$$

(The sum is just the same term repeated k times)

$$= -(P(i) \log(\sigma^\beta(s_i)) + \alpha(1 - P(i)) \log(1 - \sigma(s_i)))$$

(Substituting the sampling rate definition $\alpha = \frac{k}{|I^-|}$) (5.17)

Differentiating Equation (5.17) on $\sigma(s_i)$ we get:

$$\frac{d\,\mathbb{E}[\mathcal{L}_i]}{d\sigma(s_i)} = -\frac{\beta P(i)}{\sigma(s_i)} + \frac{\alpha(1 - P(i))}{1 - \sigma(s_i)} \tag{5.18}$$

Our goal is to minimise the expectation $\mathbb{E}[\mathcal{L}_i]$, so equating this derivative to zero and solving for $\sigma(s_i)$ we obtain the value of $\sigma(s_i)$, which minimises the expectation:

$$\sigma(s_i) = \frac{\beta P(i)}{\alpha - \alpha P(i) + \beta P(i)} \tag{5.19}$$

We now rewrite the expectation $\mathbb{E}\left[\mathcal{L}^\beta_{\text{gBCE}}\right]$ as the sum of its individual components:

$$\mathbb{E}\left[\mathcal{L}^\beta_{\text{gBCE}}\right] = \mathbb{E}\left[\frac{1}{|I_k^-| + 1} \sum_{i \in I} \mathcal{L}_i\right] = \frac{1}{|I_k^-| + 1} \sum_{i \in I} \mathbb{E}\left[\mathcal{L}_i\right] \tag{5.20}$$

According to Equation (5.20), the expectation $\mathbb{E}\left[\mathcal{L}^\beta_{\text{gBCE}}\right]$ is minimised when, for each $i \in I$, the individual contributions $\mathbb{E}[\mathcal{L}_i]$ are minimised, i.e. when Equation (5.19) is true for each $i \in I$.

□

We now use Theorem 5.5.1 to analyse properties of both regular and generalised Binary Cross-Entropy losses. First, we show that it is possible to train a model to estimate an interaction distribution $P(i)$ exactly using gBCE loss (i.e. train a *calibrated* model, as per Section 5.4.2).

Corollary 5.5.1.1. *If a model is trained using negative sampling with sampling rate $\alpha \leq 1$ and gBCE loss $\mathcal{L}^{\beta}_{\text{gBCE}}$ with $\beta = \alpha$, then the model converges to predict probabilities calibrated with the interaction probability distribution:*

$$\sigma(s_i) = P(i) \tag{5.21}$$

Proof. We can obtain Equation (5.21) by substituting $\beta = \alpha$ in Equation (5.11). □

We now use Theorem 5.5.1 to analyse properties of regular Binary Cross-Entropy loss.

Corollary 5.5.1.2. *If a model is trained with regular BCE loss $\mathcal{L}_{\text{BCE}}$ and negative sampling, with sampling rate α, then it converges to predict scores s_i so that*

$$\sigma(s_i) = \frac{P(i)}{\alpha - \alpha P(i) + P(i)} \tag{5.22}$$

Proof. According to Equation (5.10), $\mathcal{L}_{\text{BCE}}$ is equal to $\mathcal{L}^{\beta}_{\text{gBCE}}$ with $\beta = 1$. Substituting $\beta = 1$ into Equation (5.11) we obtain Equation (5.22). □

We can now show that SASRec learns an overconfident score distribution:

Corollary 5.5.1.3. *The SASRec model with $\mathcal{L}_{\text{BCE}}$ and one negative per positive converges to yield scores s_i, such that:*

$$\sigma(s_i) = \frac{P(i)|I| - P(i)}{P(i)|I| - 2P(i) + 1} \tag{5.23}$$

Intuitively, for top-scored items, both numerator and denominator of this equation are likely to be dominated by the $P(i)|I|$ term, making overall predicted probability equal to 1 (i.e. causing overconfidence).

Proof. SASRec uses one negative per positive, meaning that its sampling rate is equal to:

$$\alpha = \frac{1}{|I| - 1} \tag{5.24}$$

Substituting Equation (5.24) into Equation (5.22), we get Equation (5.23). □

Corollary 5.5.1.3 explains why SASRec tends to predict very high probabilities for top-ranked items: when an item has a higher-than-average probability of being selected ($P(i) \gg \frac{1}{|I|}$), the term $P(i)|I|$ dominates both the numerator and denominator of Equation (5.23), meaning that the predicted probability $\sigma(s_i)$ will be very close to 1.

5.5.2 Relation between BCE and gBCE

In Section 5.5.1 we showed that gBCE is equal to regular BCE loss when the power parameter β is set to 1. We now show that these two loss functions have a deeper relation, which allows using well-optimised versions of BCE from deep learning frameworks instead of gBCE.

Theorem 5.5.2. *Let s^+ be the predicted score for a positive item and $s^- = \langle s_{i_1^-}, s_{i_2^-}..s_{i_{|I^-|}^-} \rangle$ be the predicted scores for the negative items. Then*

$$\mathcal{L}^{\beta}_{\text{gBCE}}(s^+, s^-) = \mathcal{L}_{\text{BCE}}(\gamma(s^+), s^-) \tag{5.25}$$

where

$$\gamma(s^+) = \log\left(\frac{1}{\sigma^{-\beta}(s^+) - 1}\right) \tag{5.26}$$

Proof. According to the definition of the logistic sigmoid function (Equation (5.1)),

$$
\begin{aligned}
\sigma(\gamma(s^+)) &= \frac{1}{e^{-\gamma(s^+)}+1} \\
&= \frac{1}{e^{-\log\left(\frac{1}{\sigma^{-\beta}(s^+)-1}\right)}+1} && \text{(Substituting } -\gamma(s^+) \text{ with its definition (Eq. (5.26)))} \\
&= \frac{1}{e^{\log\left(\sigma^{-\beta}(s^+)-1\right)}+1} && \text{(Using properties of the } \log(\cdot) \text{ function)} \\
&= \frac{1}{\sigma^{-\beta}(s^+)-1+1} && \text{(The exponent and the logarithm cancel each other out)} \\
&= \frac{1}{\sigma^{-\beta}(s^+)} = \sigma^{\beta}(s^+) && (5.27)
\end{aligned}
$$

Substituting $\sigma^{\beta}(s^+) = \sigma(\gamma(s^+))$ into the definition of $\mathcal{L}^{\beta}_{\text{gBCE}}$ (Equation (5.9)) and taking into account the definition of $\mathcal{L}_{\text{BCE}}$ (Equation (5.7)) we get the desired equality:

$$
\begin{aligned}
\mathcal{L}^{\beta}_{\text{gBCE}}(s^+, s^-) &= -\frac{1}{|I_k^-|+1}\left(\log(\sigma^{\beta}(s_{i^+})) + \sum_{i\in I^-}\log(1-\sigma(s_i))\right) \\
&= -\frac{1}{|I_k^-|+1}\left(\log(\sigma(\gamma(s^+))) + \sum_{i\in I^-}\log(1-\sigma(s_i))\right) \\
&= \mathcal{L}_{\text{BCE}}(\gamma(s^+), s^-)
\end{aligned}
$$

□

In practice, Theorem 5.5.2 allows us to transform the predicted positive scores by using Equation (5.26) and then train the model using the regular BCE loss instead of using gBCE directly. This is actually preferable because many machine learning frameworks have efficient and numerically stable implementations for standard loss functions such as BCE loss. Indeed, in our implementation[4], we also rely on the score transformation in Equation (5.26) and regular BCE loss instead of using gBCE directly.

4. All code for this chapter is available at https://github.com/asash/gsasrec

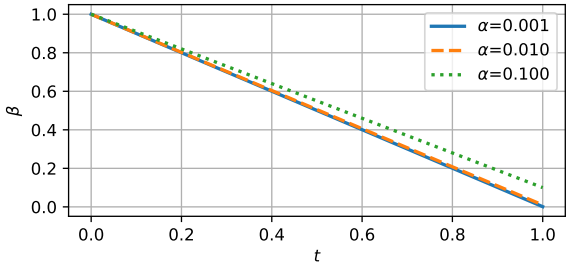


Figure 5.2: Relation between calibration parameter t and power parameter β when using different sampling rates α.

5.5.3 Calibration Parameter t

As shown in Section 5.5.1, setting the power parameter $\beta = 1$ in gBCE resembles the regular BCE loss, whereas setting β equal to the sampling rate α results in learning a fully calibrated distribution. This means that reasonable values of the β parameter lie in the interval $[\alpha..1]$. In practice, we found working with this interval inconvenient; as a result, we usually do not control the α parameter directly and instead infer it from the number of negatives and size of the dataset. Similarly, the possible values of β depend on these variables as well. To make the interval of possible values independent of α, we control the power parameter β indirectly with the help of a *calibration parameter* t, which adjusts β as follows:

$$\beta = t \cdot (\alpha - 1) + 1 \tag{5.28}$$

This substitution makes model configuration simpler: we select t in the interval $[0..1]$, where $t = 0$ ($\beta = 1$) corresponds to regular BCE loss, and $t = 1$ ($\beta = \alpha$) corresponds to the fully calibrated version of gBCE, which drives the model to estimate the interaction probabilities $P(i)$ exactly (according to Corollary 5.5.1.1).

Figure 5.2 illustrates the relation between the calibration parameter t and the power parameter β given three different sampling rates α according to Equation (5.28). As the figure shows, the relation is linear. We can also see from the figure that in most of the cases, parameter β can be approximated as $1 - t$:

$$\beta \approx 1 - t$$

However, this approximation does not work when t tends to 1 because, in this case, β approaches the value of sampling rate α instead of 0. In summary, Equation (5.28) defines a linear reparametrisation of the gBCE's power parameter β with the help of calibration parameter t. In contrast with the range of possible values of the power parameter β, the range of possible values of t does not depend on the dataset size, and therefore, it is more convenient to use in practice.

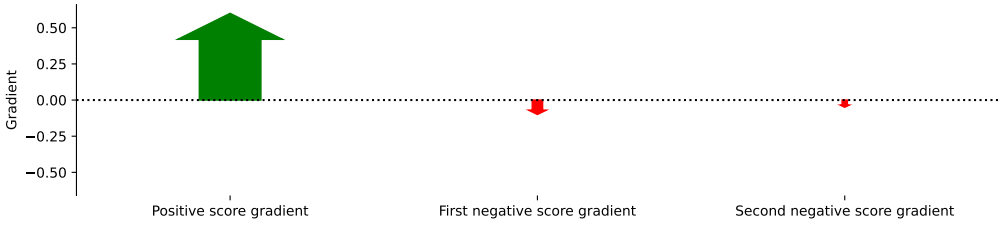
5.5.4 Illustrative Example

To better understand *why* gBCE helps to avoid overconfidence, we consider the following illustrative example. Let's say that we are training a recommendation model using negative sampling with two randomly sampled negatives out of an overall of 3415 negatives (this corresponds to the catalogue size of the MovieLens-1M dataset that we use in our experiments). In this example, the sampling rate is $\alpha = \frac{2}{3145} \approx 0.000585$. Figure 5.3a lists out scores predicted by the model in our example and corresponding probabilities obtained by the sigmoid transformation of the scores. As the table shows, in our example, the score predicted by the model for the positive item is $s^+ = -0.4054$, and the two negative scores are $s_1^- = -2.1972$ and $s_2^- = -2.9444$. These scores correspond to predicted interaction probabilities of 0.4 for the positive item, 0.10 for the first negative and 0.05 for the second negative. As we can see in our example, the positive item is already ranked above the negative samples. So, from the ranking perspective, there is no need for further model parameter updates for this particular example.

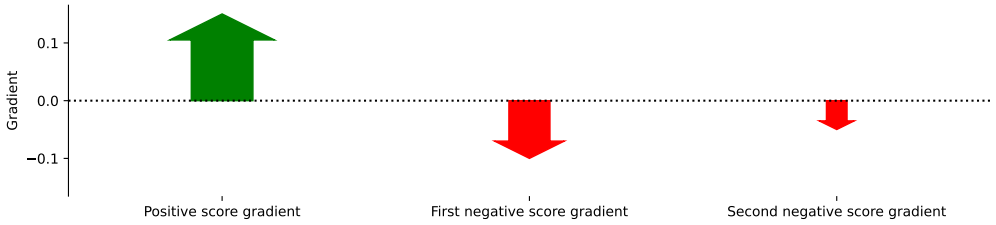
However, BCE and gBCE loss functions are pointwise losses: these losses ignore the ranks and drive the model to make the predicted probability for the positive $\sigma(s^+)$ as close as possible to 1 and make the predicted probabilities for negatives $\sigma(s_1^-)$ and $\sigma(s_2^-)$ as close as possible to 0, i.e. drive predicted the positive score s^+ to $+\infty$, and drive the negative scores s_1^- and s_2^- to $-\infty$. Frequently, these goals (increasing positive scores and decreasing negative scores) are conflicting, and the learning algorithm has to find a balance between them. In gradient descent-based learning methods, the balance between the goals is defined by the gradients of the predicted scores, e.g. Burges [19] proposed to interpret score gradients as "forces" (or "little arrows"), attached to the scores; and the balance of these "forces" defines the overall direction of the model parameter updates.

Item	Predicted score s	Predicted Probability (i.e. $\sigma(s)$)
Positive (s^+)	-0.4054	0.40
First negative (s_1^-)	-2.1972	0.10
Second negative (s_2^-)	-2.9444	0.05

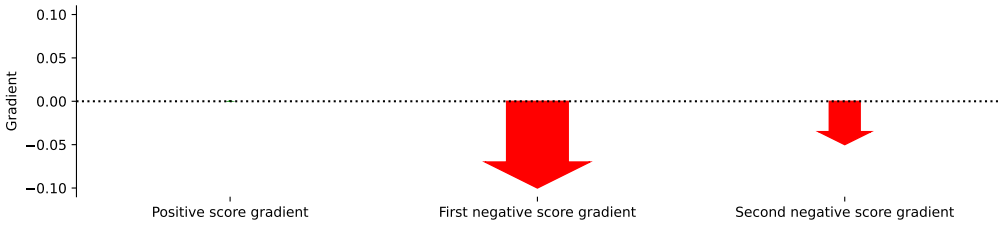
(a) Scores predicted by a model



(b) No calibration: $t = 0.0$, i.e. standard BCE



(c) gBCE with recommended calibration: $t = 0.75$



(d) gBCE with full calibration: $t = 1.00$

Figure 5.3: Illustrative example of the effect of calibration parameter t on the gradients of predicted scores. In this example, we assume that the model is trained using *two* randomly sampled negatives out of 3415, which corresponds to sampling rate $\alpha = \frac{2}{3415} \approx 0.0006$.

Figure 5.3 illustrates the balance between positive and negative gradients for three configurations of gBCE, with calibration parameter t selected from $\langle 0, 0.75, 1.0 \rangle$. Figure 5.3b shows the balance of the gradient for $t = 0.0$ (i.e. standard BCE loss). As we can see from the figure, in this configuration, the positive score gradient dominates, and therefore, the loss prefers increasing the positive score over decreasing the negatives. Therefore, in this configuration, over time, the positive predicted probability $\sigma(s^+)$ will become very close to 1.0, even though it means creating many false positives.

Figure 5.3c demonstrates the gradients balance in our *recommended* (see Section 5.6.2.3) setting $t = 0.75$. As we can see, in this configuration, there is a balance between the gradients of positive and negative scores. Therefore, when setting $t = 0.75$, the model is less prone to overconfidence; however, the balance helps the model to converge quickly.

Finally, Figure 5.3d shows the gradients balance for the setting $t = 1.0$, which corresponds to full calibration (recall Corollary 5.5.1.1). As we can see, in this setting, the negative gradients dominate (the scale of the positive gradient is so small that it is not even visible). In this scenario, overconfidence is not an issue, and the model learns to predict probabilities calibrated with the actual interaction probabilities (we also show this empirically in Section 5.6.2.2). However, the loss's focus on negative samples slows down the learning process, and therefore, training such a model requires more time compared to our recommended scenario.

In summary, the calibration parameter t in gBCE changes the balance between positive and negative score gradients. With small values of $t \approx 0$, the positive score gradients dominate, which leads to overconfidence and, as a consequence, suboptimal model effectiveness. With large values of $t \approx 1$, the gradients of negative scores dominate, slowing down the model training. The recommended value of $t = 0.75$ balances the positive and negative gradients. In this case, the model mostly avoids the problems caused by overconfidence while still converging relatively quickly.

This concludes the analysis of gBCE and its properties. We now discuss how we use gBCE with Transformer-based Sequential Recommendation models.

5.5.5 gSASRec and gBERT4Rec

gSASRec (generalised SASRec) is a version of the SASRec model with an increased number of negatives, trained with gBCE loss. Compared with SASRec, gSASRec has two extra hyperparameters: (i) number of negative samples per positive $k \in [1..|I^-|]$, and (ii) parameter $t \in [0..1]$, which indirectly controls the power parameter β in gBCE using to Equation (5.28). In particular, when $k = 1$ and $t = 0$, gSASRec is the original SASRec model, as SASRec uses 1 negative per positive and gBCE becomes BCE when $t = 0$. While our primary focus is on the SASRec model, it is possible to apply gBCE with other models; as an example, we use it also with BERT4Rec backbone (we call this model gBERT4Rec, see Section 5.6.2.4).

In the next section, we empirically evaluate gSASRec and show that its generalisations over SASRec are indeed beneficial and allow it to match BERT4Rec's performance while retaining negative sampling.

5.6 Experiments

We design our experiments to answer the following research questions:

gBCE and gSASRec Research Questions

RQ5.1: How does negative sampling affect BERT4Rec's performance gains over SASRec?

RQ5.2: What is the effect of gBCE on predicted item probabilities?

RQ5.3: What is the effect of negative sampling rate and parameter t on the performance of gSASRec?

RQ5.4: How does gBCE loss affect the performance of SASRec and BERT4Rec models trained with negative sampling?

RQ5.5: How does gSASRec perform in comparison to state-of-the-art Sequential Recommendation models?

RQ5.6: What is the effect of the calibration parameter t on model gradients and model convergence speed during gSASRec training?

RQ5.7: What is the impact of gBCE on Expected Calibration Error?

5.6.1 Experimental Setup

5.6.1.1 Datasets and Metrics

We conduct our main experiments with three datasets: MovieLens-1M [71], Steam [170] and Gowalla [32][5]. MoveLens-1M and Steam have a relatively small number of items. Hence, these datasets are suitable for evaluating models even without applying negative sampling. As a demonstration that gSASRec is suitable for larger datasets, we also use the Gowalla dataset, which contains over 1 million items in the catalogue and, as we have shown in Section 4.4.3.2, is problematic for BERT4Rec. We refer back to Section 2.4.1 for the details about datasets and data splitting strategies.

5. In Appendix A in the journal version of this work [184], we additionally experiment with the RetailRocket dataset, which has recently been recommended as a good fit for the Sequential Recommendation task [79].

We evaluate our models using the Recall and NDCG metrics measured at cutoff 10. We also calculate Recall at cutoff 1, because according to Equation (5.23), we expect SASRec to be more overconfident on the highest-ranked metrics, and mitigating overconfidence should have a bigger effect on metrics measured at the highest cutoff.

5.6.1.2 Models

In our experiments, we compare gSASRec with the regular SASRec model, which serves as the backbone of our work.[6] We also use BERT4Rec as a state-of-the-art baseline. For all models, we set the sequence length to 200.

Additionally, we use two simple baselines: a non-personalised popularity model, which always recommends the most popular items, and the classic Matrix Factorisation model with BPR [199] loss. Our implementation of SASRec (and gSASRec) are based on the original code[7], whereas our implementation of BERT4Rec is based on the more efficient implementation[8] that we developed in our replicability study (Chapter 3). To ensure that the models are fully trained, we use an early stopping mechanism to stop training if NDCG@10 measured on the validation dataset has not improved for 200 epochs. Based on hyperparameter tuning, we use a learning rate of 0.0001 for the Steam dataset and a learning rate of 0.001 for the Gowalla and MovieLens-1M datasets. In all cases, we use the Adam optimiser.

---

6. Recall that standard SASRec uses BCE as a loss function
7. https://github.com/kang205/SASRec/
8. https://github.com/asash/bert4rec_repro

Table 5.1: Effects of model architecture and negative sampling on NDCG@10, for the MovieLens-1M (ML-1M) and Steam datasets. * denotes a significant change ($pvalue < 0.05$) in NDCG@10 caused by negative sampling (comparing horizontally) or model architecture (comparing vertically).

Datase ↓t	Negative sampling and loss function→ Architecture↓	1 negative per positive; BCE Loss (as SASRec)	No negative sampling; Softmax Loss (as BERT4Rec)	**Negative sampling and loss effect**
ML-1M	SASRec	0.131	0.169	**+29.0%***
	BERT4Rec	0.123	0.161	**+30.8%***
	Architecture effect	**-6.1%**	**-4.7%**	
Steam	SASRec	0.0581	0.0721	**+24.1%***
	BERT4Rec	0.0513	0.0746	**+45.4%***
	Architecture effect	**-11.7%***	**+3.4%***	

5.6.2 Results

5.6.2.1 RQ5.1. How does negative sampling affect BERT4Rec's performance gains over SASRec

To answer our first research question, we train both BERT4Rec and SASRec on the Steam and MovieLens-1M datasets using the sampling strategies which were originally used in these models: (i) one negative per positive and BCE loss (as in SASRec) and (ii) all negatives per positive and Softmax loss (as in BERT4Rec).[9] We use the original training objectives for both architectures: item masking in BERT4Rec and sequence shifting in SASRec; we also retain the architecture differences in the models (i.e. we keep uni-directional attention in SASRec and bi-directional attention from BERT4rec). The results of our comparison are summarised in Table 5.1.

As can be seen from the table, in all four cases, changing the sampling strategy from the one used by SASRec to the one used in BERT4Rec significantly improves effectiveness. For example, SASRec's NDCG@10 on MovieLens-1M is improved from 0.131 to 0.169 (+29.0%) by removing negative sampling and applying Softmax loss. BERT4Rec achieves a larger improvement of NDCG@10 on Steam (0.0513 → 0.0746: +45.4%) when changing the sampling strategy from 1 negative to all negatives. In contrast, the effect of changing the architecture is moderate (e.g. statistically indistinguishable in 2 out of 4 cases) and frequently negative (3 cases out of four, 1 significant)[10].

9. In this RQ, our goal is to understand BERT4Rec's gains over SASRec better, so we only experiment with their original loss functions and sampling strategies; We apply other loss functions, such as Sampled Softmax, and more negative samples in Section 5.6.2.4.

10. Item masking objective used by BERT4Rec also serves as a data augmentation technique, which can be beneficial to prevent overfitting in small and sparse datasets, see also Appendix A in the journal version of the paper [184].

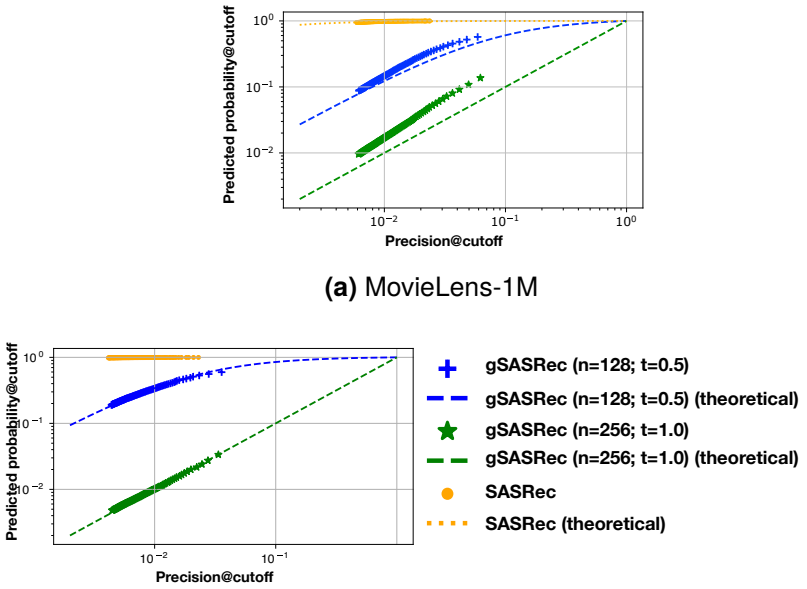
in answer to RQ5.1, we conclude that the absence of negative sampling plays the key role in BERT4Rec's success over SASRec, whereas any gain by applying BERT4Rec's bi-directional attention architecture is only moderate and frequently negative. Therefore, the performance gains of BERT4Rec over SASRec can be attributed to the absence of negative sampling and Softmax loss and not to its architecture and training objective. This is contrary to the explanations of the original BERT4Rec authors in [230], who attributed its superiority to its bi-directional attention mechanism (on the same datasets). We now analyse how gBCE changes the distribution of predicted probabilities.

5.6.2.2 RQ5.2. Effect of gBCE on predicted interaction probabilities

To analyse the effects of gBCE on predicted probabilities, we train three models: a regular SASRec model and two configurations of gSASRec: a first with 64 negatives and $t = 0.5$ and a second with 256 negatives and $t = 1.0$. Our goal is to compare the actual probabilities $P(i)$ with probabilities predicted by the model $\hat{p}_i$. As we discuss in Section 5.3.1, $P(i)$ is unknown, so direct measurement of such relation is hard. Hence, as a substitute for $P(i)$, we use the popular mean Precision@K metric, which, according to Cormack et al. [35], can be seen as a measurement of the conditional probability of an item being relevant, given its rank is less than K. We compare this metric with the average predicted probability of items retrieved at rank less than K. We perform this comparison for cutoffs K in the range [1..100]. Figure 5.4 displays the comparison results for MovieLens-1M and Steam datasets, illustrating the expected theoretical relationship between Precision@K and Predicted Probability@K based on Theorem 5.5.1.

Figure 5.4b shows that the theoretical prediction from Theorem 5.5.1 closely matches the observed relationship between Precision@K and Predicted Probability@K in the Steam dataset. In the MovieLens-1M dataset (Figure 5.4a), a slight discrepancy appears between the theoretical prediction and observed relationship, likely because the smaller number of users in the dataset doesn't meet the requirement of Theorem 5.5.1 for an adequate amount of training samples.

Despite these small discrepancies, the relation follows the trends expected from our theoretical analysis. In particular, Figure 5.4 shows that as expected from Corollary 5.5.1.3, SASRec is indeed prone to overconfidence and, on average, predicts probabilities very close to 1 for all ranks less than 100. In contrast, the probabilities predicted by gSASRec are considerably less than 1. For example, for MovieLens-1M, gSASRec trained with 128 negatives and $t = 0.5$, on average, predicts probability 0.57 at K=1, while the version with 256 negatives and $t = 1.0$ predicts prob-



(a) MovieLens-1M

(b) Steam

Figure 5.4: Relation between Mean Precision@K metric and Mean predicted probability@K for cutoffs K in $[1..100]$ range. The figure also includes theoretical prediction for the relation according to Theorem 5.5.1.

ability 0.13 at the same cutoff. Together, this analysis shows that gSASRec trained with gBCE successfully mitigates the overconfidence problem of SASRec. Furthermore, from the figure, we also see that when parameter t is set to 1, the mean predicted probability is well-calibrated with mean precision at all rank cutoffs (particularly on the Steam dataset). This is well-aligned with Corollary 5.5.1.1, which states that when parameter β in gBCE is set equal to the sampling rate (i.e. setting parameter $t = 1$) results in learning in fully calibrated probabilities. Overall, in answer to RQ5.2, we conclude that gBCE successfully mitigates the overconfidence problem in a manner that is well-aligned with our theoretical analysis. We next turn to the impact of gBCE on effectiveness.

5.6.2.3 RQ5.3. Effect of negative sampling rate and parameter t on the performance of gSASRec

In comparison to SASRec, gSASRec has two additional hyperparameters: the number of negative samples and the parameter t, which adjusts probability calibration. To explore the impact of these parameters on performance, we conduct a grid search by selecting the negative sample count from [1, 4, 16, 64, 256] and the calibration parameter t from [0, 0.25, 0.5, 0.75, 1.0].

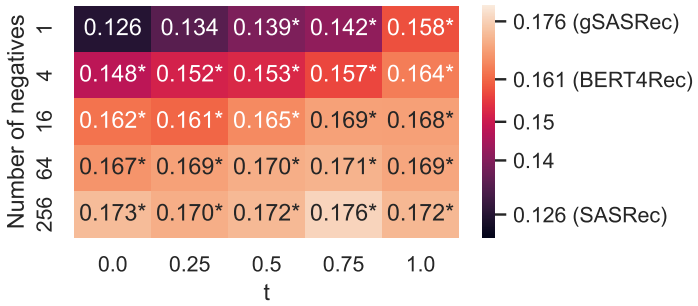
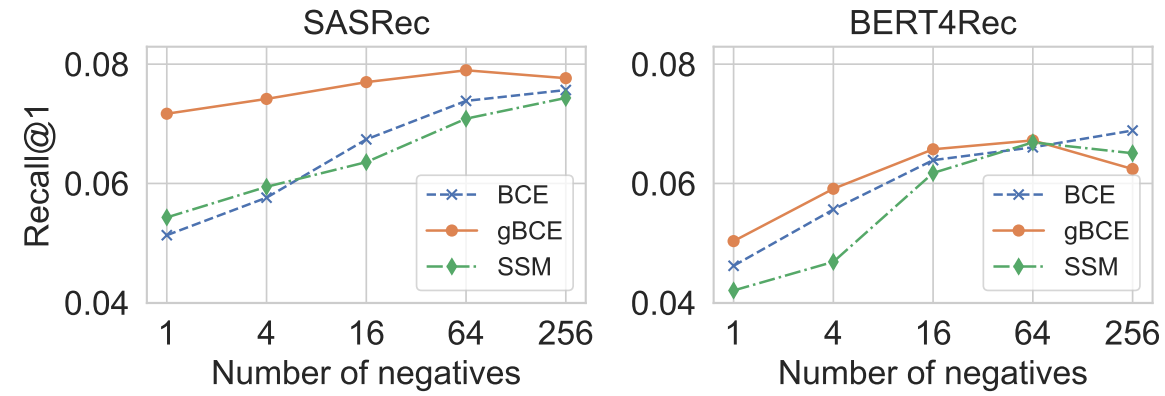


Figure 5.5: gSASRec: Effect of varying number of negatives and calibration parameter t on NDCG@10, MovieLens-1M. * denotes a significant improvement over SASRec ($pvalue < 0.05$, Bonferroni multiple test correction).

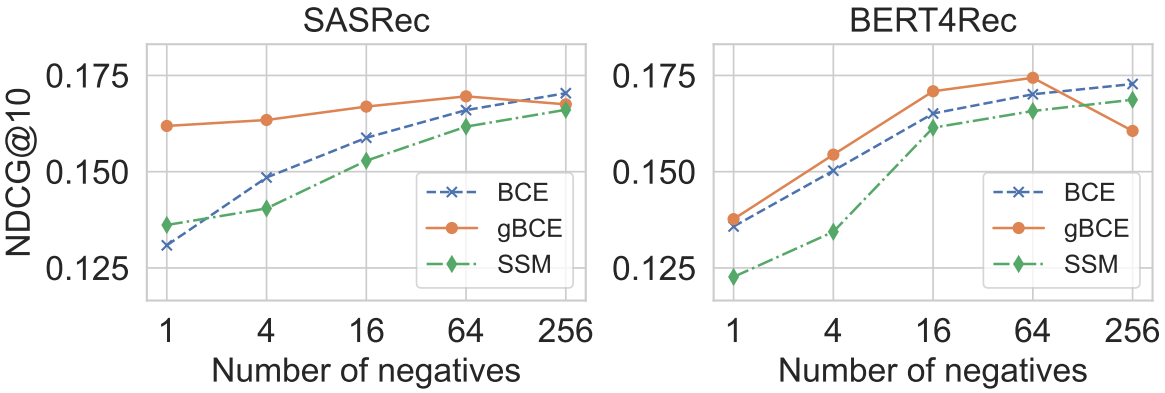
Figure 5.5 portrays our grid search on MovieLens-1M (on other datasets, we observed a similar pattern and omitted their figures for brevity). From the figure, we observe that, as expected from the theoretical analysis, both t and the number of negatives have a positive effect on model effectiveness. For example, when the number of negatives is set to 1, varying t from 0 to 1 allows increasing NDCG@10 from 0.126 to 0.158 (+25%, significant, compared to SASRec, which is also gSASRec with 1 negative and calibration $t = 0$). Interestingly, the result of gSASRec with 1 negative and $t = 1$ is similar to what BERT4Rec achieves with all negatives (0.158 vs. 0.161: -1.86%, not significant). We also observe that when the number of negatives is higher, setting a high value of t is less important. For example, when the model is trained with 256 negatives (7.49% sampling rate), the model achieves high effectiveness with all values of t. This is also not surprising – by design, more negative samples and higher values of t should have a similar effect in gBCE. During our experiments, we also observed that setting parameter t very close to 1 also increased the training time of the model. Keeping this in mind, in practical applications, we recommend setting t between 0.75 and 0.9, and the number of negatives between 128 and 256 – this combination works well on all datasets, converging to results that are close to the best observed without increasing training time. This answers RQ5.3.

5.6.2.4 RQ5.4. Effect of gBCE loss on negatively sampled SASRec and BERT4Rec

To investigate the effect of gBCE on SASRec and BERT4Rec models with negative sampling, we train the models with the number of negative samples selected from [1, 4, 16, 64, 256] and the loss function selected from [BCE, gBCE, Sampled Softmax Loss (SSM)] on the MovieLens-1M dataset. In this experiment, we use a fully-calibrated version of gBCE ($t = 1.0$). Figure 5.6 summarises the results of the experiment. As we can see from the figure, gBCE performs better



(a) Recall@1



(b) NDCG@10

Figure 5.6: Performance of SASRec and BERT4Rec architectures, trained on the MovieLens-1M dataset with a variable number of negatives and various loss functions. BCE is a classic Binary-Cross Entropy loss, gBCE - Generalised Binary Cross-Entropy (t=1.0), SSM - Sampled Softmax loss with uniform sampling.

than both BCE and Sampled Softmax loss when the number of negatives is small. For example, for BERT4Rec trained with 4 negatives, gBCE has a higher Recall@1 (0.059) than both BCE (0.055; -5.8% compared with gBCE) and Sampled Softmax (0.046, -20%) and has the highest NDCG@10 of 0.154, while BCE has NDCG@10 of 0.150 (-2.6% compared with gBCE) and Sampled Softmax has NDCG@10 of 0.134 (-12.9%). In the case of SASRec, the difference is even larger when the number of negatives is small (recall that SASRec trained with gBCE is also gSASRec). For example, with 16 negatives, gBCE achieves Recall@1 0.0769, BCE achieves 0.0673 (-12.5%) and Sampled Softmax achieves 0.0635 (-17.5%). We hypothesise that gBCE affects SASRec more than BERT4Rec due to their training objectives. SASRec predicts the next item in a sequence, while BERT4Rec predicts randomly masked items. Consequently, altering SASRec's loss function directly impacts its performance in next-item prediction. In contrast, changing BERT4Rec's loss function only affects the masking task, which is less directly related to the next item prediction task [176, 181]. In particular, we speculate that this discrepancy

between the training objective and the actual next-item recommendation task causes the slightly inferior effectiveness of BERT4Rec trained using 256 negatives with gBCE when compared to BCE or SSM: as the model was trained for one task and evaluated on another, there is no guarantee that the use of the better loss functions leads to better effectiveness on test data. In general, when more negatives are sampled, gBCE becomes less beneficial. For example, with 256 negatives, all three loss functions achieve similar NDCG@10 (0.1674 gBCE; 0.1703 (+1.7%) BCE and 0.1660 (-0.08%) Sampled Softmax). This is an expected result because 256 negatives represent a significant proportion of all negatives (7.5%), and overconfidence becomes less of an issue for BCE and Sampled Softmax.

In conclusion, for RQ5.4, gBCE outperforms BCE and Sampled Softmax in SASRec and BERT4Rec with few negatives; improvement is larger in SASRec. However, with many negatives, traditional loss functions like BCE and Sampled Softmax work well unaltered, but high sampling rates are impractical due to memory and computational constraints.

5.6.2.5 RQ5.5. gSASRec performance in comparison to state-of the-art Sequential Recommendation models.

To answer RQ5.5, we compare gSASRec with the baseline models. We also add to the comparison a version of SASRec trained with full Softmax (without sampling) because, as we discuss in RQ5.1, it exhibits SOTA performance; however, we omit non-standard versions of BERT4Rec and SASRec trained with BCE or Sampled Softmax, because as we report in RQ5.4, they are not beneficial compared to gBCE. We also exclude BERT4Rec with gBCE from our analysis because, as per RQ5.4, gSASRec achieves superior results when measured by Recall@1 and similar results when evaluated by NDCG@10. After tuning hyperparameters on the validation set, we report the results of gSASRec with 128 negatives and $t = 0.9$ for Steam and Gowalla, and with 256 negatives and $t = 0.75$ for MovieLens-1M. Table 5.2 summarises the results of our evaluation. The table shows that gSASRec achieved the best or the second best result on all datasets according to all metrics. Indeed, on the smaller datasets (MovieLens and Steam), where we were able to train BERT4Rec and SASRec without sampling, gSASRec performs similarly to the best unsampled model (e.g. +4.1% NDCG@10 on MovieLens-1M compared to SASRec-Softmax (not significant) or -1.74% compared to BERT4Rec on Steam, signific-

Table 5.2: Evaluation results. Bold denotes the best model on the dataset for that metric, and underlined is the second-best model. * denotes a significant difference with the best-performing model ($pvalue < 0.05$). SASRec-Softmax is a SASRec-based model trained without negative sampling and Softmax loss (as BERT4Rec).

Model				Datasets											
				MovieLens-1M				Steam				Gowalla			
Category	Model	Negative Sampling	Loss	Recall @1	Recall @10	NDCG @10	Time (min)	Recall @1	Recall @10	NDCG @10	Time (min)	Recall @1	Recall @10	NDCG @10	Time (min)
Baselines	Popularity	N/A	N/A	0.005*	0.036*	0.017*	0.0	0.0077*	0.0529*	0.0268*	0.0	0.0011*	0.0081*	0.0041*	0.1
	MF-BPR	Yes	BPR	0.010*	0.075*	0.037*	0.1	0.0071*	0.0393*	0.0206*	0.4	0.0083*	0.0282*	0.0170*	2.1
Unsampled	BERT4Rec	No	Softmax	0.058*	0.294	0.161*	86	0.0281	**0.1379**	**0.0746**	642	Infeasible (requires >100GB of GPU memory, see Section 5.1)			
	SASRec-Softmax	No	Softmax	0.073	0.293	0.169	9	0.0280	0.1323*	0.0721*	80				
Sampled	SASRec	Yes	BCE	0.046*	0.247*	0.131*	13	0.0193*	0.1121*	0.0581*	32	0.0505*	0.1831*	0.1097*	145
	gSASRec	Yes	gBCE	**0.082**	**0.300**	**0.176**	23	**0.0283**	0.1355	0.0735	58	**0.0782**	**0.2590**	**0.1616**	191

Table 5.3: Comparison of gSASRec with recent reported results on MovieLens-1M. Bold indicates the best value.

	Recall@10	NDCG@10
DuoRec [189]	0.294	0.168
ALBERT4Rec [174]	0.300	0.165
CBiT [50]	**0.301**	0.169
gSASRec	0.300	**0.176**

ant). Interestingly, on MovieLens-1M, both SASRec-Softmax (our version of SASRec trained without negative sampling) and gSASRec significantly improve Recall@1, suggesting that at least in some cases SASRec's unidirectional architecture may be beneficial. This also echoes our observations while analysing RQ5.1.

Crucially, gSASRec always significantly outperforms the regular SASRec model (+34% NDCG@10 on MovieLens-1M, +26% on Steam, +47% on Gowalla). The result on Gowalla is particularly important, as it demonstrates that gSASRec is suitable for datasets with more than 1 million items, and it improves SASRec's results by a large margin on this large dataset. From Table 5.2 we also see that all versions of SASRec (including gSASRec) require less training time than BERT4Rec. For example, on MovieLens, gSASRec is 73.2% faster to train compared to BERT4Rec (23 minutes vs. 85 minutes) and, on Steam, gSASRec is 90.9% faster (58 minutes vs. 642 minutes). However, we also see that gSASRec requires more training time than SASRec (e.g. 58 vs 32 minutes on Steam); we explain that by the fact that more accurate probabilities estimation with gBCE requires more training epochs to converge (238 epochs vs. 170 epochs in our experiment). We also observe that when it is possible to compute softmax over all items, gSASRec's effectiveness is on par with SASRec-Softmax: on both smaller-scale MovieLens-1M and Steam datasets, the results achieved by gSASRec are slightly better (e.g. +2.3% Recall@10 on MovieLens-1M), but not statistically significantly. Therefore, our experiments do not show either clear advantages or disadvantages of using full Softmax loss vs gBCE when it is possible to compute item scores over the whole catalogue.

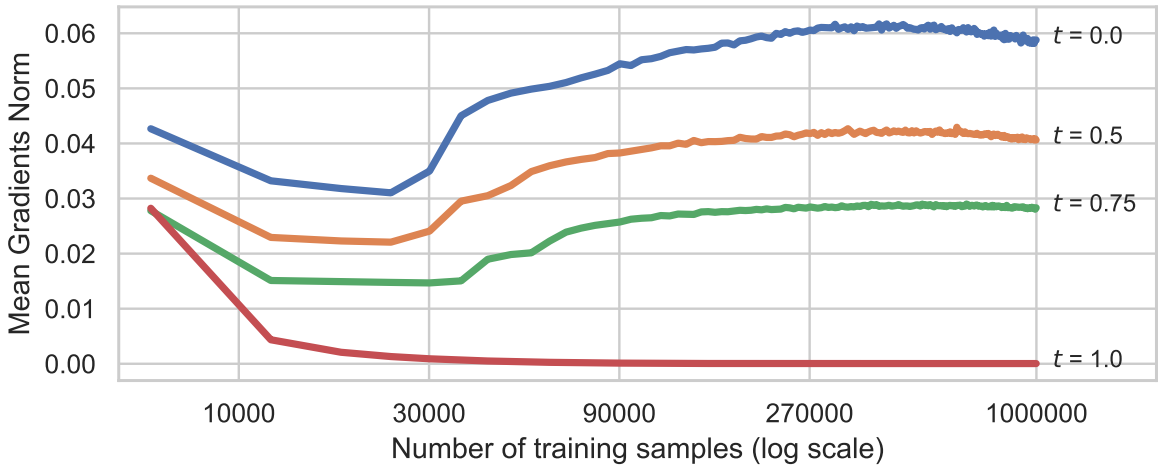


Figure 5.7: L2 norm of the gradient of the model parameters in gSASRec, measured during model training with different calibration parameters t; MovieLens-1M dataset.

Finally, for MovieLens-1M, we compare the results achieved by gSASRec with those of the most recently proposed models in the literature, which report the best results, namely ALBERT4Rec [174] (an effective model similar to BERT4Rec, but based on ALBERT [120]), and two contrastive models: DuoRec [189], and CBiT [50]. All papers from our selection use the same data-splitting strategy and unsampled metrics, so the results are comparable. Table 5.3 summarises this comparison. As we can see from the table, all these publications report Recall@10 close to 0.3, which is similar to what we obtain with gSASRec. However, only gSASRec achieves an NDCG@10 above 0.17. Furthermore, as observed from Figure 5.5, this result is not a one-off occurrence but a consistent outcome when the model is trained with 256 negatives, making it unlikely to be a statistical fluctuation. This is likely due to its better focus on highly-ranked items, as gBCE is specifically designed for improving overconfidence in highly-scored items.

Overall, our experiments show that gSASRec performs on par with SOTA models, retaining the negative sampling required for use on big catalogues and converging faster than BERT4Rec.

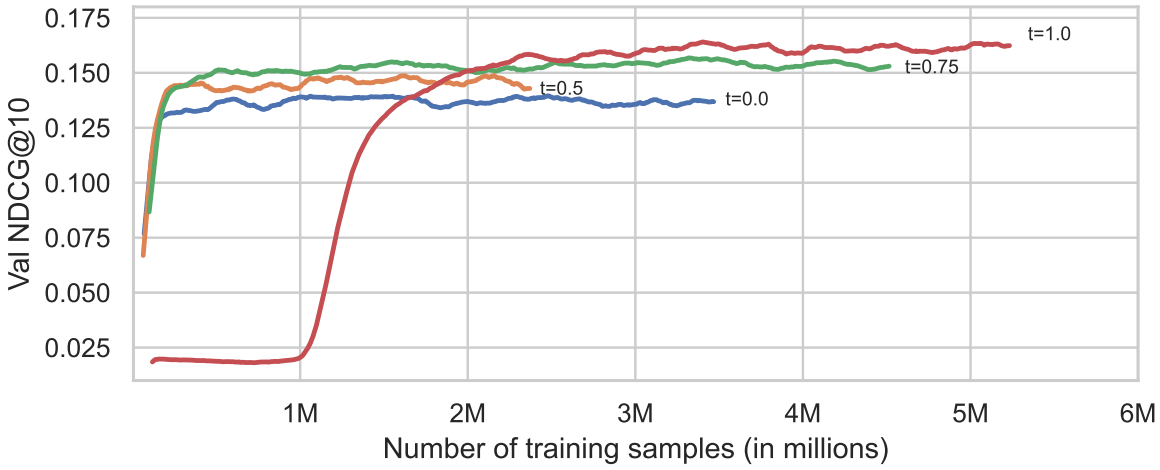


Figure 5.8: NDCG@10 of gSASRec with different calibration parameters t measured on the validation set during model training; MovieLens-1M dataset.

5.6.2.6 RQ5.6. Effect of the calibration parameter t on model gradients and model convergence speed.

In Section 5.5.4, we speculated that low values of calibration parameter t lead to large model parameter gradients that prevent gradient decent finding optimal model parameters, while high values of t may lead to slow model convergence. In this section, we validate this hypothesis empirically. To do this, we train the gSASRec model with one randomly sampled negative per positive and calibration parameter t selected from $\langle 0.0, 0.5, 0.75, 1.0 \rangle$. In each case, we monitor the L2 norm of the model parameter gradients and NDCG@10 measured on the validation set.

Figures 5.7 and 5.8 report the L2 norm of model parameters gradient and validation NDCG@10 over the course of model training. Figure 5.7 shows that lower values of parameter t indeed lead to higher gradients. For example, when $t = 0.0$ (i.e., standard BCE), the mean gradient L2 norm reaches 0.6 after 270000 training samples, while with $t = 0.75$, the mean gradient L2 norm only approaches 0.3 after the same number of training steps. With $t = 0$, the L2 norm quickly drops to values close to 0. Figure 5.8 shows that with all t except $t = 1.0$, the model converges very quickly, only requiring a few hundred thousand samples to reach the plateau of NDCG@10 metric. In contrast, when $t = 1.0$, the model requires more than 3 million steps to converge: as we discussed in Section 5.5.4, in this case, the loss focuses too much on the negative samples, requiring more time to learn to distinguish them from positives. However, we also see that with higher values of t, the model achieves higher values of NDCG@10; this is consistent with our findings in RQ5.3.

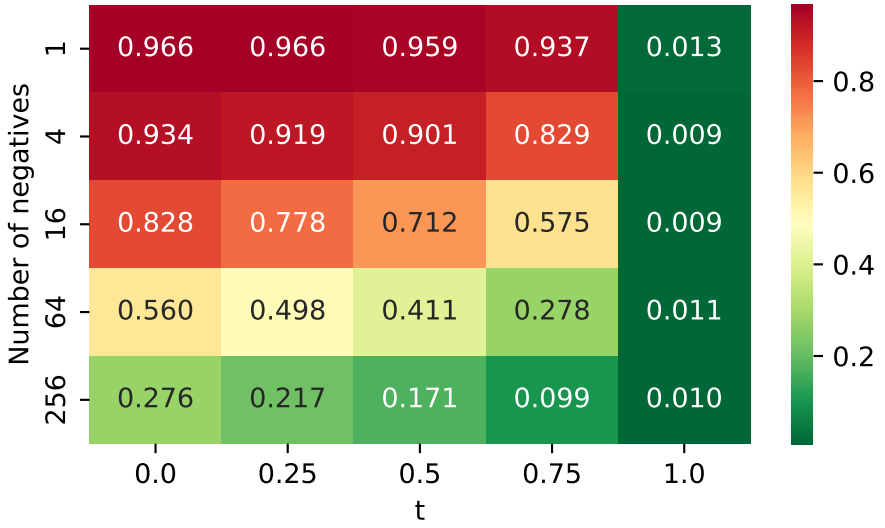


Figure 5.9: Effect of varying number of negatives and calibration parameter t on Expected Calibration Error (ECE), MovieLens-1M dataset. Lower ECE corresponds to better calibration.

From this analysis, we also note that the effect of increasing parameter t is somewhat similar to decreasing the model's learning rate, as increasing t leads to higher effectiveness but slower training. However, in contrast to learning, t is the parameter of the loss function itself and not of the training algorithm. Indeed, Theorem 5.5.1 proves that changing parameter β (which we indirectly control using parameter t) leads to a different optimal prediction with respect to the loss function. In our early experiments, decreasing the learning rate increased *stability* of model convergence, i.e. with larger learning rates, the model diverges. However, we did not observe changes in *effectiveness* of the model beyond statistical fluctuations. Establishing better theoretical connections between t and learning rate is an interesting research direction, which we leave for future work.

In summary, answering RQ5.6, we conclude that smaller values of t lead to large gradients and quick convergence, but the model converges at the lower value of NDCG@10. In contrast, setting a high value of t leads to small model gradients and slow convergence, but the model converges at a higher level of NDCG@10. Our recommended value of $t = 0.75$ is a reasonable compromise: in this case, the model converges quickly and at the high values of NDCG@10.

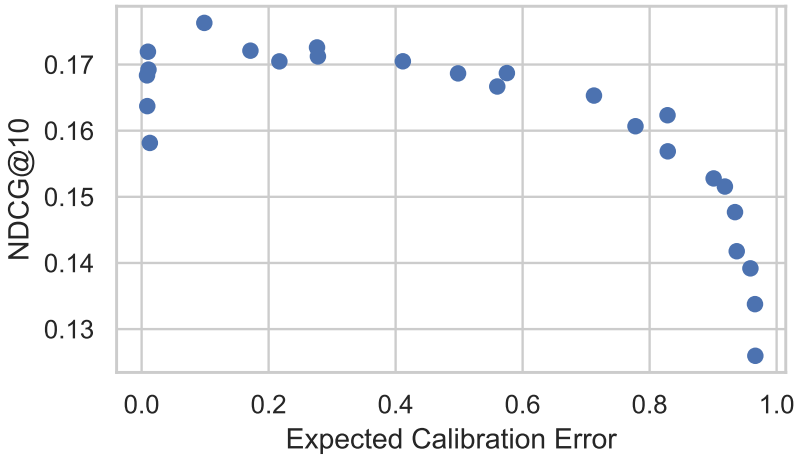


Figure 5.10: Relation between ECE and NDCG@10 in gSASRec when varying the number of negative samples and the calibration parameter t, MovieLens-1M dataset. Each point corresponds to a single gSASRec configuration; we use the same configurations as in Figure 5.9.

5.6.2.7 RQ5.7. Impact of gBCE on Expected Calibration Error.

In Section 5.4.2, we mentioned that overconfidence is closely related to the model's calibration — the ability of the model to accurately predict *actual* interaction probabilities. To validate that gBCE can indeed improve models' calibration, we measure gSASRec's Expected Calibration Error (ECE, Equation (5.8)) that measures the difference between predicted probabilities and the actual proportion of positive outcomes on the Movielens-1M dataset while varying the number of sampled negatives and the calibration parameter t. When measuring ECE, we only took into account predictions up to the ranking cutoff of 10.

Figure 5.9 summarises the results of the experiment. The Figure represents a heat map of Expected Calibration Error as a parameter of the number of negatives and t, with the red colour corresponding to higher (worse) values of ECE and green corresponding to smaller (better) values of ECE. As we can see, more negatives lead to better model calibration, and a higher value of t also leads to better calibration. In particular, setting $t = 1.0$ leads to a small calibration error (approximately 0.01) regardless of the number of sampled negatives – these findings are in line with the theoretical analysis in Section 5.5. Also, note that with $t = 0$ and one sampled negative, gSASRec turns into regular SASRec. From the figure, we see that it has the worst calibration error (0.966) compared to any other configuration of gSASRec.

We also notice similarities between Figure 5.5, which analyses model effectiveness, and Figure 5.9, which analyses model calibration. Figure 5.10 further shows the relation between models' ECE and NDCG@10. Each point in the figure represents a single configuration of the gSASRec model trained on the MovieLens-1M dataset; the same configurations are used as in Figure 5.5 and Figure 5.9. As we can see in Figure 5.10, there is a strong negative correlation between ECE and NDCG@10; the Pearson correlation between the two metrics is -0.748. This correlation provides evidence that calibration is important for model effectiveness with cross-entropy-based losses. gBCE improves calibration by reducing model overconfidence, which in turn leads to better overall model effectiveness.

Overall, in answer to RQ5.7, we find that gBCE can reduce the Expected Calibration Error compared to standard BCE when using it with a large value of calibration parameter t. We also find a strong negative correlation between the Expected Calibration Error and model effectiveness, measured by NDCG@10.

5.7 Conclusions

In this chapter, we studied the impact of negative sampling on Sequential Recommender systems. We showed (theoretically and empirically) that negative sampling coupled with Binary Cross-Entropy loss (a popular combination used by many sequential models) leads to a shifted score distribution, called overconfidence. We showed that overconfidence is the only reason why SASRec underperforms compared to the state-of-the-art BERT4Rec. Indeed, when we control for negative sampling, the two models perform similarly. We proposed a solution to the overconfidence problem in the form of gBCE and theoretically proved that it can mitigate overconfidence. We further proposed gSASRec, which uses gBCE, and experimentally showed that it can significantly outperform the best unsampled models (e.g. +9.47% NDCG@10 on the MovieLens-1M dataset compared to BERT4Rec) requiring less training time (e.g. -90.9% on the Steam dataset compared to BERT4Rec), while also being suitable for large-scale datasets. We also showed that gBCE may be beneficial for BERT4Rec if it is trained with negative sampling (e.g. +7.2% compared to BCE when trained with 4 negatives). The theory and methods presented in this chapter could also be applied to not just Sequential Recommendation models but also to other types of recommendation as well as to NLP or search tasks. For example, gBCE loss with multiple negative samples has already been shown to work well for training document retrieval models [173]. We hope to see more applications of gBCE in recommender systems and adjacent domains in both academia and industry.

In Section 2.3.5, we have identified three Limitations of Transformer-based recommendation models that are related to large-size catalogues. One of them (Limitation L2.2: "Computing all scores too expensive during training") we have addressed in this chapter by using negative sampling and gBCE loss function. In the following two chapters, we address the remaining large catalogue-related limitation of the Transformer-based Sequential Recommendation models (Limitation L2.3: "Item embedding tensor too large" and Limitation L2.4: "Computing all scores too expensive during inference"). In particular, in the next chapter, we address Limitation L2.3 by developing RecJPQ, an efficient item embedding tensor compression technique.

Chapter 6

Compressing Item Embedding Tensors

In this chapter, we continue to address the limitations of Transformer-based recommendation models with large catalogues. In particular, as stated by Limitation L2.3 in Section 2.3.5, with large catalogues, the Item Embedding Tensor may contain more parameters than the rest of the model and may require storing hundreds of billions of float numbers, making model training infeasible. Inspired by tokenisation techniques from language models, in this chapter, we address the Limitation L2.3 by proposing *RecJPQ*, a novel Item Embedding Tensor compression technique for sequential recommendation. RecJPQ decomposes atomic item ids into a small number of shared sub-item ids. Because sub-ids are shared between items, RecJPQ achieves up to 50× model size compression in our experiments without hindering model effectiveness.

This chapter is organised as follows: Section 6.1 introduces the large Item Embedding Tensor problem in Sequential Recommender Systems; Section 6.2 discusses existing methods for compressing item embeddings and their limitations; Section 6.3 covers Product Quantisation (a generic vector compression method) and Joint Product Quantisation (a version of Product Quantisation designed for document retrieval) – two method that inspired the RecJPQ design;Section 6.4 describes RecJPQ, our solution for compressing item embeddings; Section 6.5 describes experiments with RecJPQ; Section 6.7 concludes the chapter with final remarks.

The material of this chapter is based on our paper [180], which was published as a full research paper in the proceedings of the ACM WSDM'24 conference.

6.1 The Large Item Embeddings Problem

One of the key components of the modern Deep Learning-based Sequential Recommender Systems is the Item Embedding tensor. Figure 6.1 illustrates item embeddings in a typical neural Sequential Recommendation model. As the figure shows, item embeddings usually have two roles in the model architecture: (i) to convert the sequence of input item ids to a sequence of item representation vectors and (ii) to convert the sequence embedding produced by the model into the distribution of predicted item scores. In both cases, a recommender system that works with an item set I requires an embedding tensor with $|I| \cdot d$ parameters, where d is the size of each embedding.

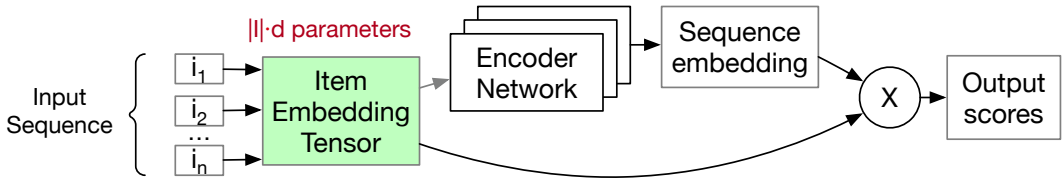


Figure 6.1: Item embeddings in a typical Sequential Recommender System. These item embeddings are used in two ways: (i) to obtain sequence representation and (ii) to generate item scores. The embedding tensor requires $|I| \cdot d$ trainable parameters, where $|I|$ is the items catalogue size, and d is the size of an embedding. When catalogue size $|I|$ is large, item embeddings comprise most of the model's parameters.

When a recommender has many items in the catalogue, various challenges arise in training the neural recommendation model. Firstly, the item embedding tensor may contain more model parameters than the rest of the model. For example, there are more than 800 million videos on YouTube [84]. If a recommender model uses 128-dimensional embeddings, the whole Item Embeddings Tensor will have more than 100 billion parameters, which is comparable with the number of parameters of the largest available machine learning models [17], even without accounting for the parameters of the model's intermediate layers. This is a problem specific to recommender systems: in the related area of dense passage retrieval [102, 105], passage embeddings are obtained by encoding passage text using a pre-trained language model; however, item side information, such as text, is not necessarily available in a typical recommender systems scenario; therefore, item embeddings should be directly learned from the interactions. Secondly, a large number of such trainable parameters also makes the model prone to overfitting.

There are some existing methods [99, 249, 260] for item embedding compression (we discuss these methods in Section 6.2). However, most of these methods compress the embedding tensor *after* the model is fully trained (including training the full embedding tensor). However, as argued above, training may be prohibitively expensive in large-catalogue recommender systems. Hence, this chapter addresses the problem of a large item embedding tensor in Sequential Recommendation models *at the training stage*.

To mitigate this problem, we propose a novel RecJPQ technique inspired by the success of a recent Joint Product Quantisation (JPQ) work [273] for passage retrieval. JPQ itself is based on Product Quantisation (PQ) [93], a popular method of compressing vectors by splitting them into sub-embeddings and encoding them using a discrete *item*[1] *codebook* (the codebook maps from item ids to the associated *sub-item ids*; see Section 6.3.1 for the details). The main innovation of JPQ compared to the standard PQ method is that it learns the *sub-item embeddings* as part of the overall model training process. In contrast, PQ requires training the model first and

1. For consistency, we explain prior work using *item ids* instead of *passage ids*.

only then compressing the embeddings (frequently, this second step uses external tools, such as FAISS [97]). This means that JPQ does not need to keep the embedding matrix in memory during model training. We argue that this innovation is valuable for recommender systems. Indeed, as mentioned above, real-life recommender systems can have hundreds of millions of items in their catalogues and keeping full embeddings tensor in memory may be prohibitively expensive. This is particularly important for deep-learning-based Sequential Recommender Systems because these models require keeping the whole model in GPU (or TPU) memory during training. GPU memory is costly even when compared to regular computer RAM.

Unfortunately, it is hard to adapt JPQ to the recommendation scenario, as it is specific to textual information retrieval. In particular, JPQ assumes the existence of a pre-trained (passage retrieval) model and index, which it uses to assign items to sub-item ids (see more details in Section 6.3.2). These pre-trained models rarely exist in item recommendations. Hence, in RecJPQ, we experiment with performing the initial assignment of sub-ids using three different strategies: (i) discrete truncated SVD (sub-ids obtained by discretising the item representations obtained by an SVD decomposition of the user-item matrix); (ii) discrete BPR (sub-ids obtained by discretising the item embeddings obtained from BPR); and (iii) random assignments. We describe these assignment strategies in detail in Section 6.4.

RecJPQ is a model component that replaces traditional item embeddings in Sequential Recommender Systems. In general, it can be applied to any recommender system based on item embeddings, but in this chapter, we focus specifically on sequential models, as in these models, item embeddings comprise the biggest part of the model (e.g. sequential models usually do not have user embeddings). In contrast with existing methods, RecJPQ does not require training full uncompressed embedding and does not modify the original model loss function. Our experimentation on three datasets (see Section 6.5) demonstrates that RecJPQ can be successfully applied to different models, including SASRec, BERT4Rec and GRU, achieving a large factor of embeddings compression (e.g. 47.94$\times$ compression of SASRec on Gowalla) without any effectiveness degradation. Moreover, on 2 out of our 3 experimental datasets, applying RecJPQ *increases* model performance (e.g. +0.96% NDCG@10 on Booking.com dataset, significant improvement); we attribute these improvements to model regularisation.

In short, the contributions of this chapter are:

1. We propose RecJPQ, a novel technique for reducing the size of Sequential Recommendation models during training based on JPQ;

Table 6.1: Existing embedding compression methods. The desired method characteristics are highlighted in bold.

Model Agnostic	Method	Backbone	Sequential backbone	Trains full embeddings	Compression unit
No	EODRec [260]	SASRec [100]	**Yes**	Yes	**Item**
	LightRec [133]	DSSM [88]	No	Yes	**Item**
	MDQE [249]	SASRec [100]	**Yes**	Yes	**Item**
Yes	PreHash [220]	BiasedMF [114]; NeuMF [75]	No	**No**	User
	Quotient Remainder [219]	DCN [250]; DLRM [162]	No	**No**	Features
	MGQE [99]	SASRec [100]; NeuMF [75]; GCF [75];	**Yes**	Yes	**Item**
Yes	RecJPQ (ours)	SASRec [100]; BERT4Rec [230]; GRU [81]	**Yes**	**No**	**Item**

2. We propose three strategies for assigning sub-item ids to items, two of which (discrete truncated SVD and discrete BPR) assign similar codes to similar items, and one assigns codes randomly;
3. We extensively evaluate RecJPQ on three datasets and show that RecJPQ allows reducing the models' size without hindering the model performance.

6.2 Embeddings Compression in Recommender Systems

This section covers existing work on compressing and discretising embeddings in recommender systems, identifies the limitations in existing work and positions our contributions in the context of existing methods. Table 6.1 summarises existing methods and positions RecJPQ, our proposed compression technique. The table highlights with boldface the desirable characteristics necessary for training a large-scale[2] sequential model, specifically: the method can be applied to work with different *backbone* sequential models, and does not require training full embeddings (as we work with the assumption that the full embeddings tensor does not fit into GPU memory); and we want the model to focus on item embeddings rather than embeddings of other entities, such as users or features. As illustrated in the table, the methods for compressing the models can be broadly divided into two groups: *model dependent* and *model agnostic*.

In the model-dependent methods [133, 260], the embedding compression mechanism is integrated as a component into the recommendation model itself: the training architecture has to be aware of the compression, and the loss function must encompass components responsible for the embedding compression. However, we argue that these methods exhibit a number of limitations:

2. For simplicity, we say that a catalogue is "large-scale" if it has more than 1 million items, as it becomes challenging to train recommender systems on that scale [176].

L6.1: Model-dependent methods are, by their nature, tied to a specific model, making them inflexible when adapting to a specific task. For example, the core component of LightRec [133] (Recurrent Composite Encoding) is tightly integrated into the DSSM [88] architecture, and it is unclear whether or not it can be used outside of DSSM.

L6.2: Model-dependent methods usually require training (uncompressed) item embeddings and then use knowledge distillation or teacher-student techniques to obtain compressed representations of the embeddings. This approach substantially reduces the final model size, thereby helping inference on smaller devices, but requires a large amount of GPU memory while training, thereby limiting the overall number of items in the catalogue. For this reason, the main positioning for EODRec model [260] is the on-device recommendation: while the final model produced by this method is small, it requires storing full item embeddings while training. Post-training quantisation methods [268], which reduce the model's size via quantising its weights into lower-precision numbers (e.g. float16, or int8,) also have this limitation – they need to have access to the full model before quantising. Similarly, Mixed Precision Training [156] builds a smaller precision model, but it requires keeping full precision weights in memory. Placing the embeddings into Approximate Nearest Neighbours [97] or Hierarchical Navigable Small Worlds [151] indexes also requires access to the full embeddings during model training, and therefore also exhibits this limitation.

L6.3: Model-dependent methods require multi-component loss functions, some of which are responsible for the recommendation task and others for the model compression. This is a form of multi-objective optimisation, which is a challenging problem [41, 215], as finding the balance between the loss components for different objectives usually requires extensive hyperparameters search.

On the other hand, the existing model-agnostic methods [219, 220] do not depend on the specific model architecture, and likewise do not add extra components to the loss functions. Typically, these methods implement a mechanism that takes the place of the embeddings tensor in the backbone model, and hence can be used with many models. However, on inspection of the relevant work, we elicit additional limitations of these methods:

L6.4: Many methods are not designed for compressing item embeddings. For example, PreHash [220] is a method specific for compressing user embeddings (i.e. it uses the user's history to construct user embeddings). The method uses an attention network over the history of user interactions. Adapting this network structure for items embeddings is a hard task: a user may only interact with a few items; in contrast, a popular item may be interact with by millions of users. The attention mechanism depends quadratically on the sequence length, and therefore, applying it to users who interacted with a popular item would be prohibitively expensive.

L6.5: Finally, many methods lack structure in compressed item representations. This leads to situations where unrelated items have similar representations and, conversely, when similar items have dissimilar representations. Both these cases may limit the models' generalisation ability and hinder the models' performance. One example of such unstructured methods is hashing-based Quotient Remainder [219], which compresses embeddings of categorical features (e.g., genres). Quotient Remainder assigns feature codes based on the quotient and the remainder of the division of the feature id by some number. When applied to item ids (items can also be seen as categorical features), the quotient and the remainder are unrelated to the item characteristics. Hence, similar items are unlikely to have similar codes. Nevertheless, Quotient Remainder is one of the few methods that can be used to train a model on a large-scale dataset, and therefore, we use this method as a baseline in our experiments.

Overall, among the related work, we argue that existing methods exhibit a number of Limitations (L6.2-L6.5). In summary, the model-dependent methods require training full embeddings first, limiting the maximum number of items that can be considered in the catalogue of the recommender system. On the other hand, methods which do not require training full embeddings first, such as Quotient Remainder, rely on heuristics and may result in unrelated items having similar representations. On the surface, the nearest related work to ours is VQ-Rec [83] as it also applies JPQ-style technique to Sequential Recommendation; however, similarly to JPQ, it relies on the availability of textual features and pre-trained language models. In contrast, our work's main novelty is adapting JPQ to the scenario when (e.g., textual) side information is not available. In the next section, we describe Product Quantisation – a vector compression method and JPQ [273], a method of embedding compression for information retrieval, which directly learns embeddings in compressed form, reducing GPU memory requirements. Then, in Section 6.4, we propose RecJPQ – an adaptation of JPQ to the Sequential Recommendation task, which successfully addresses Limitations L6.2-L6.5.

Table 6.2: Analysis of PQ's impact on memory requirements for storing Item Embeddings Tensor for selected recommendation datasets, based on 512-dimension float32 vector embeddings. The table compares base memory usage with different code lengths, shown as percentages relative to the base.

Dataset	Num Items	Size of Item Embedding Tensor			
		Base	Code length=2 512 centroids 1.00 MB	Code length=8 2,048 centroids 4.00 MB	Code length=32 8,192 centroids 16.00 MB
MovieLens-1M	3,416	6.67 MB	14.988%	59.953%	239.813%
Booking.com	34,742	67.86 MB	1.474%	5.895%	23.580%
Gowalla	1,280,969	2.44 GB	0.040%	0.160%	0.640%

6.3 Product Quantisation and JPQ

We now describe Product Quantisation (PQ) and Joint Product Quantisation (JPQ), two methods that serve as backbones for our method. Section 6.3.1 covers PQ, a classic embedding compression technique. Section 6.3.2 describes JPQ – a recently proposed information retrieval method that learns compressed embeddings directly instead of compressing them after the model training.

6.3.1 Product Quantisation

Product quantisation [65, 93] is a well-cited method of compressing vectors used by many libraries, such as FAISS [97] and nanopq [153]. Its main idea is to split a collection of d-dimensional vectors, V, into m collections $V_i; i = 1..m$ of smaller vectors of $\frac{d}{m}$ dimensions each. The original vectors can be recovered back via concatenation: $V = concat(V_1, V_2, ...V_m)$. Product quantisation then clusters each V_i into b clusters (e.g. using the k-means algorithm [149]) and replaces each vector v_{ij} with the centroid of the assigned cluster $c_{ij} \approx v_{ij}$. With this replacement, the original vectors collection can be approximated as a concatenation of the *sub-vector matrices* $C_1...C_m$, which are constructed from $V_1...V_m$ by replacing each vector v_{ij} by the closest cluster centroid c_{ij}: $V \approx concat(C_1, C_2, ...C_m)$. In each sub-vector matrix C_i, there are, at most, b different rows, as each row corresponds to one of the centroids of the clusters, so instead of storing full matrix C_i, we can store these unique rows in the separate tensor $\hat{C}_i$, whose elements $\hat{c}_{ij}, j \in \{1..b\}$ correspond to the unique sub-vectors. To compress the approximate embedding matrix, we need only store the ids of the sub-vectors (sub-ids) for each item. Overall, there are m sub-vector sets C_i, so each vector can be encoded using m integer codes, and each code can have b different values; therefore, overall, this scheme can encode up to b^m different vectors. The

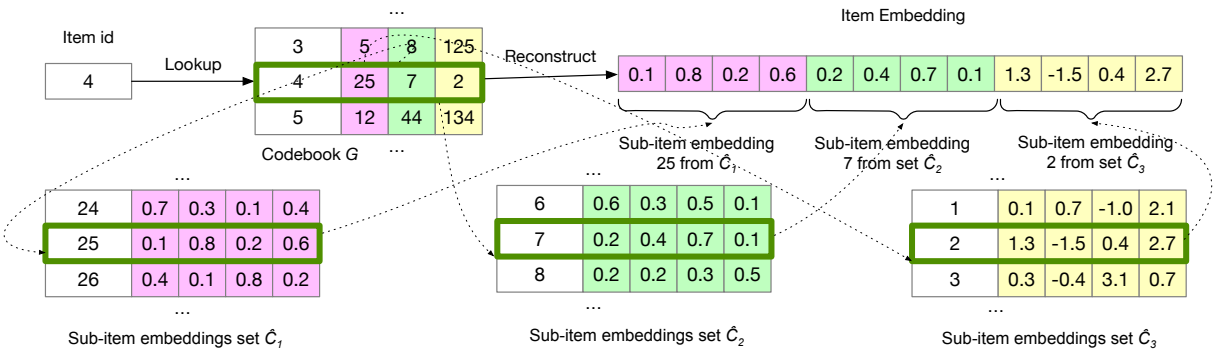


Figure 6.2: Reconstruction of item embeddings using Product Quantisation: Codebook length $m = 3$, item embedding length $d = 12$, number of sub-ids per split $b = 256$.

vector of sub-item ids $g_i = \{g_{i1}, g_{i2}..., g_{im}\}$ associated with the vector $v_i \in V$ is known as the *code* of the vector v_i [93]. The number m of sub-item ids associated with each item is called the *length* of the code. The table G of codes associated with each vector from V is also known as a *codebook* [93]. Figure 6.2 illustrates the vector reconstruction process applied to item embeddings. For each sub-item id g_{ij} in the codes vector g_i of an item i, we extract a sub-item embedding associated with this sub-item id and then concatenate the sub-item embeddings to obtain reconstructed item embeddings.

The number of splits m is usually considerably smaller than the original vector dimensions d: $m \ll d$ to achieve compression. The number of sub-embeddings per split, b, is typically a power k of 256, so that the codes can be stored as k-byte integers. In this chapter, for simplicity and following JPQ [273], we fix $k = 1$, so each sub-item id can be represented with a single byte; therefore, we only store m bytes for each item. Even with fixed k (and therefore b), we can adjust model capacity by controlling the number of sub-item ids associated with each item, m, and the dimension of the sub-item embeddings, $\frac{d}{m}$, by adjusting d. Figure 6.2 illustrates PQ applied to item embeddings for $m = 3$, $d = 12$, $b = 256$. Further, to illustrate the achieved compression, if embeddings are stored as 256-dimensional float32 vectors, a full-item embedding requires 1KB of memory. After compression using product quantisation with $m = 8$ splits, we only need to store 8 bytes per item (0.78% of the original memory requirement). While some memory is also required to store the sub-item embeddings themselves, this is negligible for large datasets compared to the original vector requirements (see also Table 6.2 for an analysis of sub-item embeddings memory requirements).

Product Quantisation is a well-established vectors compression technique, which has been shown to be successful in approximate nearest neighbour methods [60, 93, 97, 129], information retrieval [90, 105, 209] and recommender systems [11, 99, 249]. However, Product Quantisation requires the *full* embeddings tensor to be trained before the embeddings are compressed. Indeed, the quantisation operation is not differentiable, and therefore, the model can not be trained end-to-end. Therefore, it requires first training full (non-quantised) model and only then applying compression. Some recommendation models (e.g. [99]) use differentiable variations of Product Quantisation to allow end-to-end training, but these methods still require training full embeddings alongside the quantised versions.

Overall, Product Quantisation addresses Limitations L6.1 (it is model agnostic), L6.5 (it is applicable for training item embeddings compression), and L6.4 (when sub-item ids are assigned using clusterisation, similar items will have similar codes). However, it does not address Limitations L6.2 (it requires training full embeddings first), and L6.3 (PQ uses a separate loss for embeddings reconstruction, which is not aligned with ranking loss). We now discuss JPQ, a method that can be adapted to address these remaining limitations.

6.3.2 Joint Product Quantisation

Zhan et al. recently proposed *Joint Product Quantisation (JPQ)* [273], a Product Quantisation-based method developed for dense information retrieval. The main difference with the classic product quantisation is that JPQ generates item codes *before* training the model. The code assignment is the only non-differentiable operation in Product Quantisation. Therefore, when the codes are assigned before training the model, the model can be trained end-to-end without training full item embeddings – JPQ essentially replaces the embeddings tensor in the model, where item embeddings are constructed via sub-item embeddings concatenation, as illustrated in Figure 6.2. Assuming that the codebook in the figure is a constant, all other parameters can be learned using standard gradient descent on the model's loss. In particular, this means that JPQ does not require special loss components to learn sub-item embeddings. Compared to the original Product Quantisation method, JPQ addresses Limitations L6.2 (it does not require training full item embeddings) and L6.3 (it does not require a specific loss function). However, in contrast to plain Product Quantisation, JPQ does not provide a mechanism to assign similar embeddings to similar items and, therefore, does not address Limitation L6.4. The sub-item id assignments method proposed in the original JPQ paper is specific for text retrieval (as it relies on the existence of a pre-built index for a text document collection generated, for example, using the STAR model [274]). In the next section, we introduce RecJPQ, an adaptation of JPQ to the Sequen-

tial Recommendation scenario, which does not rely on text-specific datasets and models. Our adaptation of JPQ to Sequential Recommendation requires careful design of novel sub-item id assignment strategies. Indeed, to the best of our knowledge, this is the first adaptation of the JPQ method to Sequential Recommendation.

6.4 RecJPQ

RecJPQ is a Joint Product Quantisation-based method for training recommendation models with a large catalogue of items. As we discuss in Section 6.3.2, the method used by JPQ for initial sub-item id assignments relies on the existence of a pre-built index of documents and, therefore, can not be directly used for recommendation scenarios. Hence, the main difference between RecJPQ and the original JPQ is sub-item id assignment strategies. In general, RecJPQ can be described as performing the following steps:

Step 1. Build the item-code mapping matrix (codebook) using one of the sub-item id assignment strategies (see Section 6.4.1).

Step 2. Initialise the sub-item embeddings randomly.

Step 3. Replace the item embedding tensor with the concatenation of sub-item embeddings associated with each item).

Step 4. Train the model end-to-end using the model's original training task and loss function (this process also trains the sub-item embeddings, so they do not need to be trained separately).

The way RecJPQ builds the codebook before training the main model is also similar to how language models (e.g. BERT [46]) train a tokenisation algorithm before model training. Language models also learn embeddings of sub-word tokens instead of embeddings of full words, to reduce the model's size; similarly, RecJPQ learns sub-item embeddings instead of learning item-level embeddings. Below, we describe sub-item id assignment strategies used by RecJPQ.

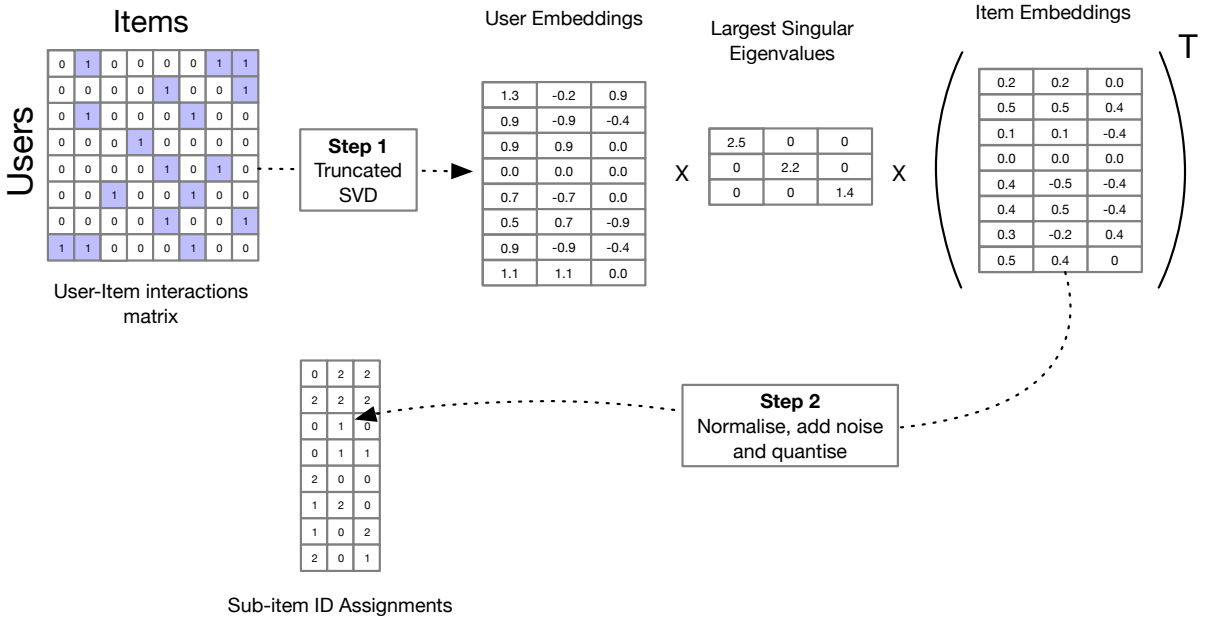


Figure 6.3: RecJPQ sub-item id assignment using Discrete Truncated SVD

6.4.1 Sub-Item Id Assignment Strategies

Random Sub-Item Id Assignments. In the most simple scenario, we can assign items to sub-item ids randomly. We compose the item code out of m random integers in this case. RecJPQ with random sub-item id assignments strategy does not address Limitation L6.4 (similar items do not have similar codes). Indeed, with random sub-item id assignments, RecJPQ becomes similar to other "random" embeddings compression methods, such as the hashing trick [253] or the Quotient Reminder method [219]. The main problem with these methods is that unrelated methods are forced to share parts of their representation, which limits the generalisability of the models. However, as we show in Section 6.5, sometimes random assignments may be beneficial, as the random assignments strategy acts as a form of regularisation. Nevertheless, as the random sub-item id assignments strategy does not address Limitation L6.4 (similar items should have similar representations), we introduce further sub-item id assignment strategies that can address this limitation in the next sections.

Discrete Truncated SVD. As discussed in Section 6.3.2, the only limitation not addressed by JPQ is Limitation L6.4 (similar items should have similar codes). Random sub-item id assignments, as discussed above, do not address this limitation either. Hence, in this section, we design a sub-item id assignments method that addresses this remaining limitation and assigns similar codes to similar items. Some approaches have used side information, such as textual data for item representations [194]; however, we address the more generic classic Sequential Recommenda-

tion scenario and, therefore, must infer item similarities from the user's sequences. To do this, we employ the PureSVD [37] algorithm, which has been shown to achieve good results in learning item representations [286] for recommender systems. Figure 6.3 illustrates the SVD-based code assignment strategy. We now describe this strategy in detail.

We first compute a matrix of sequence-item interactions M, where rows correspond to sequences (users), and columns correspond to items. This matrix's elements m_{ij} are either 1 if i^{th} contains interactions with item j and 0 otherwise. We then compute the truncated SVD decomposition of matrix M with m latent components: $M \approx U \times \Sigma \times V^T$, where U is the matrix of user embeddings, V is the matrix of item embeddings, and Σ is the diagonal matrix of largest singular values.

To ensure that items that have been interacted with by exactly the same set of users obtained different embeddings, we normalise V using the min-max normalisation range and add a small amount of Gaussian noise:

$$\hat{v}_{ab} = \frac{v_{ab} - \min_k v_{ak}}{\max_k v_{ak} - \min_k v_{ak}} + \mathcal{N}(0, 10^{-5}); \forall v_{ab} \in V$$

The variance of the noise (10^{-5}) is negligible compared to the range of possible values of normalised embeddings ($[0..1]$ after min-max normalisation). Therefore it has a very small influence on the position of the items in the embeddings space. However, if two items have exactly the same embeddings after decomposition (e.g. this can happen if two items appear in exactly the same set of sequences), the noise allows us to distinguish these two embeddings. Lastly, the assignment of sub-item id involves discretising each dimension of the normalised item embeddings into b quantiles so that each quantile contains an approximately equal number of items. We use these bins as sub-item id assignments for the items. Note that although this method requires computing an m-dimensional Item Embeddings Tensor (and there can be hundreds of millions of items), it does not require computing them as part of a deep learning model training on a GPU. Indeed, truncated SVD is a well-studied problem. There are effective algorithms for performing it that do not require modern GPUs [69]. Moreover, as the method only requires computing m-dimensionsional embeddings, the table will be many times smaller than full d-dimensional embeddings, so the method requires $\frac{d}{m}$ times less memory to store embeddings. Finally, performing truncated SVD is possible in a distributed manner (e.g., using Apache Spark [47]), which allows truncated SVD to be applied on large datasets. In summary, discrete truncated SVD allows assigning similar codes to similar items; it does not require a GPU for intermediate computations and can be easily performed for very large datasets.

Discrete BPR. Truncated SVD is not the only Matrix Factorisation method that can be used for initial sub-item id assignments. In particular, we also use the classic BPR approach [199] to obtain coarse item embeddings. The method also learns user embeddings (or, in our case, sequence embeddings) U and item embeddings V. The estimate of the relevance of an item i for user j is defined as the dot product of user and item embeddings: $r = u_j \cdot v_i$. In contrast with truncated SVD, BPR does not directly approximate the user-item interaction matrix. Instead, BPR optimises a pairwise loss function that aims to ensure that positive items are scored higher than negative items: $\mathcal{L}_{BPR} = -\log(\sigma(u_i \cdot v_{j^+} - u_i \cdot v_{j^-}))$, where v_{j^+} is the embedding of a positive item for the user u, v_{j^-} is the embedding of a randomly sampled negative item, and σ is the sigmoid function.

BPR is one of the most cited methods in recommender systems; therefore, we use BPR as an alternative strategy for coarse item embedding learning. The rest of the discrete BPR strategy is the same as in the truncated SVD: we also normalise the learned embeddings using a min-max normalisation and add a small amount of noise to ensure different embeddings for different items are obtained. Similar to truncated SVD, BPR does not require learning on a GPU, and there exist distributed implementations [44] that allow for learning item embeddings on very large datasets, so overall, it can be used as an alternative to truncated SVD for sub-item id assignments in RecJPQ.

This concludes the description of RecJPQ's sub-item id assignments. We now argue why RecJPQ can improve effectiveness.

6.4.2 RecJPQ as a Regularisation Mechanism

The interactions with items in recommender systems typically exhibit a long tail distribution [169], meaning that a few popular items receive the most interactions. In contrast, most items comprise the "long tail" with few interactions. As the training data for these long-tail items is limited, recommendation models can suffer from overfitting on such items [278], causing overall performance degradation.

Goodfellow et al. [64, Ch. 7] argued that one of the most powerful regularisation techniques is *parameter sharing* – a technique where certain parameters of the model are forced to be equal. RecJPQ is a special case of parameter sharing: we force different items to share parts of their embeddings. This prevents the model from learning embeddings that are too specific to only a

few training sequences, as each part of the embedding appears in many other sequences via the sharing mechanism. In our experiments (Section 6.5), we indeed observe that RecJPQ may act as a model regulariser and improve the model's performance; this is especially apparent in the Gowalla dataset, where the proportion of long-tail items is the largest[3].

RecJPQ Summary. RecJPQ is a model component that takes the place of the item embeddings in Sequential Recommender Systems. RecJPQ is based on the JPQ method, which is a variation of the PQ method. RecJPQ addresses all of the limitations described in Section 6.2: it is model-agnostic (Limitation L6.1); does not require training full embeddings (Limitation L6.2); does not modify the backbone model's loss function (Limitation L6.3); it is suitable for item embeddings compression (Limitation L6.5); it can assign similar codes to similar items with the help of discrete truncated SVD or discrete BPR (Limitation L6.4). Furthermore, we argue that RecJPQ may act as a model regulariser, which is an additional advantage when there are many long-tail items in the catalogue. This concludes the description of RecJPQ. We now turn to the experimental evaluation of RecJPQ.

6.5 Experimental evaluation of RecJPQ

Our experiments address the following research questions:

RecJPQ Research Questions

RQ6.1: What is the effect of the sub-item id assignment strategy?

RQ6.2: How do code length and embedding size impact effectiveness?

RQ6.3: What is the effect of RecJPQ on size/effectiveness tradeoff?

3. We also note that our ALBERT4Rec model, described in Section 3.5.3, also employs parameter sharing. However, unlike RecJPQ, it shares parameters between transformer layers rather than between item embeddings. The high effectiveness of ALBERT4Rec further confirms the usefulness of parameter sharing as a regularisation mechanism.

6.5.1 Experimental Setup

Backbone Models. In our experiments, we use two state-of-the-art Transformer-based recommendation models: BERT4Rec and SASRec. For both models, we use the versions[4] from our recent reproducibility study(Chapter 3), which provides efficient & effective implementations (using the Huggingface Transformers library [255]). Additionally, to demonstrate that RecJPQ can be applied to other architectures, we use a GRU-based model. This model uses the GRU4Rec [80] architecture, but a slightly different configuration, e.g. it uses LambdaRank [19] as a loss function, which we have shown to be effective (see Section 4.5).

Datasets & Metrics We experiment with: (i) MovieLens-1M [71] – this is a movie rating dataset that is one of the most popular benchmarks for recommender systems; (ii) Booking.com [63] – a multi-destination trips dataset, and (iii) Gowalla [32] – a check-in dataset (see Table 2.1 for the salient characteristics of the datasets). The number of items in these datasets varies from relatively small (3.4K in MovieLens-1M) to large (1.3M in Gowalla) – this allows testing RecJPQ in different settings (RecJPQ is designed for large datasets, and we expect it to compress the model by a larger factor on Gowalla). The datasets are also diverse regarding the number of "long-tail items" (defined as items with less than five interactions), ranging from no long-tail items at all in MovieLens to 75.8% in Gowalla. As discussed in Section 6.4.2, RecJPQ acts as a model regulariser in long-tail distributions, and we expect to see the highest regularisation effect on the Gowalla dataset.

Overall, experimental evaluation follows Section 2.4. We set the maximum sequence length at 200. For measuring effectiveness, we use NDCG@10, and as the model size metric, we use the file size of the model checkpoint.

Baselines. We deploy an adaptation of Quotient Remainder [219] (denoted Q-R) as a baseline compression approach, applied to each base model – this parameter-free hashing-based approach encodes each item using two hashes: the quotient and the remainder of the division of item id by $\left\lceil \sqrt{|I|} \right\rceil$ where $|I|$ is the catalogue size. Q-R guarantees that each item has a unique code.

4. The code for the paper is available at https://github.com/asash/RecJPQ

Table 6.3: Impact of RecJPQ with different sub-item id assignment strategies on model size and effectiveness for the MovieLens-1M and Booking.com datasets. Relative Size corresponds to model checkpoint size as the percentage of the base model. $^{=}$, $^{+}$, and $^{-}$ denote significance testing results compared to the base, respectively: indistinguishable ($pvalue > 0.05$, Bonferroni multi-test correction), better or worse. Bold denotes the best NDCG@10 in each column.

Dataset	Model→ Strategy↓	BERT4Rec		GRU		SASRec	
		NDCG@10	Rel. Size	NDCG@10	Rel. Size	NDCG@10	Rel. Size
ML-1M	Base	**0.157**	100.0%	0.072	100.0%	**0.131**	100.0%
	Hashing (Q-R)	0.040^{-}	92.4%	0.017^{-}	61.6%	0.009^{-}	124.9%
	RecJPQ-BPR	$0.156^{=}$	93.2%	$\mathbf{0.076}^{=}$	62.5%	$0.130^{=}$	128.0%
	RecJPQ-Random	$0.156^{=}$	93.2%	$0.075^{=}$	62.5%	$0.125^{=}$	127.6%
	RecJPQ-SVD	$0.154^{=}$	93.2%	$0.074^{=}$	62.5%	$0.129^{=}$	127.9%
Booking	Base	0.376	100.0%	0.209	100.0%	0.137	100.0%
	Hashing (Q-R)	0.192^{-}	62.8%	0.186^{-}	27.6%	0.014^{-}	9.2%
	RecJPQ-BPR	$0.375^{=}$	63.3%	$\mathbf{0.334}^{+}$	27.5%	0.242^{+}	8.7%
	RecJPQ-Random	0.316^{-}	62.3%	0.324^{+}	27.5%	$\mathbf{0.256}^{+}$	8.9%
	RecJPQ-SVD	$\mathbf{0.379}^{+}$	63.3%	$\mathbf{0.334}^{+}$	27.6%	0.185^{+}	8.8%

We do not apply post-training embedding quantisation (e.g. float16), nor use other methods from Table 6.1 as baselines, as they are not suitable for our task: EODRec, LightRec, MDQE and MGQE require training full embeddings (we assume that training full embeddings is not an option for a large catalogue), and PreHash is specific for compressing user embeddings, so is not suitable for item embeddings. However, reducing the model size by decreasing the embedding dimensionality can also be seen as a simple baseline. We analyse models using different embedding sizes in RQ6.3.

To analyse the effect of the sub-item id assignment strategy on model performance/model size tradeoff, we compare the original (base) versions of BERT4Rec, SASRec and GRU with RecJPQ versions trained with Random, discrete truncated SVD and discrete BPR sub-item id assignment strategies. We do not train GRU and BERT4Rec on Gowalla, as these models do not use negative sampling. Training models on this dataset without negative sampling is not feasible due to the large GPU memory requirement for storing output scores, while applying negative sampling is a substantial change to the models' training process that is outside of the scope of this chapter (note that in Section 5.6.2.4 we showed that it is possible to train BERT4Rec on large catalogues using gBCE loss; we further show that this result generalises to RecJPQ-enhanced version of BERT4Rec in Section 6.6). In all cases, we use 512-dimensional embeddings and the code of length $m = 8$ (we experiment with other embedding sizes and lengths of the code in the next section). One exception is for the SASRec base model on Gowalla; in this case, we use 128-dimensional item embeddings (item embeddings larger than 128 dimensions consume all available GPU memory when embedding compression techniques are not deployed).

Table 6.4: Impact of RecJPQ sub-id assignment strategies on SASRec model size and effectiveness on the large-scale Gowalla dataset. Notations follow Table 6.3.

	Base	Hashing (Q-R)	RecJPQ-BPR	RecJPQ-Random	RecJPQ-SVD
NDCG@10	0.110	0.081^{-}	0.033^{-}	$\mathbf{0.173}^{+}$	0.122^{+}
Relative Size	100.0%	2.8%	2.8%	2.9%	2.9%

6.5.2 Results

RQ6.1. Effect of sub-item id assignment strategy. Table 6.3 shows the experimental results on the smaller ML-1M and Booking datasets, while Table 6.4 reports results for the Gowalla dataset. The tables compare NDCG@10 and model size of compressed variations of backbone models with the base (uncompressed) model. Significant differences compared to the corresponding base model (BERT4Rec, GRU or SASRec) are indicated. In general, the tables show that RecJPQ substantially reduces the model checkpoint size in most cases. For example, the RecJPQ versions of the GRU models on the Booking dataset are approximately 27% of the original in size. On the Gowalla dataset, compressed models are approximately 3% of the original. We also see that model size does not depend on the sub-item id assignment strategy. Indeed, sub-item id assignments only influence the values of the model parameters but not the number of parameters. Moreover, Q-R models have approximately the same compression level as RecJPQ models. We speculate that after compression, the model checkpoint size is dominated by other model parameters (e.g., attention matrices). In our configuration, the Sub-Item Embeddings Tensor only requires a few megabytes of memory (see Table 6.2). In contrast, the full model checkpoint of a compressed model is typically tens of megabytes (e.g., 92.8MB for SASRec using RecJPQ-BPR trained on Gowalla).

RecJPQ only increases the model size for SASRec on MovieLens-1M, due to the dataset's small item count: on this dataset, the overhead of storing sub-item embeddings and the codebook is larger compared to the benefit of compressing the embeddings table. Using RecJPQ with smaller embeddings might reduce the model size without affecting effectiveness on this dataset (see also RQ6.3).

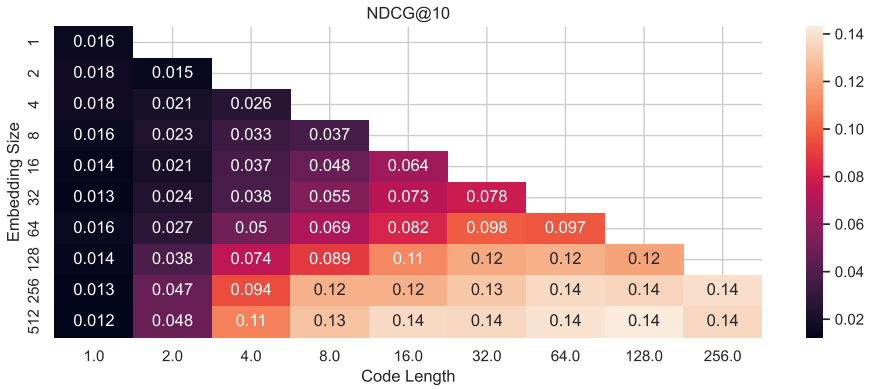
On the other hand, we observe from Table 6.3 and Table 6.4 that the choice of the best assignment strategy depends on both the model and the dataset. For example, on MovieLens-1M, the choice of the strategy is not important, and in all cases, RecJPQ versions of the models are statistically indistinguishable from all corresponding base models. On the larger Booking dataset, the choice of the best strategy is model-dependent. For BERT4Rec, the best results are achieved with BPR

(NDCG@10 0.375, statistically indistinguishable from the base) and SVD (NDCG@10 0.379, +0.97%, significant). At the same time, the Random strategy significantly underperforms the base configuration (NDCG@10 0.316, -15.98%) – this shows that in some cases, assigning similar codes to similar items is indeed important. However, in 2 cases, Random performs statistically significantly better than SVD and BPR. For example, Random assignments perform best on Gowalla with the SASRec base (a significant improvement of +57% over the base). SVD assignments also moderately improve the result in this case (+10%, significant). At the same time, BPR decreases the quality by a large margin on Gowalla dataset (-70%)[5]. We explain the success of the Random strategy on the Gowalla dataset as giving a larger regularisation effect (random assignments make the learning task harder, so the model has fewer chances to overfit). Overall, from Tables 4 and 5, we conclude that there is no "one size fits all" choice of the sub-item id assignment strategy, and it depends on both dataset and model salient characteristics; the exact best combination of model/strategy may depend on regularisation requirements (as we observe in Gowalla), the prevalence of strong sequential patterns (as in Booking, see Section 4.5 for details) and so on. This suggests that the sub-item id assignment strategy could be treated as a hyperparameter and tuned for each model/dataset combination. However, by default, we recommend using RecJPQ with the SVD strategy – in all cases, this achieves significantly better (on the Booking and Gowalla datasets) or statistically indistinguishable (on the MovieLens-1M dataset) results compared to the base model. We also note that RecJPQ with the SVD strategy is always better than the Q-R hashing baseline (Q-R is always significantly worse than the base model, whereas RecJPQ is better or indistinguishable). The question of whether or not it is possible to select the best model/strategy combination without doing an exhaustive hyperparameter search is an interesting research direction, which we leave for future work.

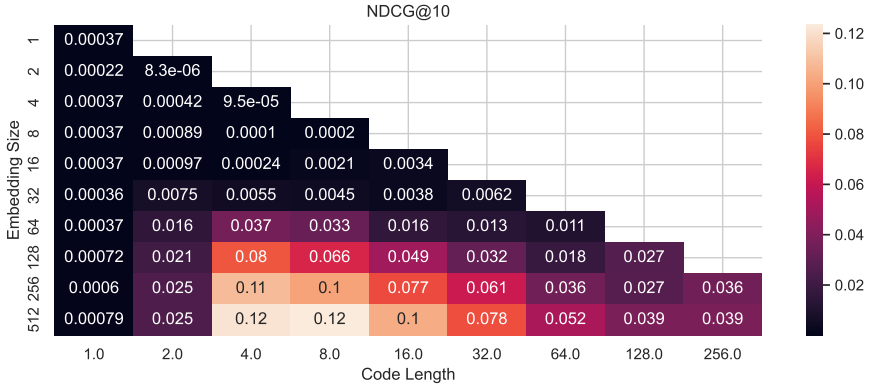
We note that there is no degradation of the training efficiency when training the RecJPQ versions of the models. Indeed, while there are some fluctuations in the training time the model requires to converge, the magnitude of the required time remains the same: training of the base version of BERT4Rec requires 18.8 hours on Booking.com, and the RecJPQ-SVD version of BERT4Rec requires 16.1 hours. The training time of the SVD model (used for initial sub-item id assignments) in the same case is negligible compared to the training of the main model ($\sim$1 minute).

---

5. The percentages for the Gowalla dataset seem large because this dataset is difficult: it has the largest number of items and the largest proportion of long-tail items.



(a) MovieLens-1M



(b) Gowalla

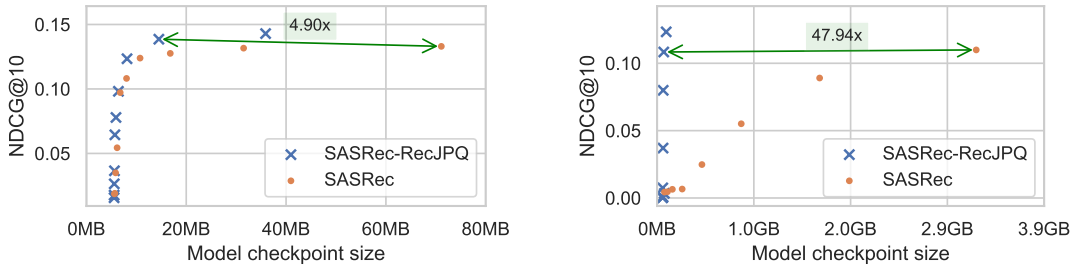
Figure 6.4: RecJPQ performance for SASRec while varying embedding size d and the number of sub-item ids per item m.

In summary, in answer to RQ6.1, we conclude that RecJPQ achieves large model compression levels. The compression is particularly impressive on datasets with large catalogues (like Gowalla). The compression does not depend on the sub-item id assignment strategy. However, the sub-item id assignment strategy greatly affects the model performance. The effect is model- and dataset-dependent, so the strategy should be treated as a hyperparameter. However, the SVD is a safe choice, as it always provides comparable results (i.e. statistically indistinguishable) or better than the base model.

RQ6.2. Effects of code length m and the embedding size on model performance. To answer RQ6.2, we perform a grid search over embedding size and code length on the MovieLens-1M and Gowalla datasets. We use SASRec as the backbone (the only model that can be easily trained on Gowalla) and apply the SVD sub-item id assignment strategy. We select the embedding size d from $\{2^0, 2^1, ..., 2^9\}$ and code length m from $\{2^0, 2^1, ..., 2^8\}$. Note that $m \leq d$, as RecJPQ splits each embedding of size d into m sub-embeddings.

Figure 6.4 illustrates the results of the grid search. The figure shows the NDCG@10 of the SASRec-RecJPQ model for each combination of code length (x-axis) and embedding size (y-axis), in the form of a heatmap for both datasets. As we can see from the figure, a larger embedding size generally positively affects the model performance. This result echoes similar findings of a recent reproducibility paper [201]; however, interestingly, in the RecJPQ case, increasing embedding dimensionality does not change the amount of information we store per each item, as the length of the code defines it rather than the embedding size. Instead, it increases model capacity, increasing the amount of information that can be stored in each sub-item embedding, allowing to account for more item characteristics. For example, on Gowalla, the largest embedding we can train using base SASRec is 128 dimensions, while with RecJPQ, we can train the model even with 512-dimensional embeddings.

On the other hand, larger code lengths are not always helpful. As we discussed in Section 6.4.2, RecJPQ forces the model to share parts of embeddings with other items, acting as a regularisation mechanism. A shorter code length forces items to share more information, causing stronger regularisation. As we can see, on the less sparse MovieLens-1M – where all items have more than five interactions – regularisation is not an issue, and longer codes are beneficial. For example, the best result is achieved with 512-dimensional embeddings and a code of length 128



(a) MovieLens-1M **(b)** Gowalla

Figure 6.5: NDCG/Size tradeoff for SASRec and SASRec-RecJPQ.

(NDCG@10 0.14). In contrast, for Gowalla, where most items are long-tail items with less than five interactions (hence, the embeddings of these items should be regularised), the best NDCG is achieved with the code of length 8 (NDCG@10 0.12). The fact that the model can perform better with shorter codes confirms that RecJPQ can behave as a regularisation technique.

In short, in answer to RQ6.2, we conclude that larger embeddings are generally beneficial for model performance. However, the sparser Gowalla dataset benefits from shorter code lengths, due to the regularisation effect of parameter sharing brought by RecJPQ.

RQ6.3. Size-Performance tradeoff. To address our last research question, we analyse the trade-off between model checkpoint size and NDCG@10 achieved by the model when trained with different embedding sizes. We select the embedding size from $\{2^0, 2^1, ..., 2^9\}$ and train the original versions of SASRec and SASRec-RecJPQ with the SVD strategy on MovieLens-1M and Gowalla. For RecJPQ, we select optimal code length m for the dataset/embedding size pair (according to the grid search from Fig. 6.4). For SASRec on Gowalla, we train up to the embedding size of 128 due to GPU memory limits.

Figure 6.5 illustrates the tradeoff between model checkpoint size and NDCG@10 for both SASRec and SASRec-RecJPQ on the two datasets. Each point on the figure corresponds to one embedding size. As can be seen from the table, a larger model size (corresponding to larger embeddings) leads to better performance for both SASRec and SASRec-RecJPQ (this echoes findings in the previous research question). However, SASRec-RecJPQ's performance grows much faster with increasing model size than observed for vanilla SASRec. For example, the largest vanilla SASRec model achieves roughly the same performance as the 4.9× smaller SASRec-RecJPQ version of the model (71MB vs. 15 MB). This effect is even more prominent in Gowalla, where the number of items is larger: the largest SASRec model achieves roughly the same performance as the 47.94× smaller SASRec-RecJPQ model (3.2GB vs. 69MB).

Overall, in answer to RQ6.3, we conclude that while larger models benefit model performance, RecJPQ improves this tradeoff by a large margin (i.e. to achieve the same performance, RecJPQ requires much fewer parameters than the original model). This effect is more markedly pronounced for the larger Gowalla dataset.

6.6 Discussion

In Chapters 4-6, we introduced a number of enhancements that can be applied to Transformer-based Sequential Recommender Systems. In particular, in Chapter 4, we introduced RSS, the training objective that balances ranking effectiveness and training efficiency; in Chapter 5, we introduced the gBCE loss function that counters undesirable effects of negative sampling when training recommendation models with large catalogues, as well as gSASRec and gBERT4Rec models that use gBCE; finally in this chapter we introduce RecJPQ, a technique that allows compressing the Item Embeddings tensor. We now show that these enhancements are orthogonal to each other and can be combined when necessary.

Indeed, to demonstrate that RecJPQ, introduced in this chapter, can be combined with other enhancements introduced earlier in this thesis, we use a large-scale Gowalla[6] dataset to train several models that combine the techniques and analyse their effectiveness and efficiency. In particular, we train the following models:

1. SASRec + RecJPQ; the vanilla SASRec model, enhanced with RecJPQ;
2. gSASRec + RecJPQ; the gSASRec model, enhanced with RecJPQ;
3. gBERT4Rec + RecJPQ; the gBERT4Rec mod, enhanced with RecJPQ;
4. gSASRec + RecJPQ + RSS; the gSASRec model, enhanced with RecJPQ and trained with the RSS training objective.

Table 6.5 presents ranking effectiveness metrics (NDCG@10, Recall@10) and training time for these models. As we can see from the table, models employing the gBCE objective (gSASRec, gBERT4Rec) outperform standard BCE-based SASRec in ranking effectiveness by a large margin. Specifically, NDCG@10 ranges from 0.1667 to 0.1718 for gBCE-based models compared to 0.1143 for SASRec, confirming that combining gBCE with RecJPQ, which compresses the Item Embeddings Tensor, effectively reduces model overconfidence and enhances performance.

6. Note that training regular BERT4Rec on this dataset is not feasible, while training regular SASRec results in suboptimal efficiency due to overconfidence, see Section RQ5.5.

Table 6.5: Comparison of Recommendation Models on Gowalla dataset. **Bold**: Best value, *: Statistically significant difference with the gBERT4Rec+RecJPQ model ($pvalue < 0.05$, Bonferroni multi-test correction))

Model	Recall@10	NDCG@10	Convergence Time (seconds)
SASRec + RecJPQ	0.1402*	0.1143*	**5007**
gSASRec + RecJPQ	0.2148*	0.1667	6000
gSASRec + RecJPQ + RSS	0.2123*	0.1632*	6333
gBERT4Rec + RecJPQ	**0.2374**	**0.1718**	30365

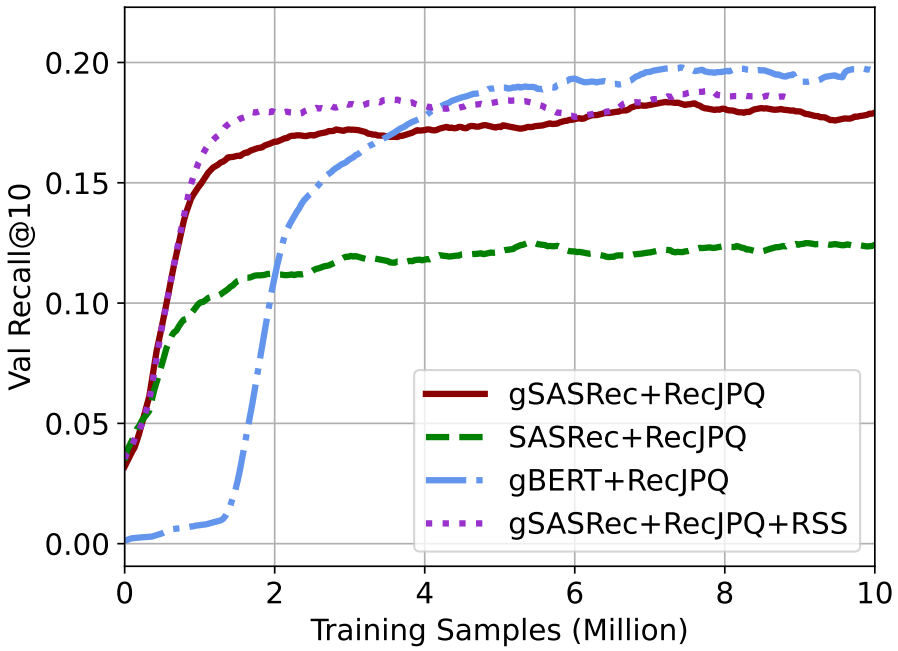


Figure 6.6: Validation Recall@10 learning curves of RecJPQ-enhanced model variants plotted against the number of training samples (in millions).

Additionally, we also observe from the table that the gBERT4Rec-based model, utilising the Item Masking objective, takes substantially longer to converge (30365 seconds) compared to SASRec-based models employing Sequence Shifting (SASRec+RecJPQ, gSASRec+RecJPQ) or RSS objectives (gSASRec + RecJPQ + RSS) (e.g., 6333 seconds for gSASRec + RecJPQ + RSS). This aligns with our earlier analysis (Chapter 4), where we demonstrated that Sequence Shifting and RSS objectives achieve better training efficiency because they more closely align with the Next Item Prediction task, while Item Masking is only weakly related to Next Item Prediction.

To further analyse training efficiency, we examine the Validation Recall@10 measured throughout model training for these figures (Figure 6.6). From the figure, we see that gSASRec+RecJPQ+RSS converges much faster than gBERT4Rec+RecJPQ: its performance saturates after approximately 2 million training samples, whereas gBERT4Rec+RecJPQ requires around 6 million samples to saturate. Additionally, within fewer than 4 million training samples, gSASRec+RecJPQ+RSS achieves the best validation NDCG@10 across all models. This result confirms the RSS objective's effectiveness in scenarios with limited training, aligning closely with our original motivation for designing RSS in Chapter 4.

In summary, our analysis confirms that RecJPQ can effectively be combined with the gBCE loss (enabling the training of effective models with large catalogues) and with the RSS objective (allowing the training of effective models under the limited training conditions). These findings demonstrate the complementary nature of the methods introduced in Chapters 4–6.

6.7 Conclusions

In this chapter, we discussed the challenge of training Sequential Recommender Systems with large datasets, primarily due to the large item embedding tensor. Existing embedding compression methods have limitations, leading to our proposed method, RecJPQ. Our evaluation of RecJPQ on three datasets showed significant model size reduction, e.g., 47.94× compression of the SASRec model on the Gowalla dataset. Additionally, RecJPQ serves as a model regularisation technique, improving the model's quality, with SASRec-RecJPQ using SVD strategy outperforming the original SASRec model (+35% NDCG@10 on Booking, +10% on Gowalla).

Our original goal when developing RecJPQ was only to compress the item embedding tensor. However, decomposing atomic item ids into sub-ids created a *structure* within the item ids, enabling other applications for RecJPQ beyond just model compression. One of these applications we discuss in the next chapter: we show that by using salient properties of RecJPQ, it is possible to avoid exhaustive item scoring during the model inference and, as a consequence, reduce model response time by a large margin.

Chapter 7

Efficient Inference of RecJPQ-based Recommendation Models

In Chapters 5 and 6, we addressed challenges related to training Sequential Recommendation models with large catalogues. However, as stated by Limitation L2.4, even if we can *train* a sequential recommendation model, its *inference* remains problematic. Indeed, most Transformer-based sequential recommendation models (including both BERT4Rec and SASRec) compute item relevance scores as dot products between the Item Embedding Tensor and the sequence embedding (see also Section 2.2.1 and Figure 2.7). With a large catalogue, this scoring operation becomes expensive and sometimes infeasible due to hardware limitations. Moreover, large catalogues often come with a large user base, further increasing hardware demands. Reducing these requirements – such as enabling inference without GPU acceleration – can make large-scale deployment more practical.

In this chapter, we address Limitation L2.4, building upon salient characteristics of RecJPQ, a technique that splits item ids into sub-item ids, which we introduced in the previous chapter. The chapter is organised as follows: in Section 7.1, we introduce PQTopK, an efficient scoring algorithm based on pre-computing sub-item id scores; in Section 7.2 we demonstrate how dynamic pruning techniques adapted from information retrieval can eliminate the need for exhaustive scoring; Section 7.3 contains concluding remarks.

The material of this chapter is based on two research papers: Section 7.1 is based on our paper [182], which was published as a short paper in the proceedings of the ACM RecSys'24 conference. Section 7.2 is based on a full research paper [172] that has been accepted as a full research paper to the ACM SIGIR'25 conference.

7.1 Efficient Inference by Pre-Computing Sub-Id Scores

7.1.1 Need for Efficient Inference

As we discussed in Section 2.3.5, slow inference with large catalogues is one of the key limitations (Limitation L2.4) of Transformer-based recommendation models, such as SASRec and BERT4Rec. Efficient inference is especially important when considering a model deployment on CPU-only hardware (i.e. without GPU acceleration). Indeed, deploying a trained model on CPU-only hardware is often a practical choice, considering the high running costs associated with GPU accelerators. Hence, in this chapter, we specifically focus on the CPU-only inference efficiency of Transformer-based sequential recommendation models.

The inference of a Transformer-based recommendation model consists of two parts: computing a *sequence representation* using the *backbone* Transformer model, followed by computing the scores of individual items using this representation (see Section 2.3 for details). The main cause of the slow inference by the Transformer-based models arises not from the Transformer backbone model itself but from the computation of all the item scores. Indeed, the inference time of a given Transformer backbone model is constant w.r.t. the number of items (after embedding lookup, which is $\mathcal{O}(1)$ operation, the Transformer model only works with embeddings, which do not depend on the number of items); however, computing item scores has a linear complexity w.r.t. the number of items. Hence, to speed up inference, there are three options: (i) reduce the number of scored items, (ii) reduce the number of operations per item, and (iii) efficiently parallelise computations. In the first category are the approximate nearest neighbour methods, such as FAISS [97] or Annoy [225]. While these methods can be practical in some cases (for example, when using multistage-architectures [36]), there are two problems: (i) these methods are *unsafe* [240, 246], meaning that the results retrieved using an ANN index may omit some candidates that would have been scored high by the model and (ii) they require item embeddings to be present in the first place in order to build the index, and training item embeddings for all items in large catalogue case may not be feasible in the first place [180]. Therefore, this section focuses on analysing the efficiency of existing methods and reducing the number of operations per item and parallelising the computations (we will discuss how to safely reduce the number of scored items in Section 7.2). In particular, we build upon RecJPQ, an approach for compressing embedding tables in Transformer-based sequential recommenders, that we described in Chapter 6. RecJPQ achieves compression by representing items using a concatenation of shared sub-item ids. Prior works that built upon similar ideas of sub-id-based recommendation, such as LightRec [133], showed that the sub-id-based method could indeed improve model inference

time. Inspired by LightRec[1], we describe a sub-id-based scoring algorithm for RecJPQ-based models, which we call *PQTopK*. We further analyse if RecJPQ-enhanced Transformer-based recommendation models can be efficiently inferred on catalogues with (multiple) millions of items using the PQTopK algorithm in a CPU-only environment.

The main contributions of this section can be summarised as follows:

1. We analyse inference efficiency of RecJPQ-enhanced versions of SASRec and BERT4Rec and find that it is more efficient than Matrix-Multiplication based scoring used in the original models;
2. We show that scoring efficiency of RecJPQ-based models can be improved using the PQTopK algorithm;
3. We explore the limits of PQTopK-based inference using simulated settings with up to 1 billion items in catalogue and show that inference remains efficient with millions of items

To the best of our knowledge, this is the first analysis of the inference of sub-id-based sequential models on large-scale datasets and the first demonstration of the feasibility of using these models in the large-catalogue scenario.

7.1.2 Item scoring with RecJPQ

Typically, to generate recommendations given a history of interactions $h = \{i_1, i_2, i_3 ... i_n\}$, Transformer-based models first generate a sequence embedding $\phi \in \mathbb{R}^d$. The scores for all items, $r = (r_1, \ldots, r_{|I|}) \in \mathbb{R}^{|I|}$, are then computed by multiplying the Item Embedding Tensor $V \in \mathbb{R}^{|I| \times d}$, which is usually shared with the embeddings layer of the Transformer model, by the sequence embedding ϕ, i.e.,

$$r = V\phi \tag{7.1}$$

This default transformer scoring is also illustrated by Figure 2.7 in Section 2.2.1. However, in this section, we show that for RecJPQ-based models, inference can be more efficient than Matrix-Factorisation-based scoring using Equation (7.1).

---

1. Note that LightRec is not a sequential model, and hence can not be directly used as a replacement for RecJPQ.

Recall that to compress the item embedding matrix V, RecJPQ associates each item id to a list of *sub-ids*, akin to language models breaking down words into sub-word tokens. RecJPQ reconstructs an item's embedding by combining the sub-id embeddings assigned to it. More formally, RecJPQ first builds a *codebook* $G \in \mathbb{N}^{|I| \times m}$ that maps an item id i to its associated m sub-ids:

$$G[i] \rightarrow \{g_{i1}, g_{i2}, ..., g_{im}\} \tag{7.2}$$

where g_{ij} is the j-th sub-id associated with item i and m is the number of splits. As we discussed in Section 6.4.1, there are several strategies for associating ids to sub-ids, however as we have shown in Section 6.5.2, Truncated SVD-based assignment is always a safe choice; hence we assume this SVD-based sub-id assignment strategy throughout this chapter.

For each split $k = 1, \ldots, m$, RecJPQ stores a sub-item embedding matrix $\hat{C}_k \in \mathbb{R}^{b \times \frac{d}{m}}$, where b is the number of distinct sub-ids in each split. The j-th row of $\hat{C}_k$ denotes the sub-item embedding $\hat{c}_{k,j} \in \mathbb{R}^{\frac{d}{m}}$ associated with the j-th sub-id, in the k-th split. Then, RecJPQ reconstructs the item embedding v_i as a concatenation of the associated sub-id embedding:

$$v_i = \hat{c}_{1,g_{i1}} \parallel \hat{c}_{2,g_{i2}} \parallel ... \parallel \hat{c}_{m,g_{im}} \tag{7.3}$$

Finally, an item score can be computed as the dot product of the sequence embedding and the constructed item embedding:

$$r_i = v_i \cdot \phi \tag{7.4}$$

A straightforward use of Equation (7.4) for item scoring in RecJPQ-based recommendation models does not lead to any computational efficiency improvements compared to models where all item embeddings are stored explicitly: in both cases, the algorithm would need to multiply the sequence embedding w by the (reconstructed) embeddings of all items. However, the sub-id representations of RecJPQ allow a more efficient scoring algorithm, which we describe next.

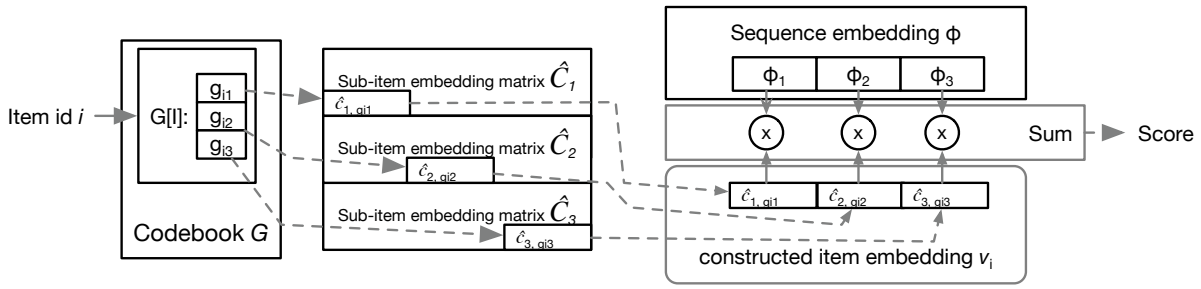


Figure 7.1: Computing an item score using the PQTopK algorithm in a RecJPQ-based model with $m = 3$ splits.

7.1.3 PQTopK Algorithm

PQTopK is a scoring algorithm for RecJPQ-based models that uses pre-computation of sub-id scores for improved inference efficiency. While similar algorithms have previously been described, for example, for a different recommendation scenario [133] and for document retrieval [273], to the best of our knowledge, it has not been previously applied for sequential recommendation nor Transformer-based models.

PQTopK first splits the sequence embedding $\phi \in \mathbb{R}^d$ obtained from a Transformer model into m sub-embeddings $\{\phi_1, \phi_2 ... \phi_m\}$, with $\phi_k \in \mathbb{R}^{\frac{d}{m}}$ for $k = 1, \ldots, m$, such that $\phi = \phi_1 \,\|\, \phi_2 \,\|\, ... \,\|\, \phi_m$. By substituting Equation (7.3) and the similarly decomposed sequence embedding ϕ into Equation (7.4), the final item score for item i is obtained as the sum of sub-embedding dot-products:

$$r_i = v_i \cdot \phi = (\hat{c}_{1,g_{i1}} \,\|\, ... \,\|\, \hat{c}_{m,g_{im}}) \cdot (\phi_1 \,\|\, ... \,\|\, \phi_m) = \sum_{k=1}^{m} \hat{c}_{k,g_{ik}} \cdot \phi_k$$

Let $S \in \mathbb{R}^{m \times b}$ denote the *sub-id score matrix*, which consists of *sub-id scores* $s_{k,j}$, defined as dot products of the sub-item embedding $\hat{c}_{k,j}$ and the sequence sub-embeddings ϕ_k:

$$s_{k,j} = \hat{c}_{k,j} \cdot \phi_k \tag{7.5}$$

The final score of item i can, therefore, also be computed as the sum of the scores of its associated sub-ids:

$$r_i = \sum_{k=1}^{m} s_{k,g_{ik}} \tag{7.6}$$

Figure 7.1 also graphically illustrates how item scores are computed using PQTopK by combining the pre-computed sub-id scores using RecJPQ's codebook.

Algorithm 2 PQTopK(G, S, K, V).

Input: G is the codebook (mapping: item id $\rightarrow$ sub-item tokens)
Input: S is the matrix of pre-computed sub-item scores, indexed by split and token
Input: K is the number of results to return
Input: $\tilde{I} \subseteq I$ are the items to score; all items ($\tilde{I} = I$) if not given
1: $scores \leftarrow$ empty array of scores for all items in $\tilde{I}$, initialised to 0
2: **for** $item\_id \in \tilde{I}$ **do** ▷ This loop can be efficiently parallelised
3: $score[item\_id] \leftarrow \sum_{k=1}^{m} S[k, G[item\_id, k]]$
4: **end for**
5: **return** TopK($score$, K) ▷ Returns a list of ⟨ItemId, Score⟩ pairs

The number of splits m and the number of sub-ids per split b are usually chosen to be relatively small, so that the total number of sub-id scores is much less compared to the size of the catalogue, e.g., $m \times b \ll |I|$.

Therefore, this allows to compute the matrix S only once for a given sequence embedding and then reuse these scores for all items. This leads to efficiency gains compared to matrix multiplication, as scoring each item now only requires $m \ll d$ additions instead of d multiplications and d additions per item. The time for pre-computing sub-item scores does not depend on $|I|$ and we can assume that it is negligible w.r.t. the exhaustive scoring of all items.

Algorithm 2 illustrates the PQTopK in pseudocode. Note that the algorithm has two loops: the outer loop (line 2) iterates over the items in the catalogue, and the inner loop (line 3) iterates over codes associated with the item. However, as the item scores are independent of each other, both loops can be efficiently parallelised[2].

The original RecJPQ code[3] is also based on the same idea of pre-computing item scores and then computing item scores as the sum of associated sub-id scores. However, in the original RecJPQ code, the order of loops is swapped compared to the PQTopK algorithm: the outer loop iterates over the splits, and in the inner loop, the scores for each item are accumulated for each item (we list RecJPQ's original scoring algorithm in Algorithm 3). Due to the iterative accumulation of item scores, the outer loop in RecJPQ's scoring algorithm is not parallelised. In Section 7.1.5, we show that this makes original RecJPQ's scoring algorithm less efficient compared to PQTopK.

2. We achieve parallelisation using TensorFlow accelerated computation framework.
3. Here, we refer to the version of the RecJPQ code that we prepared to supplement the original publication [180].

Algorithm 3 RecJPQScore(G, S, K, V) Scoring algorithm originally used in RecJPQ.

Input: G is the codebook (mapping: item id $\rightarrow$ sub-item ids), Eq. (7.2)
Input: S is the matrix of pre-computed sub-item scores, indexed by split and sub-item, Eq. (7.5)
Input: K is the number of results to return
Input: $V \subseteq I$ are the items to score; all items ($V = I$) if not given

```
scores ← empty array of scores for all items in V, initialised to 0
for k ∈ 1..m do                          ▷ Not parallelised in the original RecJPQ code
    for item_id ∈ V do
        score[item_id] += S[k, G[item_id, k]]
    end for
end for
return TopK(score, K)
```

7.1.4 PQTopK Experimental Setup

We designed our experiments to answer two research questions:

PQTopK Research Questions

RQ7.1: How does PQTopK inference efficiency compare to baseline item scoring methods?

RQ7.2: How does PQTopK inference efficiency change when increasing the number of items in the catalogue?

Datasets. We experiment with two real-world datasets: Booking.com [63] (∼35K items) and Gowalla [32] (∼1.3M items). We follow Section 2.4.1 for data preprocessing. Additionally, to test the inference speed of different scoring methods, we use simulated data with up to 1 billion items in the catalogue.

Backbone Models. In RQ7.1, we experiment with two commonly used Transformer models, SASRec and BERT4Rec. To be able to train the models on large catalogues, we replace the item embedding layer with RecJPQ. Moreover, the original BERT4Rec model does not use negative sampling, which makes it infeasible to train on large catalogues, such as Gowalla (see Section 4.4.3.2). Hence, to be able to deal with large catalogues, we use gBERT4Rec (Section 5.5.5), a version of BERT4Rec trained with negative sampling and gBCE loss. The configuration of the models follows experimental methodology from Section 6.5. In particular, we use 512-dimensional embeddings; we use 2 Transformer blocks for SASRec and 3 Transformer blocks for BERT4Rec. When answering RQ7.1, we use RecJPQ with $m = 8$ splits but vary m in RQ7.2. In RQ7.2, we exclude the backbone model from our analysis; therefore, the results are model-agnostic and apply to any backbone.

Scoring Methods. We analyse three scoring methods: (i) Transformer Default, matrix multiplication-based scoring $r = V\phi$ used by default in SASRec and BERT4Rec (w/o any RecPQ enhancements); (ii) the original RecJPQ scoring (Algorithm 3); (iv) PQTopK scoring (Algorithm 2). We implement[4] all algorithms using TensorFlow [1].

Metrics. Our main focus is on the model inference speed. We measure inference using the median scoring time per user (mST, time required by the model to return recommendations). We do not use GPU acceleration when measuring any response time (details of our hardware configuration are in Table 7.1). We separately measure total response time, time spent by the model for running the backbone Transformer model, and time spent by the scoring algorithm. For completeness, we also report effectiveness using NDCG@10, even though optimising model effectiveness is outside of the scope of this chapter and all scoring methods for RecJPQ-based models have the same effectiveness.

7.1.5 Analysis of the experiments with PQTopK

RQ6.1. Comparison of PQTopK and other scoring methods. Table 7.2 reports effectiveness and efficiency metrics for SASRec and BERT4Rec on both Booking.com and Gowalla datasets. We first observe that nDCG@10 values do not depend on the scoring method, as all algorithms compute the same score distribution. We also see that the backbone model inference time does not depend on the scoring method as well, as different scoring methods are applied on top of the backbone Transformer model (i.e. we use different "heads" in Transformer terminology). Interestingly, the time required by the backbone Transformer model does not depend on the dataset either: e.g., BERT4Rec requires roughly 37 milliseconds on both Booking and Gowalla, while SASRec requires roughly 24 milliseconds[5]. This makes sense as Transformer complexity depends on the embedding dimensionality, the number of Transformer blocks and the sequence length but not on the number of items in the catalogue.

On the smaller Booking.com dataset, we see that the running time of the backbone Transformer model dominates the total model response time, and the differences between different scoring methods are rather unimportant. For example, when using gBERT4Rec on this dataset, the slowest scoring method (Transformer Default) requires 43 milliseconds per user. In contrast, the faster

4. Code for this section: https://github.com/asash/RecJPQ-TopK.
5. We speculate that the difference between SASRec (Transformer Decoder-based) and BERT4Rec (Transformer Encoder-based) is likely due to implementation-specific factors such as implementation of the attention mechanism, or framework-level optimizations.

Table 7.1: Hardware Configuration

CPU	AMD Ryzen 5950x
Memory	128 GB DDR4
OS	Ubuntu 22.04.3 LTS
Accelerated computing framework	TensorFlow 2.11.0
GPU Acceleration	Not used

Table 7.2: Efficiency analysis of item scoring methods. mST is the Median Scoring Time, measured in milliseconds; SAS is the SASRec model and gBERT is the gBERT4Rec model.

		Dataset: Booking			Dataset: Gowalla		
	Scoring method	mST (Scoring)	mST (Total)	Backbone measures	mST (Scoring)	mST (Total)	Backbone measures
gBERT	Default*	6.22	43.37	NDCG@10: 0.328	133.40	171.04	NDCG@10: 0.168
	RecJPQ	3.90	41.08	Model Response Time:	33.87	71.42	Model Response Time:
	PQTopK	**3.09**	**40.23**	37.16	**13.79**	**51.33**	37.52
SAS	Default*	6.27	30.03	NDCG@10: 0.188	131.35	156.07	NDCG@10: 0.120
	RecJPQ	3.77	27.53	Model Response Time:	29.65	54.32	Model Response Time:
	PQTopK	**2.93**	**26.69**	23.75	**10.03**	**34.72**	24.67

method (PQTopK) requires 40 milliseconds (Δ<10%) – even though PQTopK is two times faster compared to Transformer Default scoring when comparing without the backbone model inference (6.22ms 3.09ms). In contrast, on the larger Gowalla dataset with more than 1M items, there is a large difference between different scoring methods. For example, when using Default Transformer scoring with SASRec, inference time is dominated by the item scoring (131ms out of 171ms).

When using SASRec as the backbone with original RecJPQ scoring, both the backbone and the scoring head contribute similarly towards total scoring time (SASRec takes 24ms while scoring takes 29ms). In contrast, when using PQTopK, the total time is dominated by the Transformer model itself (e.g., PQTopK only uses 10ms. out of 34 when using the SASRec backbone). If we isolate scoring time, the Gowalla with SASRec backbone dataset with PQTopK is $13\times$ faster than the Transformer default and $3\times$ faster than the original RecJPQ scoring.

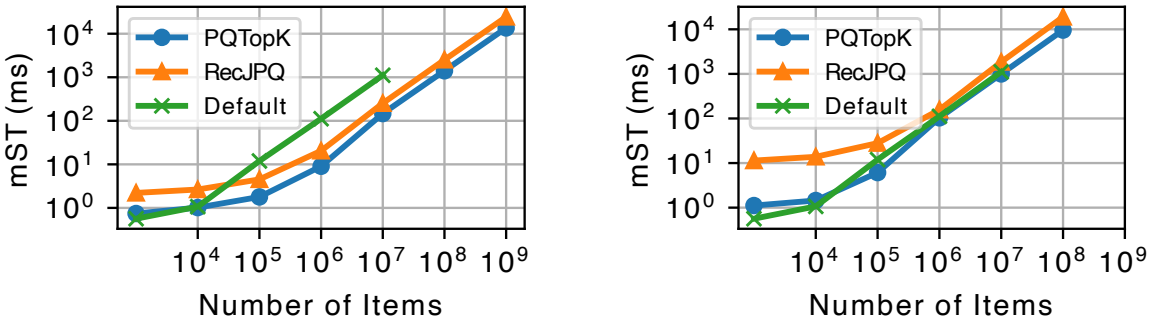
In summary, answering RQ7.1, we find that PQTopK is the most efficient method among the baselines. On the Gowalla dataset with more than a million items, PQTopK requires much less time compared to backbone Transformer models. On the other hand, on smaller datasets with only a few thousand items (such as Booking.com), even Default Matrix Multiplication remains efficient.

RQ7.2. PQTopK efficiency with very large catalogues. As observed in RQ7.1, the inference time of a backbone Transformer model (without scoring head) is constant w.r.t. catalogue size $|I|$. Therefore, as our goal is efficiency analysis, we exclude the Transformer model from the analysis and simulate it using a random model output for each output. We also generate a random sub-id embedding matrix $\hat{C}$ to compute item scores. In all cases, we include the time required for selecting top-k (`tf.math.top_k()` in TensorFlow) after scoring, as this time also depends on the number of items in the catalogue.

Figure 7.2 reports the mean response time for Default Transformer scoring, PQTopK and RecJPQ without the backbone Transformer model, for $m = 8$ splits (7.2a) and $m = 64$ splits (7.2b). Both Figures 7.2a & 7.2b include the matrix multiplication-based Transformer Default baseline that does not use the number of splits.

We observe from the figures that with a low number of items in the catalogue ($\leq 10^4$), the default matrix multiplication-based approach is the most efficient, requiring less than a millisecond for scoring. However, as we observed in RQ7.1, with this small number of items, the actual method is not that important, as the scoring time is likely to be dominated by the backbone model inference.

With the smaller number of splits, $m = 8$, matrix multiplication becomes less efficient compared to PQ-based methods for item catalogues with more than 10^5 items. Note that the figure is shown in logarithmic scale, meaning that, for example, at 10M items, PQTopK is $10\times$ more efficient compared to the default approach. Also, note that the matrix multiplication baseline only extends up to 10^7 items: after that point, the default approach exhausts all available system memory (128GB). We also observe that PQTopK is always more efficient than RecJPQ. Despite (due to the logarithmic scale) the lines looking close to each other, PQTopK is always faster than RecJPQ by 50-100%. For example, with 10M items in the catalogue, PQTopK requires 146ms per user, whereas RecJPQ requires 253ms (+68%). With 100M items in the catalogue, PQTopK remains relatively efficient ($\approx$ 1 second per user); however, with 1 billion items, the method requires more than 10 seconds per user. Arguably, 10 seconds per item is not suitable for interactive recommendations (for example, when the model inference occurs during web page loading), but may still work in situations when recommendations can be pre-computed (e.g. updated once every day).



(a) Number of splits: $m = 8$ **(b)** Number of splits: $m = 64$

Figure 7.2: Efficiency of PQTopK on simulated data

On the other hand, as we can see from Figure 7.2b, with a large number of splits ($m = 64$), Default and PQtopK perform similarly; e.g., both methods require ∼100ms for scoring 1M items, 50ms faster than RecJPQ. However, on our hardware, Default consumes all available memory above 10M items (this is why the line for Default on Figures 7.2b and 7.2b does not go beyond 10^7 items), whereas PQTopK and RecJPQ allow for scores up to 100M items. Nevertheless, PQTopK scoring, in this case, requires 10 seconds per user, limiting its application to the pre-computing scenario.

Finally, we observe that with catalogues with more than 10^5 items, the response depends linearly on the number of items for all scoring methods. However, with less than 10^5 items, there is an "elbow-style" non-linearity that can be explained by the fact that the time required by auxiliary operations such as function calls becomes important at this small scale.

Summarising RQ7.2, we conclude that PQTopK with 8 splits is a very efficient algorithm that allows performing efficient inference on catalogues even with hundreds of millions of items. With a larger number of splits $m = 64$, the inference time of PQTopK is similar to the default matrix multiplication scoring, but it allows scoring up to 10^8 items. In contrast, matrix multiplication exhausts available memory with catalogues larger than 10^7 items, which highlights the importance of RecJPQ for reducing memory consumption.

7.1.6 PQTopK Summary

This section analysed the inference time of Transformer-based sequential recommender systems with large catalogues. We found that using RecJPQ enhancement, which enables training on large catalogues via sub-item-id representation, coupled with an efficient PQTopK scoring algorithm, allows model inference on large catalogues. In particular, using PQTopK, we sped up RecJPQ-enhanced SASRec 1.56× compared to the original RecJPQ scoring and 4.5× compared to default SASRec scoring on the Gowalla dataset with 1.3M items. We also showed that, when considering the pre-scoring scenario, PQTopK can be applicable to catalogues of up to 1 billion items.

However, despite being efficient, the PQTopK algorithm still requires exhaustive catalogue scoring. Hence, with very large catalogues, it still may become slow and expensive degradation, as evidenced by Figure 7.2. Hence, in the next section, we design a novel RecJPQPrune algorithm that builds upon PQTopK but allows for the exact finding of Top K items without an exhaustive catalogue scoring.

7.2 Avoiding Exhaustive Scoring with Dynamic Pruning

7.2.1 Introduction

As described in Section 7.1, most sequential recommendation models use the "score-and-rank" approach: they first compute scores for all items in catalogue and then return Top-K highest-ranked items as recommendations. Indeed, even the PQTopK algorithm still applies the score-and-rank approach, which requires scoring all items. However, we observe that the way PQTopK computes item scores given the scores of its sub-items is similar to how traditional information retrieval (IR) models compute document scores given a query: an item score can be computed as the sum of the scores of the individual sub-items, which is similar to computing document score as a sum of token scores in the "bag-of-word" retrieval method, such as BM25 [207].

This inspires us to examine the applicability of *dynamic pruning* techniques that are typically employed to increase the efficiency of bag-of-word retrieval methods. Indeed, in this section, we propose the pruning-based RecJPQPrune method for efficient calculation of top K ranked items under RecJPQ. The idea of the method is based on the hypothesis that highly-ranked items should also have highly-scored sub-items. Instead of *exhaustively* computing scores for all items in the catalogue, we only compute scores for items that are associated with the highest-scored sub-items. Moreover, the specifics of sub-item representations in RecJPQ allow us to compute an upper bound for the item scores, which allows us to stop the item scoring way before all items in the catalogue have been scored.

Recent generative recommender systems, such as TIGER [194] and GPTRec [177], also rely on sub-item representations and avoid exhaustive item scoring by generating item ids autoregressively. However, both TIGER and GPTRec mention generation speed as a limitation, as they require a Transformer invocation for every generated sub-id, which makes them inefficient for retrieval and outside the scope of this section.

In summary, this section contributes:

1. A novel dynamic pruning approach, RecJPQPrune, for speeding up inference for large-catalogue RecJPQ-based recommender models, with no impact on effectiveness;
2. Experiments examining median and tail scoring times on two large datasets with millions of items, and for three Transformer-based sequential recommender models;
3. A study into factors affecting the efficiency of RecJPQPrune for different models and different users

RecJPQPrune provides marked efficiency benefits – for instance, on the Tmall dataset with 2.2M items, we can reduce the median model scoring time by 64$\times$ compared to default Transformer scoring, and 5.3$\times$ compared to PQTopK.

The structure of this section is as follows: Section 7.2.2 outlines existing applications of pruning in search and other machine learning scenarios. Section 7.2.3 describes RecJPQPrune. Our experimental setup, and our empirical validation demonstrating the benefits of RecJPQPrune follow in Sections 7.2.4 & 7.2.5. To explain the variance between models and between users, in Section 7.2.6 we make a first examination of pruning difficulty in RecJPQPrune.

7.2.2 Pruning in Document Retrieval

In the classical document retrieval task, the goal of the Information Retrieval (IR) system is to retrieve textual *documents* from a document collection $\mathcal{D}$ that are estimated most likely to be *relevant* to a given textual query q. We focus on "bag-of-word" retrieval approaches, as exemplified by BM25 [207], in which both the documents $d \in \mathcal{D}$ and the query q are represented as multisets of *terms* t.

Bag-of-word approaches compute query-document relevance estimates as:

$$\text{score}(q, d) = \sum_{t \in q} w(t, d) \tag{7.7}$$

where $w(t, d)$ is the weight of the term t in the document d. Each document has a non-negative integer known as a *document identifier* (docid). Every term present in the collection has a *posting list*, which comprises the docids of all documents where the term appears. The aggregated posting lists for all terms form the *inverted index* of $\mathcal{D}$.

The docids within a posting list can be arranged in ascending order, or by descending score/impact [6]; we assume such ordering for the remainder of the section. The traditional approaches for processing queries and matching them to documents are the *term-at-a-time* (TAAT) strategy, where the posting lists of query terms are processed sequentially, and the scores for each document are summed up in an *accumulator* data structure; or *document-at-a-time* (DAAT), where the posting lists of query terms are processed simultaneously while maintaining docid alignment.

Processing queries exhaustively with TAAT or DAAT can be very inefficient. As a result, various dynamic pruning techniques [241] have been proposed[6], which aim to omit the scoring of (portions of) documents during query processing if they cannot make the final top K retrieved set. Dynamic pruning strategies can be described as *safe-up-to-rank* K – meaning they are guaranteed to calculate the exact scores for each retrieved document, at least as deep as rank K – or *unsafe* – indicating that their retrieval effectiveness may be negatively impact compared to an exhaustive scoring.

---

6. In the deep learning literature, the term *pruning* typically means *weight pruning*. In our case, we use the term pruning as used in IR literature [241], meaning pruning candidates from the scoring process to speed up scoring. For applications of weight pruning to recommender systems see, for example, [218] or [43].

All dynamic pruning optimisations for TAAT involve a two-phase strategy. In the initial phase, the TAAT algorithm is applied, processing one term at a time in ascending order of document frequency. New accumulators are created and updated until a pruning condition is satisfied. Subsequently, the second phase commences, during which no new accumulators are created [18, 159, 160].

Among the dynamic pruning strategies for DAAT, MaxScore [246], WAND [16], and their variants [48, 152] stand out as the most widely used. Both approaches enhance the inverted index by recording the maximum score contribution for each term. This enables the safe skipping of substantial segments within posting lists if those segments only consist of terms whose combined maximum scores are lower than the scores of the top K documents already identified during query processing, known as the threshold, and denoted θ. They also utilise a global per-term upper bound, i.e., the maximum score across all documents containing the term, in order to make pruning decisions.

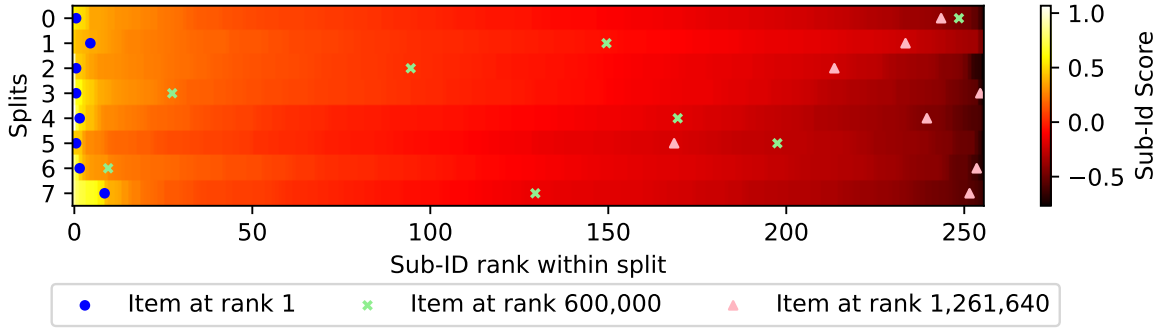
Finally, there are a number of dynamic pruning strategies for impact-ordered posting lists, such as score-at-a-time [6]. Similarly, our approach uses score-sorted ids to perform computations as efficiently as possible [148], but we do not leverage related dynamic pruning techniques e.g. anytime ranking [288], due to their inherent unsafeness, leaving to future work the analysis of unsafe settings.

In this work, we propose a safe-up-to-rank-K novel dynamic pruning strategy, RecJPQPrune, which is a hybrid of both TAAT and DAAT dynamic pruning, but designed specifically for scoring RecJPQ item representations in recommender systems. We position RecJPQPrune within the dynamic pruning literature in Section 7.2.3.

Other works focussed on improving the efficiency of expensive machine-learned ranking models have addressed (i) early termination of regression trees [22], or, (ii) more recently, the early termination of layers in Transformer-based models [263], such as cross-encoders. These optimisations are applied for each candidate document being ranked.

In our setting, the Transformer model needs to be applied only once to obtain a representation of the user's recommendation need, like a query encoder in neural dense document retrieval. Hence, approaches that speed up model inference are orthogonal to our proposed RecJPQPrune strategy.

Figure 7.3: Relation between item ranks and sub-item scores for user 82082 from Gowalla: SASRecJPQ model [180] with 8 splits and 256 sub-item ids per split. We highlight items ranked at the top, middle and bottom of the ranking.



7.2.3 RecJPQPrune

We now describe our RecJPQPrune method, discussing the principles we use for dynamic pruning of RecJPQ-based representations (Section 7.2.3.1), the algorithm itself (Section 7.2.3.2), and its positioning w.r.t. existing IR dynamic pruning strategies (Section 7.2.3.3).

7.2.3.1 Dynamic Pruning Principles for RecJPQPrune

Our goal is to build an algorithm that allows us to find the top-ranked items while avoiding exhaustive catalogue scoring. In Section 7.2.2, we discussed that for document retrieval, a similar problem can be solved using pruning techniques. Comparing how document scores are computed in bag-of-word document retrieval models (Equation (7.7)) and how item scores are computed in PQTopK (Equation (7.6)), we find parallels between the two: in both cases, the final entity score is computed a sum of individual sub-entity scores. However, addressing the efficient computation of item scores requires the construction of a novel dynamic pruning algorithm. RecJPQPrune is based on three principles that allow the scoring of some items to be omitted. We now describe these principles, while, later in Section 7.2.3.3, we compare and contrast RecJPQPrune with existing dynamic pruning approaches.

P7.1: Items with high scores typically have sub-items with high scores as well.

The intuition behind this principle is that the total score of an item is calculated as the sum of its sub-item scores. Hence, a high overall score generally requires several high-scoring sub-items. Figure 7.3 shows empirical evidence for this principle. The figure illustrates how sub-item id scores are distributed for user 82,082 in the Gowalla dataset computed by the RecJPQ-enhanced SASRec model, where the number of splits $M = 8$ and the number of sub-items per split $B = 256$. In the figure, the sub-item ids are ranked within their splits according to their score, with the highest-scored sub-item ids (bright-yellow colours) being on the left of the figure and the lowest-scored sub-item ids (dark-red colours) being on the right of the figure. The figure highlights the sub-item ids associated with a top-ranked item, a middle-ranked item and a bottom-ranked item. As we can see, all sub-item ids of the top-ranked item appear to the left of the figure, i.e., they are scored high within their respective splits, and have bright colours, i.e., they are scored high across all sub-item ids. For the middle-scored item, we have a mixture of relatively high and relatively low-scored sub-item ids; however, none of the sub-item ids is the highest-scored sub-item id in the respective split. Most of the sub-item ids of the low-scored item also scored low. Overall, Figure 7.3 supports Principle P7.1.

In summary, Principle P7.1 suggests processing first the items associated with highly-scored sub-item ids during scoring before processing those linked to less highly-scored sub-item ids. This ensures we are likely to encounter all high-scored items relatively quickly. However, to achieve efficiency gains compared to exhaustive scoring, we need to be able to terminate scoring after all high-scored items have been found; to do that, we use Principle P7.2.

P7.2: We can terminate scoring once the remaining items have no chance to enter the top K results

Inspired by existing dynamic pruning techniques, we consider if, after processing a few sub-item ids as described in Principle P7.1, we can guarantee that any item in the set of yet-to-be-scored items $I^U \subset I$ cannot enter the current top K items. Indeed, after a few iterations of scoring items associated with highly-scored sub-item ids, as described in Principle P7.1, we have, for each split m, a list U_m of unprocessed sub-item ids; moreover, let $U = \{U_1, U_2, ..U_m\}$ denote all unprocessed sub-item ids across all m splits. The not-yet-scored items, I^U, are guaranteed to have all their sub-item ids appearing in U, because items associated with already processed sub-item ids were scored when their respective sub-item ids were processed. Therefore, from

Equation (7.6), we can derive a score upper bound σ for any unscored item $i \in I^U$:

$$r_i = \sum_{k=1}^{m} s_{k,g_{ik}} \leq \sum_{k=1}^{m} \max_{j \in U_k} s_{k,j} = \sigma \tag{7.8}$$

When processing sub-item ids as described in Principle P7.1, we can keep the minimum score within the current top K highest scored items as a threshold θ. While the following *pruning condition* holds

$$\sigma > \theta \tag{7.9}$$

some items can still be entered into the top K items. However, θ rises as items are admitted into the top K, and σ falls as the unprocessed sub-item ids are less important. When the condition Equation (7.9) no longer holds, we can guarantee that no item that has not been scored yet can enter into the top K items; therefore, we can safely terminate the scoring algorithm. In summary, Principle P7.2 argues that top K items can be found without exhaustive scoring of all items in the catalogue, and provides us with the pruning condition that helps to identify the moment when scoring can be terminated.

P7.3: Highly scored sub-item ids are frequently found in the same split.

In RecJPQ, sub-item ids are obtained using SVD decomposition of the user-item interaction matrix with different splits corresponding to different latent features of items. This means that if two items had similar values of a latent feature in the SVD decomposition, they would also have similar sub-item ids in the corresponding split. In short, in RecJPQ, *similar items are assigned to similar sub-item ids*. As a result, in RecJPQ, highly scored sub-item ids are frequently clustered in the same split. For example, looking again at Figure 7.3, we see that most of the highly-scored sub-item ids for the illustrated user are located in the last split. This suggests that once we find a promising sub-item id, we can process not only items associated with this sub-item id, but also items associated with other highly-scored sub-item ids from the same split. In other words, Principle P7.3 allows for *batch processing* of sub-item ids.

7.2.3.2 RecJPQPrune Algorithm

Using Principles P7.1-P7.3, we can now derive the RecJPQPrune algorithm, illustrated in Algorithm 4.

According to Principle P7.1, RecJPQPrune processes sub-item ids in the descending order of their scores stored in S (S is the same matrix of pre-computed sub-item id scores, as used by PQTopK, see Section 7.1.3). The values in S are computed efficiently at line 3. Then, it sorts sub-item ids into the array Q according to their scores within each split (line 7), using the array P to track the position of the unprocessed sub-items in each split. After that it iterates through the splits in Q and positions in P while the pruning condition holds (line 12). At each iteration, it finds the maximum-scored unprocessed sub-item ids and corresponding split, denoted as k^* (line 13). From the split k^*, RecJPQPrune scores BS sub-item ids at a time, using the variable i^* (line 15). In order to be able to quickly score all items associated with the best split m^* and the batch of best sub-item ids i^*, RecJPQPrune uses m *inverted indexes* $\mathcal{L}_1, \ldots, \mathcal{L}_m$. For a given split $k \in [m]$, the inverted index $\mathcal{L}_k$ maps a sub-item id to the set of all item id associated with the sub-item id (in effect, $\mathcal{L}_1, \ldots, \mathcal{L}_m$ are the inverse of RecJPQ's codebook G). When processing i^*, RecJPQPrune retrieves all items associated with it from the inverted index $\mathcal{L}_{k^*}(i^*)$ (line 17). Then it computes the scores of all items associated with this sub-item id using the PQTopK algorithm (line 19), and updates the current best top K items (line 20). Note that because every item is associated with multiple sub-item ids, RecJPQPrune may score some items multiple times; therefore, when updating current best top K items using the `merge` operation, RecJPQPrune also deduplicates any repeated items. It then removes the sub-item ids in the batch from the unprocessed sub-items (line 21), updates the upper bound σ (line 24) and the pruning threshold θ (line 26). RecJPQPrune iterates until the pruning condition (Equation (7.9)) is met, after which it terminates and returns the current best top K items.

The use of batch processing addresses Principle P7.3, in that BS sub-item ids are identified at each outer loop iteration, and all items associated with these sub-item ids are processed in a single iteration. Following Principle P7.3, all these sub-item ids are taken from the same split k^*, enabling effective vectorisation of the for-loop at line 15. Moreover, the for-loops at lines 5 & 23, the PQTopK algorithm and the `merge` operation are also vectorisable using common tensor manipulation frameworks, such as TensorFlow or PyTorch (all vectorisable operations are coloured teal in Algorithm 4). Our TensorFlow implementation can be found in our source code repository.[7] The batch size, BS, is an important parameter of the algorithm. On the one

7. Code for this section: `https://anonymous.4open.science/r/recjpq_dp_pruning-A17C/`

Algorithm 4 RecJPQPrune(ϕ, G, $\mathcal{L}$, K, BS)

Input: Sequence embedding ϕ
Input: Codebook G
Input: Inverted indexes $\mathcal{L}_1, \ldots, \mathcal{L}_m$
Input: Number of results to return K
Input: Number of sub-item ids processed at every iteration BS
$P \leftarrow$ empty array of m current sub-item positions, one per split
$Q \leftarrow$ empty array of m empty sub-item id lists, one per split
$S \leftarrow$ compute the sub-item scores matrix
$\sigma \leftarrow 0$
for $k = 1, \ldots m$ **do**
    $P[k] \leftarrow 1$
    $Q[k] \leftarrow$ sorted sub-item ids in split k according to scores in S
    $\sigma \leftarrow \sigma + S_{k,Q[k][1]}$
end for
$R_K \leftarrow$ empty list of $\langle$item id, score$\rangle$ pairs
$\theta \leftarrow -\infty$
while $\sigma > \theta$ **do**
    $k^* \leftarrow \arg\max_{1 \leq k \leq m} S_{k,Q[P[k]]}$
    $I^* \leftarrow$ empty list of $\langle$item id, score$\rangle$ pairs
    for $j = 0, \ldots, BS - 1$ **do**
        $i^* \leftarrow Q[k^*][P[k^*] + j]$
        $I^* \leftarrow I^* \cup \mathcal{L}_{k^*}(i^*)$
    end for
    $I_K \leftarrow \texttt{PQTopK}(G, S, K, I^*)$
    $R_K \leftarrow \texttt{merge}(R_K, I_K, K)$
    $P[k^*] \leftarrow P[k^*] + BS$
    $\sigma \leftarrow 0$
    for $k = 1, \ldots, m$ **do**
        $\sigma \leftarrow \sigma + S_{k,Q[k][P[k]]}$
    end for
    $\theta \leftarrow R_K[K]$.score
end while
return R_K

hand, larger batch sizes increase parallelism, hence making the algorithm more efficient. On the other hand, by increasing the batch size, we score additional items at every iteration, and hence, we may score more items than necessary before reaching the pruning condition. We address the selection of the appropriate batch size in Section 7.2.5.3.

Safety of RecJPQPrune. RecJPQPrune is a safe-up-to-rank K dynamic pruning algorithm. Indeed, compared to the scoring of all items from all sub-items, the same exact scores are obtained to rank K but minimising the processing of items that do not make the final top K. The safety is guaranteed by the fact that at termination time, the upper bound for scores for unprocessed items σ is lower than the minimum score of the K best items that the algorithm already found

so that no unprocessed item can be included in the top K items. Like existing dynamic pruning techniques, it is possible to make RecJPQPrune more efficient but unsafe (for instance by overinflating the threshold θ), resulting in potential effectiveness degradations. However, in this work, we focus on the safe setting, and leave unsafe settings to future work.

7.2.3.3 Relation to Dynamic Pruning Literature

We now highlight parallels and contrasts with previous work in dynamic pruning. Firstly, we draw parallels in terms of nomenclature: items are documents; and sub-item ids are like terms, except that each item has a fixed number m of sub-item ids, one from each split. These observations help us to position RecJPQPrune within the dynamic pruning literature. Indeed, existing dynamic pruning techniques cannot address this task, as our sequence embedding (query) can match with any sub-item id (term).

Principle P7.1 suggests scoring items associated with highly-scored sub-item ids. This is similar to how optimised versions of the TAAT Pruning score query terms in the decreasing order of Inverted Document Frequency [241, Sect. 3.2], allowing to find documents that are likely to be highly scored earlier. The use of a threshold θ from the top-ranked items is commonly deployed in DAAT dynamic pruning approaches (MaxScore, WAND) for the purposes of early-terminating the scoring of documents.

Some other dynamic pruning approaches used impact-ordered postings lists [241, Ch. 5]; on the surface this has some similarities to our work, however, the inclusion of splits, sub-item ids, etc. makes comparisons challenging. However, we note that Jia et al. [96] used updating upper bounds and a scoring terminating condition that bears resemblance to Principle P7.2. Principle P7.3 is novel, as we are not aware of dynamic pruning document retrieval literature considering batching of processing and the benefits of vectorisation. Moreover, we argue that RecJPQPrune is neither exclusively a DAAT nor a TAAT algorithm: like TAAT, it identifies a good sub-item id (term) to score next. Once that high-scoring sub-item id is identified, all items associated with that sub-item id are fully scored in DAAT fashion (as per Algorithm 4). In short, RecJPQPrune is a novel application of dynamic pruning ideas to RecJPQ-based scoring.

7.2.4 Experimental Setup for evaluating RecJPQPrune

Our experiments aim to answer the following Research Questions:

RecJPQPrune Research Questions

RQ7.3: What is the effect of applying our RecJPQPrune algorithm on scoring efficiency?

RQ7.4: What is the effect of ranking cutoff K on the efficiency of our RecJPQPrune algorithm?

RQ7.5: What is the effect of varying the batch size on the efficiency our RecJPQPrune algorithm?

In the following, we detail the datasets used in our experiments (Section 7.2.4.1), the used recommender models (Section 7.2.4.2), and the measures we adopt to answer our research questions (Section 7.2.4.3).

7.2.4.1 Datasets

The focus of this chapter is on large catalogues. Hence, in our experiments, we use datasets with some of the largest catalogues available for academic research. In particular, we perform experiments using two large-scale sequence recommendation datasets, namely (i) Gowalla [32], a point-of-interest check-in dataset, and (ii) Tmall [237], an e-commerce clicks dataset. Table 2.1 in Section 2.4.1 provides the statistics of the datasets. Of note, both have larger numbers of items, i.e., 1.2M items in Gowalla and 2.1M items in Tmall, than conventional recommendation datasets such as MovieLens-1M, which has only 3K items. Indeed, as shown in Section 7.1.5, for less than 30K items, even Transformer Default is very efficient, and further efficiency optimisation is not required

Data pre-processing and splitting in this chapter follow our common experimental setup (Section 2.4.1) except for using temporal leave-one-out strategy instead regular leave-one-out.[8]

8. Regular leave-one-out has recently been critiqued for potential data leakages [79].

7.2.4.2 Models & Baselines

Following Section 7.1.4 we use RecJPQ versions of SASRec (SASRecJPQ) and gBERT4Rec (gBERT4RecJPQ). In addition, to further show the generalisability of RecJPQPrune, we also experiment with RecJPQ-based version of gSASRec (section 5.5.5; gSASRecJPQ). For consistency, we align model training with our experiments with RecJPQ itself (see details in Section 6.5). We use $m = 8$ splits and $b = 256$ sub-ids per split.

For each model, we apply three methods for computing item scores: Transformer Default, which uses matrix multiplication, i.e. the scoring procedure used in Transformer models by default[9] (Equation (7.1)); PQTopK (Algorithm 2); and our RecJPQPrune method.

We do not consider ANN implementations such as FAISS [97], as they are not safe and cause significantly reduced retrieval effectiveness.[10] Note that we only use RecJPQ-based models for our experiments, as training plain Transformer models without embedding compression is not feasible using consumer-grade hardware with catalogues of this size. For example, on the Tmall dataset, a full embedding table would require 2.2M items $\times$ 512 parameters per embedding $\times$ 4 bytes = 4.5GB GPU memory. Considering the memory required for model gradients, moments, model parameters, and intermediate variables, we would need more than 24GB of GPU memory—exceeding what is currently available to us. This also prevents use of other PQ-based methods (such as those in FAISS), which require the training of full embeddings before compression.

7.2.4.3 Measures

We are primarily focused on efficiency, which we analyse using model scoring time.[11] Similarly to Section 7.1.3, our target environment considers only CPUs, i.e no GPU acceleration, at the inference time. Indeed, deploying a trained model on CPU-only hardware is often a reasonable choice for many high demand environments, considering the high costs associated with GPU accelerators. We exclude the time to obtain the sequence embedding through the Transformer layers, as this is a constant for all approaches. Furthermore, we report *median* (denoted 50%tl)

9. We use a RecJPQ-based version of the models, so to use the default Transformer scoring, we obtain full item embeddings first using concatenation, i.e., as per Equation (7.3). However, to ensure a fair comparison with Transformer Default, we do not include time spent on the reconstruction of the item embeddings in the scoring time.
10. Indeed, in preliminary experiments, we found that FAISS could result in 68% degradations in NDCG@10 compared to SASRecJPQ.
11. Recall that any reported time is specific to our hardware (see Table 7.1 for our hardware details).

scoring time instead of *mean* because we observe that in our TensorFlow-based implementation, JIT compilation requires several iterations to warm up. Following the dynamic pruning literature [20, 94, 147, 239], we also report 95th percentile scoring times, as dynamic pruning techniques can vary in the amount of pruning possible for different requests. Finally, we also report the number of items scored by each algorithm, because the primary goal of RecJPQPrune is to avoid exhaustive scoring.

7.2.5 Results

We now address each of the research questions RQ7.3-RQ3 in turn.

7.2.5.1 RQ7.3 - Overall Efficiency

We first analyse the efficiency of RecJPQPrune compared to the two baseline scoring methods, across three models (SASRecJPQ, gBERT4RecJPQ, gSASRecJPQ). Table 7.3 reports the median and 95th percentile scoring times (in milliseconds) of RecJPQPrune method compared to baseline methods on the two experimental datasets, Gowalla and Tmall. For RQ7.3, we apply $K = 10$ and a $BS = 8$, but we investigate the impact of these parameters in RQ7.4 & RQ7.5.

From the table, it can be clearly seen that the Transformer Default baseline involves excessive operations, resulting in large median scoring times: more than 100ms on Gowalla and more than 200ms on Tmall. Applying the existing method PQTopK, which re-uses pre-computed sub-item id scores, reduces the median time to around $9 - 16$ms – an average speedup of $10.7\times - 12.9\times$. Both of these baselines apply no pruning, so 95%tl times are very similar to the median.

On the other hand, applying our proposed RecJPQPrune dynamic pruning method, the median scoring times are reduced to 3-6ms, with a resulting speedup of $1.5\times$-$2.9\times$ on Gowalla and $3.2\times$-$5.3\times$ on Tmall compared to PQTopK. This demonstrates the benefit of pruning at this scale, and focussing on splits that are more likely to result in the highest scored items being retrieved.

Table 7.3: Median (50%tl) and tail (95%tl) scoring times of different scoring methods, in ms. The results are reported for ranking cutoff $K = 10$. For RecJPQPrune, batch size $BS = 8$.

Scoring Method	Model					
	SASRecJPQ		gBERT4RecJPQ		gSASRecJPQ	
	50%tl	95%tl	50%tl	95%tl	50%tl	95%tl
Gowalla						
Transformer Default	123.61	126.77	123.24	126.75	123.87	126.88
PQTopK	10.19	10.88	9.57	**10.61**	10.11	10.81
RecPQPRune (ours)	**3.50**	**8.51**	**6.42**	19.79	**4.59**	**7.99**
Tmall						
Transformer Default	204.18	210.48	205.67	210.76	206.38	210.89
PQTopK	16.72	18.26	16.66	17.76	16.75	19.68
RecPQPRune (ours)	**3.18**	**4.59**	**5.11**	**6.53**	**5.20**	**6.39**

Furthermore, considering the 95%tl times, we see that the pruning method can experience users that are difficult to prune; indeed, the 95%tl scoring time for SASRecJPQ and gSASRecJPQ are 1.2×-2.4× slower than the median (although always faster than PQTopK). We examine pruning difficulty more in Section 7.2.6. Similarly, the 95%tl scoring time for gBERT4RecJPQ on Gowalla is slower than for PQTopK; as we will see in the next section, this model/dataset combination is more difficult for pruning.

Overall, for RQ7.3, we find that RecJPQPrune can achieve improved median and 95%tl scoring times, a reduction of up to $5.3\times$ compared to the median scoring time of the recent PQTopK approach (SASRecJPQ on Tmall: 16.72ms $\rightarrow$ 3.18ms), and up to $64\times$ compared to the Transformer Default baseline (204ms $\rightarrow$ 3.18ms).

7.2.5.2 RQ7.4 - Ranking Cutoff

The efficiency of existing document retrieval dynamic pruning methods are sensitive to the rank cutoff, K, because the threshold θ is obtained from the score of the current top Kth ranked document, so retrieving fewer documents implies a higher threshold, causing less documents to be scored. Similarly, we expect reducing K to also increase the efficiency of RecJPQPrune, for the same reasons.

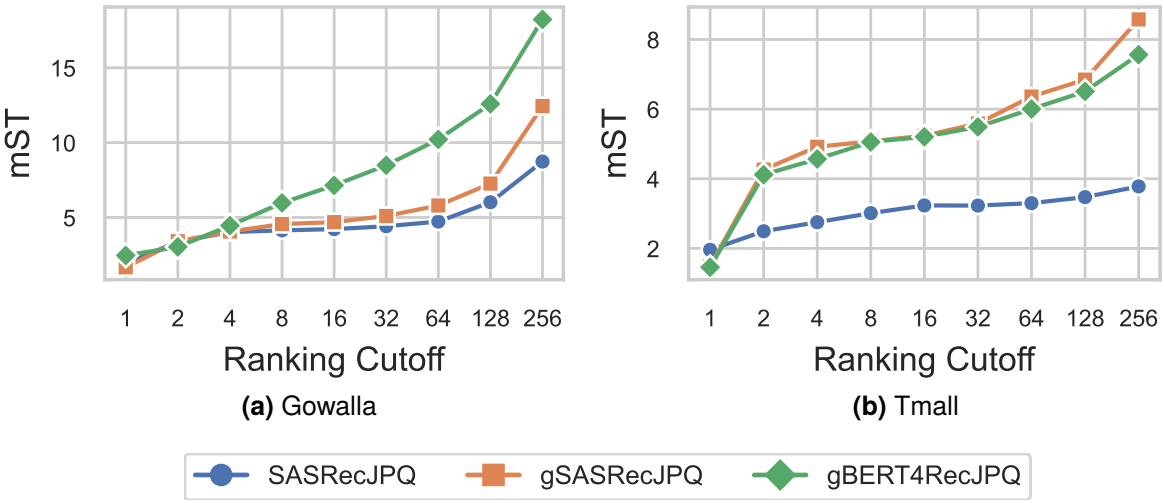


Figure 7.4: Effect of ranking cutoff on median scoring time (mST, ms).

To analyse this, we turn to Figure 7.4, which plots the scoring time of the various models on both Gowalla and Tmall datasets as the rank cutoff K is varied between 1 and 256. Its clear from the figures that, as expected, scoring time indeed increases as the rank cutoff increases. There appears to be a marked increase in scoring time between 128 and 256 retrieved items; however this is more an artifact of the logarithmic scale in the x-axis of the figures. We also observe some variance between the different models: SASRecJPQ is consistently the fastest model on both datasets as K is varied; gBERT4RecJPQ and gSASRecJPQ are typically slower (with gBERT4RecJPQ being slower than gSASRecJPQ for Gowalla). We examine the relative difficulty of pruning different models later in Section 7.2.6.

To summarise for RQ7.4, we find that, as expected, decreasing the rank cutoff decreases the scoring time. While this is expected from the existing dynamic pruning literature, it is a characteristic not previously observed Transformer-based recommender systems, where the Transformer Default method requires all items to be scored and then sorted for a given sequence embedding.

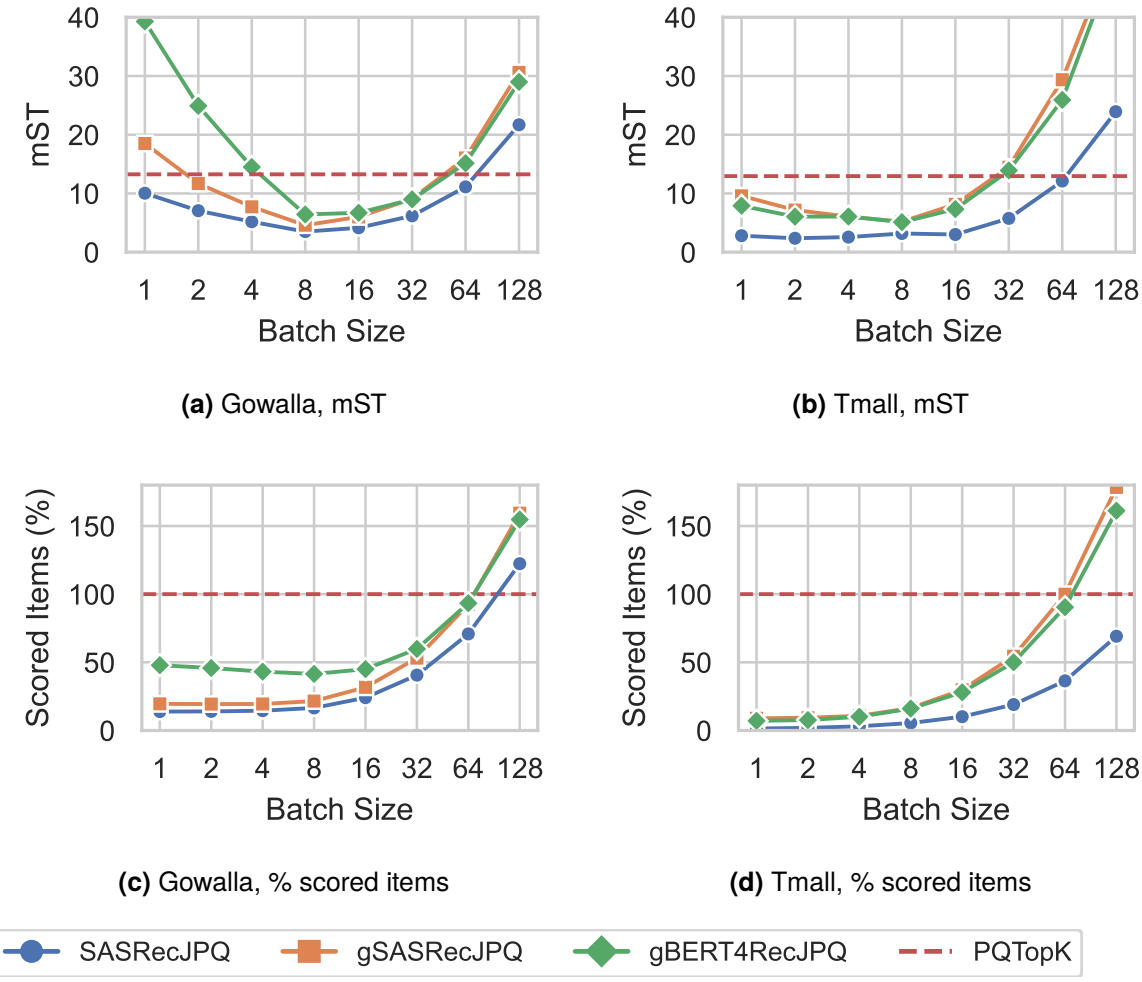
7.2.5.3 RQ7.5 - Batch Size

Finally, we consider the other parameter of RecJPQPrune, namely the batch size, BS, that controls how many sub-item ids are processed concurrently. Figure 7.5 reports the median scoring time as batch size is varied, on both Gowlla and Tmall datasets as well as the percentage of processed items during scoring. We present one line for each RecJPQ-based model, while for PQTopK we report an average across the three models (which are very close - see Table 7.3).

From Figure 7.5 (a) & (b), it is clear that for median scoring time there is a sweet spot for batch size BS - around 8 on both datasets and all models. The existence of this sweet spot confirms our Principle P7.3 and demonstrates the importance of batched processing. Smaller batch sizes typically result in increased scoring times - particularly so for gBERT4RecJPQ on Gowalla. The increases in scoring time suggest higher overheads from smaller batch sizes e.g. more applications of PQTopK and merge operations (lines 19 & 20 in Algorithm 4). High batch sizes also exhibit increased scoring times, as more items are scored than needed to achieve the pruning condition.

To quantify this behaviour, Figures 7.5 (c) & (d) report the percentage of items scored for different models as batch size is varied. From the figure, it can be seen that the percentage of items scored is typically reduced as batch size reduces - this makes sense, as fewer sub-items are selected at each main loop iteration. Indeed, it is expected that scoring time is heavily correlated with scored items [146, 238] - and therefore, the increased scoring times for small batch sizes come from the overheads, as discussed above. The exception here is again gBERT4RecJPQ on Gowalla. This outlier model is discussed further in Section 7.2.6. An interesting observation from the figures is that the percentage of items scored can exceed 100%. Indeed, RecJPQPrune does not maintain a set of already scored items, and the same item may be scored repeatedly when the algorithm processes different sub-ids associated with this item. While it could be possible to maintain a set of already processed items and process every item only once, our initial experiments showed that the overhead associated with maintaining such a set and checking every item is larger than the cost of repeated scoring of the items.

Overall, in answer to RQ7.5, we find that setting the batch size appropriately is important to achieve efficiency gains compared to the baselines. As suggested by Principle P7.3, setting the batch size too small (e.g. 1) increases computational overhead, while setting a high batch size increases the number of scored items. In our experiments, we find the ”sweet spot” for batch size is at value 8, which we recommend as a default value for RecJPQPrune.



(a) Gowalla, mST

(b) Tmall, mST

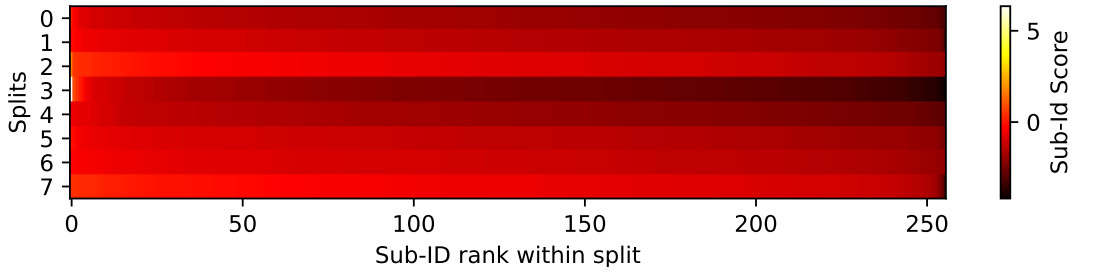
(c) Gowalla, % scored items

(d) Tmall, % scored items

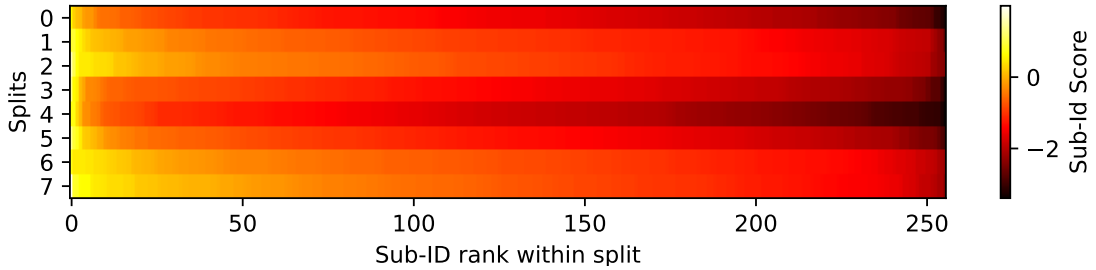
Figure 7.5: Effect of the batch size on median scoring time (mST, top) and the number of scored items (bottom) on Gowalla and Tmall in RecJPQPrune for ranking cutoff K=10. PQTopK baseline is averaged across the three models.

7.2.6 Pruning Difficulty

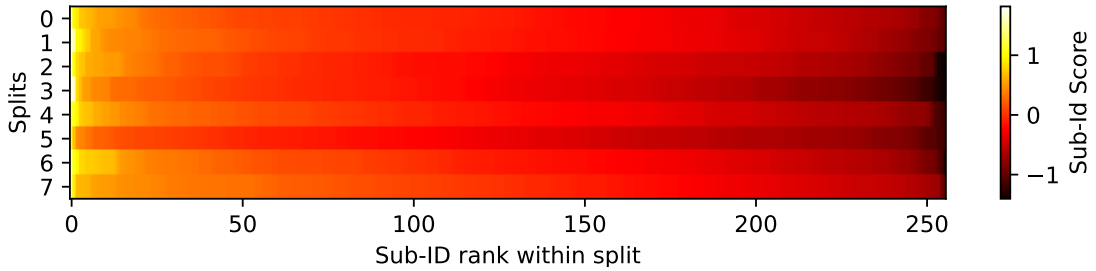
In our experiments RecJPQPrune showed improvements over both Transformer Default and PQTopK. However, from Table 7.3, we can also see that the margin between the 50th and 95th percentiles of scoring time is larger for RecJPQPrune compared to baselines. Indeed, for the baselines, the scoring complexity does not depend on the user. In contrast, for RecJPQPrune, the number of iterations depends on the user – for most users, the scoring time is low, but there are some users for which the algorithm requires many iterations.



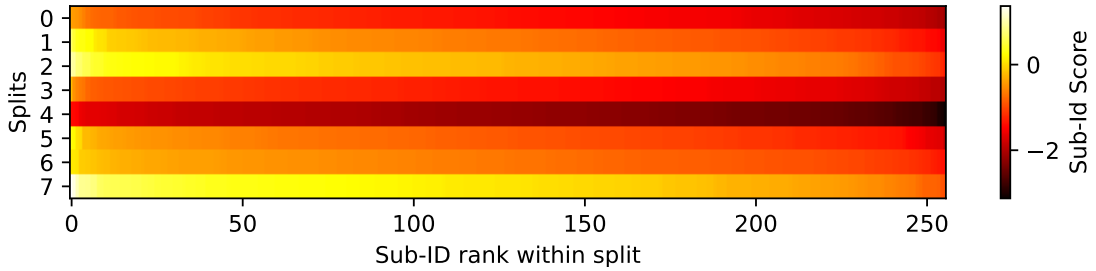
(a) Fast (1ms, 2.9% items scored, 1 iteration), gBERT4RecJPQ



(b) Average (6ms, 32% items scored, 11 iterations), gBERT4RecJPQ



(c) Average (4ms, 23% items scored, 8 iterations), SASRecJPQ



(d) Slow (91ms, 102% items scored, 35 iterations), gBERT4RecJPQ

Figure 7.6: Sub-item id scores for three users: fast (a, gBERT4RecJPQ), average (b, gBERT4RecJPQ) & (c, SASRecJPQ), and slow (d, gBERT4RecJPQ). Y-axis represent splits, x-axis represent rank of the sub-item id within split (ordered by score) and colour represents the score of the sub-item id.

To better understand what makes users more or less suitable for RecJPQPrune, we visualise sub-item id scores using on the Gowalla dataset in Figure 7.6. The figure shows the sub-item id scores for three different users for gBERT4RecJPQ: a *fast* user, for which the algorithm only requires 1ms; an *average* user, for which the algorithm needs 6ms; and a *slow* user for which the algorithm requires 91ms. To illustrate the difference between the models, we also include a visualisation of the same *average* user with the SASRecJPQ model, where scoring requires 4ms. The captions also report the percentage of items scored for these 3 users, which varies considerably, between 2.9% and 102% (recall that some items may be scored repeatedly, so the percentage of scored items can exceed 100%) implying that the *pruning difficulty* [146] of these users varies considerably. Interestingly, pruning difficulty can be inferred from inspection of Figure 7.6, as there is a clear difference between the sub-item id scores distribution for these three users. Indeed, for the fast user, there are very few sub-item ids with high scores, and all are located in the same 3rd split, so the algorithm can identify most highly scored items very rapidly (just a single iteration of the algorithm). For the average user, we see a more balanced distribution of sub-item id scores for for both gBERT4RecJPQ and SASRecJPQ, but there are still relatively few high-scored sub-item ids. For the slow user, most of the sub-item ids in the 7th and 2nd splits have high scores, forcing the algorithm to process most of the sub-item ids from these splits. Finally, considering the difference between the visualisations for the average user between gBERT4RecJPQ and SASRecJPQ, we see that highly-scored sub-ids for the latter are more concentrated to in the left side of the plot, making gSASRec more efficient for this user (4ms vs. 6ms).

Overall, these differences in scoring time highlights an important property of RecJPQPrune: *it is most efficient when the model is confident, such that the sequence emebdding only has to score a few highly scored sub-item ids before terminating*. This insight opens a path for future research: the model can be *trained* to make pruning more efficient. We postulate that this can be achieved, for example, by applying L1-regularisation on the sub-item id score distribution at training time, in order to increase sparsity (like in [121]).

7.3 Conclusions

In this chapter, we introduced PQTopK, an efficient scoring Algorithm for RecJPQ-based models. We then used PQTopK to design RecJPQPrune, a novel dynamic pruning algorithm for scoring RecJPQ-based sub-item representations within embedded sequential recommender systems. RecJPQPrune is inspired by dynamic pruning from document retrieval but tackles a completely

new and more recent problem faced by Transformer-based recommendation models asked to predict over large item catalogues. Our experiments demonstrated the efficiency benefits of the RecJPQPRune approach: for instance, on the Tmall dataset with 2.2M items, we can reduce model scoring time by 64× compared to the Transformer Default baseline, and 5.3× compared the recent PQTopK approach. This is achieved while being safe-up-to-rank-K, i.e., no impact on effectiveness up to rank K; efficiency is further enhanced as K is reduced. Overall, RecJPQ-Prune offers a substantial improvement in the deployability of Transformer-based recommendation models to industrial scale deployments with millions of items.

While this work is safe-up-to-rank-K, it is also possible to configure RecJPQPrune to be unsafe – we leave this to future work.

Finally, we believe that PQ-based implementations of Transformer models combined with RecJPQ-Prune also have applications to generative retrieval models [187] and recommender systems [177, 194], where the Transformer generates the ids to retrieve. We leave testing of our method for generative settings also to future work.

In this chapter (and, indeed, all the chapters before) we always assumed the "score-and-rank" approach for recommendation: the recommender system scores all items and then returns top K items with the highest scores. In the next chapter, we will show that, while this approach works well for optimising accuracy, it has limitations for beyond-accuracy metrics; we will also show how to overcome these limitations by using autoregressive generation instead of "score-and-rank" approach.

Chapter 8

Sequential Recommendation Models for Beyond-Accuracy Goals

In the preceeding chapters, we focused on optimising ranking accuracy in Transformer-based recommender systems. However, as we stated in Limitation L2.5 in Section 2.3.5, many real-life deployments need to be aligned with beyond-accuracy goals, such us diversity or novelty. Unfortunately, "out-of-the-box", Transformer-based recommendation models, such as SASRec and BERT4Rec are not effective for these goals. Indeed, as we show in this chapter, the traditional "score-and-rank" Top-K scoring tend to generate non-diverse recommendations due to fundamental limitations of dot-product based scoring.

Hence, in this chapter we address Limitation L2.5 by proposing a novel autoregressive Next-K recommendation strategy. The Next-K strategy generates recommendations given user history similarly to how generative language models generate text continuation given a prompt. In addition, we propose GPTRec, a generative recommendation model that uses Next-K generation. We also develop a two-stage (supervised pre-training/reinforcement-leaning based alignment) mechanism that allows to align GPTRec with any measurable goals (including complex coals, such as recommendation diversity).

This chapter is organised as follows: Section 8.1 motivates the problem of beyond-accuracy goals alignment; Section 8.2 covers related work on reinforcement learning for recommendation and adaptations of language models; Section 8.4 discusses the Top-K and Next-K recommendation generation strategies. Section 8.5 describes the GPTRec recommendation model; Section 8.6 describes the two-stage training process of GPTRec; Sections 8.7 and 8.8 contains the experimental evaluation of GPTRec; Section 8.10 contains final remarks.

The material of this chapter is based on two full research papers: (i) the paper [177] was published as a full research paper at Gen-IR workshop at the SIGIR'23 conference and (ii) the paper [175] was published as a full research paper at the GenRec workshop at the WWW'24 conference.

8.1 Beyond-Accuracy Goals in Sequential Recommender Systems

Until recently, most of the research in Sequential Recommendation was focused on optimising *recommendation accuracy* (as measured by ranking metrics, such as NDCG or Recall@K). However, many researchers argue that a modern recommender system should consider metrics beyond accuracy [59, 67]. For example, a good recommender system should provide a user with a *diverse* set of recommendations to give the user a reasonable choice [7], and should not focus too much on the popular items [15, 108, 171], because those can be easily discovered by the user even without a recommender system.

As discussed in Section 2.3, most Transformer-based recommender models, including both SASRec and BERT4Rec, adapt language model architectures to the recommendation domain by using item IDs instead of tokens. They typically generate recommendations using a score-and-rank approach, where the model identifies the most probable items to continue the sequence of interactions – analogous to how language models predict a single next token at each step. We refer to this as the *Top-K strategy*. However, this strategy is problematic due to the *SoftMax Bottleneck problem* [267]: the distribution of the scores produced by the Top-K recommendation is *uni-modal* (i.e. similar items are likely to have similar scores, see Section 8.3 for details). Hence, due to the SoftMax bottleneck, the Top-K strategy is unable to assign high scores for a diverse set of items simultaneously, as required by the beyond accuracy metrics. Indeed, the model output is likely to be dominated by similar types of items, which may be sub-optimal, and sometimes, it is better to show different types of items to a user to cover a broad set of intents that the user may have [210].

To address the limitations of the Top-K strategy and its inherent SoftMax bottleneck, we explore an alternative approach inspired by generative language models. Indeed, he rapid progress of generative language models, such as T5 [193] and the GPT family [17, 165, 191, 192] has demonstrated that these models could be successfully used for a broad class of tasks, such as text summarisation, sentiment analysis, machine translation etc. In the recommendation task, these models can be fine-tuned to directly generate item IDs as text strings by a prompt that contains the user's history [61]. Similarly, in a search task, Tay et al. showed that generative models could

replace the traditional reverse index-based retrieval and directly generate relevant document IDs by a given query. In contrast with the Top-K strategy, where the model generates its output in just one inference, generative models produce their output *autoregressively* (token-by-token) and use the information of previously generated tokens to generate the next one.

In this chapter, we propose to adapt the autoregressive text generation to Sequential Recommendation. In particular, we propose the *Next-K strategy*, where recommendations are generated autoregressively item-by-item. In that case, when the model generates a recommendation in position i, it already knows what items are recommended in position $1..i-1$, and may adjust the result accordingly. We discuss the details of Top-K and Next-K recommendation generation strategies in Section 8.4. The main challenge with the Next-K strategy is model training. Indeed, standard supervised learning algorithms do not work well for training a generative model, as they require a source of *good* recommendations (recommendations that maximise our effectiveness measure, see Section 8.6.2), which isn't typically available. For example, to train a model to generate diverse and relevant recommendations, we would need a training set of diverse and relevant recommendations for a large number of users. However, to obtain such a set, we would need a recommendation model with at least the same performance as the generative model we intend to train, which we assume is unavailable – indeed, the only available data training data in a typical recommendation scenario is historic user-item interactions. A similar problem exists in the training of language models, in that the available training data is not aligned with the desired task. However, this problem has recently been addressed by the Reinforcement Learning With Human Feedback (RLHF) approach [165]. Inspired by RLHF, we propose a reinforcement learning-based approach, where we sample recommendations from the current version of the recommendation model ("Actor"), use a separate model to estimate the quality of produced recommendations ("Critic") and then use the Critic's output to improve the main recommendation model. This approach can optimise for almost any metric measured on a full recommendation list. In particular, we apply this approach to optimise recommendation diversity and reduce popularity bias.

The goal of this chapter is to determine the feasibility of a universal training scheme for aligning a single generative recommendation model for very different recommendation goals. Note that our focus is the *universality* of the approach for optimising different metrics. Hence, to show the feasibility of the training scheme, we compare our method with the most well-studied and still widely

used re-ranking-based methods, such as Maximal Marginal Relevance (MMR) [25]. While there may exist highly specialised methods for different recommendation goals, re-ranking-based methods remain a very popular choice in real-life industrial applications, as is evident by a number of industrial recent blog posts[1].

To show that it is possible to train a generative model to produce good recommendations using the Next-K generation strategy, we propose GPTRec - a novel generative model for a Sequential Recommendation based on the GPT-2 [192] architecture. We use the pre-training/fine-tuning approach with GPTRec on two datasets, namely MovieLens-1M [71] and Steam-2M (a smaller version of the Steam [100] dataset). Our experiments show that GPTRec in generative mode matches the performance of state-of-the-art BERT4Rec when tuned for accuracy only. However, when the model is tuned for more intricate goals, such as increasing diversity or decreasing popularity bias, GPTRec provides better tradeoff between accuracy and one of these goals, compared to applying BERT4Rec with greedy re-ranking techniques such as using Maximum Marginal Relevance (MMR) [25] for improving diversity. For example, on Steam-2M, while exhibiting the same amount of diversity (measured by the Intra-List Distance (ILD) [7]), GPTRec achieves 10% higher NDCG@10 than BERT4Rec with MMR.

Following [42], for practical reasons, we limit the scope of this chapter to datasets containing only a few thousand items. While scaling Transformer-based models to large catalogues is an important and orthogonal challenge (addressed separately in Chapters 4-7 of this thesis), we note that many real-world recommendation scenarios naturally involve catalogues of limited size. Examples include single-brand online stores (e.g., apple.com), news recommendation (where only recent articles are relevant), and high-level music recommendations (such as recommending genres rather than individual tracks). Furthermore, training models with reinforcement learning—the primary approach adopted in this chapter—is computationally expensive. Given our available consumer-grade hardware, we prioritise the flexibility to perform multiple iterations and experiments quickly, aligning practical considerations with realistic application contexts.

In short, we summarise the contributions of this chapter as follows:

1. We propose a novel Next-K recommendation strategy as an alternative to traditional Top-K recommendation and demonstrate its viability by showing that GPTRec in Next-K generation mode can match the accuracy of the best Top-K models;

---

1. See for example https://langstream.ai/2023/10/04/introducing-mmr-rerank/ or https://www.vectara.com/blog/get-diverse-results-and-comprehensive-summaries-with-vectaras-mmr-reranker

2. We propose a supervised pre-training/reinforcement fine-tuning approach for training generative recommender models for complex recommendation goals;
3. We propose GPTRec, a generative Sequential Recommendation model based on the GPT-2 architecture that uses the Next-K generation strategy;
4. We show that when optimised for accuracy only, it can match state-of-the-art transformer-based models, such as BERT4Rec while generating recommendations using the Next-K technique;
5. We show that GPTRec can be optimised for complex recommendation goals, such as increased diversity and decreased popularity bias, and an optimised version of GPTRec provides a better tradeoff between accuracy and secondary metrics than state-of-the-art BERT4Rec with greedy re-ranking.

8.2 Reinforcement Learning for Recommender Systems

This section covers existing research related to this chapter: Section 8.2.1 provides a brief overview of reinforcement learning. Section 8.2.2 covers existing work on reinforcement learning for recommender systems. Section 8.2.3 covers the recent line of work of using pre-trained language models for recommendation tasks. Section 8.2.4 provides an overview of the misalignment problem in language models and how it has been addressed with the RLHF approach.

8.2.1 Basics of Reinforcement Learning

In this section, we describe the basics of reinforcement learning necessary for describing our methodology. Reinforcement Learning [233] (RL) is a class of machine learning methods in which an *agent* interacts with an *environment* by performing *actions*. The environment reacts to the actions by giving the agent *rewards* and changing the agent's *state*. The agent learns a *policy* of selecting actions depending on the state that maximises the *long-term reward* (reward accumulated over multiple steps).

We now introduce some definitions and notations used across the chapter to describe the Reinforcement Learning problem setup; Table 8.1 also lists all notations used in this chapter. Typically, a Reinforcement Learning problem is formalised using the framework of *Markov Decision Process* (MDP) [233, Ch.3]. While there are some variations in the definition of MDP in the

Table 8.1: Notations used in the chapter

Notation	Description
I	Set of items in the recommender system's catalogue
$h = [h_1, h_2, ...h_n]\,; h_i \in I$	A sequence of user-item interactions in chronological order of a given user (history)
n	Number of interactions in each sequence
H	Set of all possible user histories h
K	Generation cutoff (or ranking cutoff) for recommender system; a recommender system always returns K recommendations for a user
$g = [g_1, g_2, ...g_K]\,; g_i \in I$	A full list of recommendations generated by the recommender systems
$\mathcal{G}$	A set of all all possible recommendation lists g of length K
$g^{(i)} = [g_1, g_2, ...g_i]\,; g_j \in I$	A partially generated list of recommendations, up to & including position j
G	Set of all possible partially generated recommendations $g^{(i)}$
$R : H \times \mathcal{G} \to \mathbb{R}$	Effectivenes measure; the goal of a recommender system is to generate recommendations g that maximise R given h
$\hat{g} = [\hat{g}_1, \hat{g}_2, ...\hat{g}_K]\,; \hat{g}_i \in I$	Perfect recommendations, a list of recommendations that maximises R given h
$\mathrm{r} : H \times G \to \mathbb{R}$	Immediate reward, a portion of R associated with generating g_i
$MDP = \langle \mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma \rangle$	Markov Decision Process, a tuple that describes a Reinforcement Learning problem
$\mathcal{S} = \{s_1, s_2, ...s_n\}$	A finite set of states in an MDP
$\mathcal{A} = \{a_1, a_2, ...a_n\}$	A finite set of actions available to an agent in an MDP
$P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$	A state transition probability distribution in MDP
$r : \mathcal{S} \to \mathbb{R}$	Reward function in MDP
$\rho_0 : \mathcal{S} \to \mathbb{R}$	Distribution of initial state s_0 in an MDP
$\gamma \in [0, 1]$	Discount factor in MDP, defines the preference of immediate rewards over future rewards
$\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$	A stochastic policy, learned by an agent in reinforcement learning; the policy defines a probability of the agent taking an action given a state
$\pi_\Theta : H \times G \times I \to \mathbb{R}$	GPTRec's learnable policy, parametrised by Θ
$V_\Psi : H \times G \to \mathbb{R}$	GPTRec's value model parametrised by Ψ; represents estimated sum of future immediate rewards (discounted with γ)
δ_i	Temporal difference (TD) – defines an improvement in expected value according to value function V after selecting an action at step i
A_i	Advantages, discounted sum of future temporal differences
λ	Discount factor, used for computing advantages A_i
$\hat{f} : H \to \mathbb{R}^d$	A function used to compute sequence embeddings (d-dimensional)
$W : \mathbb{R}^{\lvert I \rvert \times d}$	A matrix of learnable item embeddings (d-dimensional)
$f : H \times I \to \mathbb{R}$	Scoring function, used in the Top-K recommendation strategy. Typically computed computed as the dot product: $f(h, i) = \hat{f}(h) \cdot W[i]$
$\phi : H \times I \times G \to \mathbb{R}$	Scoring function, used in Next-K recommendation strategy

literature, we use the definitions and notations from the TRPO paper [212] that largely influenced the Proximal Policy Algorithm [214] – the main RL algorithm we use in this chapter. In particular, a Markov Decision Process is defined as a tuple:

$$MDP = \langle \mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma \rangle \tag{8.1}$$

where S is a finite set of states, $\mathcal{A}$ is a finite set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the distribution of transition probability, $r : \mathcal{S} \to \mathbb{R}$ is the reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}$ is the distribution of initial state s_0, $\gamma \in [0, 1]$ is the discount factor that defines how much we prefer immediate award over the award at the next step.

Given an MDP, the agent-environment interaction dynamics can be described as follows:

1. The initial state s_0 is drawn from the distribution ρ_0
2. At each time step i, the agent observes the state $s_i \in \mathcal{S}$ and selects an action $a_i \in \mathcal{A}$

3. The environment draws a new state s_{i+1} from the transition probability distribution $P(s_{i+1}|s_i, a_i)$
4. The agent receives a reward $r(s_{i+1})$

The ultimate goal of the agent is this setup to maximise the long-term discounted reward:

$$r_{0:\infty} = \sum_{i=0}^{\infty} \gamma^i r_i \tag{8.2}$$

To achieve its goal, the agent follows a *stochastic policy*, which is defined as a probability distribution $\pi : \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$. When the agent observes the state s_i, it selects the next action a_i by drawing the action from the conditional distribution $\pi(a_i|s_i)$. In a typical Reinforcement Learning scenario, the agent also adjusts its policy when observing rewards and state transitions from the environment. In deep learning, the policy is typically modelled using a neural network $\pi_\Theta(a|s)$ where Θ is the set of network parameters. In that case, the policy is adjusted by changing the set of parameters Θ.

Note that in the general case, the parameters of the MDP, such as the transition probability distribution $P(s_{i+1}|a_i, s_i)$ as well as the reward function $r(s)$ are not known to the agent (however, an agent may have *some* prior knowledge about them); The agent has to *learn* the structure of these parameters through interactions with the environment and reflect this structure in the parameters of its policy.

While, generally speaking, the number of interactions that an agent performs with the environment may be infinite, in practice, in many situations, there is usually a *terminal* state, after which the MDP stops. The process then may restart again. A single sequence of state actions that end in a terminal state is known as an *episode*. The agent may use the knowledge obtained from interactions with the interactions in the previous episode to maximise the reward in the next episode.

Reinforcement Learning is an active area of research, and there are many ways of learning optimal policies. For a comprehensive overview of the problem and the methodology, we refer to the classic book [233]. In our work, we use the *Proximal Policy Optimisation* (PPO) algorithm as the main algorithm for reinforcement learning, because this algorithm has recently shown remarkable results in achieving goals similar to ours in the domain of Language Modelling. We describe the details of the PPO algorithm with application to our recommendation task in Section 8.6.3.

We now briefly describe how Reinforcement learning has been applied to Recommender Systems.

8.2.2 Reinforcment Learning for Recommender Systems

The use of RL is appealing for recommender systems because it can optimise for complex beyond-accuracy rewards, which are not necessarily differentiable (e.g. diversity), as well as optimise for long-term goals, which are hard to optimise using traditional supervised learning. Indeed, we are not the first to apply RL-based techniques for recommender systems [4, 9, 42, 226, 276, 290].

Table 8.2 summarises existing lines of research on reinforcement learning for recommender systems. As the table shows, existing approaches can be divided into two types based on what these methods use as the action space $\mathcal{A}$. We now briefly describe these existing types.

T1: Item-based methods In this type, the methods (e.g. [226, 264]) use possible items $i \in I$ as the actions and make the recommendation by sorting the items according to the probability of selecting them as the next action (i.e. using the Top-K strategy, see Section 8.4.1). However, as we argue in Section 5, it is hard to optimise the models that use the Top-K strategy for complex metrics, such as diversity, because these methods compute items' scores independently from each other.

T2: Slate-based methods Methods (e.g. [42, 232]) in this type use all possible *slates* (full lists of recommended items $g \in \mathcal{G}$) as the action space and generates the slate using a single model inference. The problem with this group of approaches is that the space of all possible recommendation lists is very large (proportional to $|I|^K$, where $|I|$ is the number of possible items, and K is the size of the slate). It is computationally infeasible to score all possible slates to select the best one. Therefore, slate-based models usually require an existing source of high-quality slates that they can use as training data, which is not always available, or rely on some other heuristics, which limit the search space. Both of these groups of methods use the historical user-item interactions as the agent's state, using which the agent selects actions.

Table 8.2: Existing applications of Reinforcement Learning for Recommender Systems. *In Sequential Recommender models, users are represented by the sequences of their historical interactions $h \in H$

Type	Example methods	States Space $\mathcal{S}$	Actions Space $\mathcal{A}$	Comment
T1: Item-based	[226, 264]	Users $h \in H$*	Items $v \in I$	These methods score items independently from each other; therefore it is hard to optimise complex effectiveness measures R that include interdependent components, such as diversity.
T2: Slate-based	[42, 232]	Users $h \in H$*	Slates $g \in \mathcal{G}$	Typically, these methods require a source of high-quality slates (e.g., simulator), due to the very large size of action space.
T3: Generative	GPTRec (ours)	Users + Partially generated recommendations $\langle h \in H; g^{(i)} \in G \rangle$*	Items $v \in I$	Focuses on aligning with a given effectiveness measure R. Does not require a simulator and access to high-quality slates.

T3. Generative The method proposed in this chapter does not fall into either of the existing types and therefore, we put it in a separate group. Indeed, similar to the methods from the slate-based method, it focuses on optimising listwise metrics; however, similar to the methods from the first group, the actions in our method correspond to items. We achieve this by changing the state space $\mathcal{S}$: in our case, the state includes not only historical user-item interactions but also a partially generated recommendations list. It allows the method to take into account items already recommended to the user, as well as plan for future actions ahead. In contrast to existing slate-based methods, our approach does not require existing sources of high-quality slates. Instead, it uses a 2-stage approach, where, in the first stage, the model learns to mimic the behaviour of the teacher model, and in the second stage, it aligns the model with the given effectiveness measure R.

Note that the application of reinforcement learning for recommender systems is an actively developing topic, and instead of providing a comprehensive survey of the applications of RL methods for recommender systems, we refer the reader seeking for the most recent developments to one of the recent survey papers [4, 138]. However, to the best of our knowledge, most existing methods fall into either T1 or T2 categories, and this chapter is the first to apply reinforcement learning for aligning autoregressive generative models with beyond-accuracy goals, as represented by T3.

We also specifically highlight the SMORL work by Stamenkovic et al. [226] as one of the most related to ours. Similar to our work, the work uses a form of reinforcement learning to optimise for beyond the accuracy goals, such as diversity and novelty. According to a recent comprehensive survey on diversification methods in search and recommendation [256], SMORL is the only Reinforcement Learning-based method that is directly comparable to ours (e.g. the method focuses on the Sequential Recommendation problem; it specifically optimises the beyond-accuracy goals; it does not require access to online recommendation systems and users). However, SMORL uses sequence as state and items as action space (i.e. it is type T2 according to our classification) and a traditional Top-K "score-and-rank" recommendation strategy, which, as we argue in

Section 8.3, suffers from the fundamental Sotmax Bottleneck problem. In contrast, our method (i.e. type T3, generative) uses a sequence + a partially generated recommendation list as a state. Interestingly, the SMORL paper compares their method with prior state-of-the-art Sequential Recommendation method optimised with Reinforcement Learning and finds that without special alignment, prior RL-based methods, such as SQN [264], tend to underperform in beyond-accuracy metrics compared to traditional models trained with supervised learning.

8.2.3 Recommendations as Text Generation

The arrival of Large Language Models, such as GPT-3 [17] and LLaMA [243], has shown that the pre-trained text generation models may serve as universal problem solvers and can be applied to a large class of tasks. This was specifically shown for recommendation in recent works. For example, the P5 model [61] (based on the pre-trained T5 language model [193]) uses text generation to generate item and user IDs directly as text strings. M6-Rec [39] (based on the pre-trained M6 language model [136]) directly generates item titles as recommendations. While using pre-trained models for recommendation is an interesting research direction, it differs from ours. Indeed, these models rely on pre-trained models, which encapsulate knowledge about the world (including the knowledge about the recommended domain) and, therefore, can be seen as recommender systems with side information.

In contrast, our proposed GPTRec does not rely on any side information and uses a more classical Sequential Recommendation setting where the only data available to the model is the sequences of item interactions made by users. This allows us to understand the properties of the generative approach better and decouple it from the benefits of the availability of side information. Note that while GPTRec uses the GPT-2 architecture as a backbone, it does not use GPT-2's pre-trained weights.

Although GPTRec does not directly use pre-trained weights of its backbone GPT-2 model, it still shares many similarities with GPT. In particular, similarly to language models, there is a *misalignment* problem: the data available for training usually is not aligned well with the desired recommendation objectives. In the next section, we briefly discuss how the misalignment problem has recently been addressed in language models with the help of Reinforcement Learning.

8.2.4 The Misalignment Problem and Reinforcement Learning with Human Feedback

One of the big problems of state-of-the-art generative language models is the problem of *misalignment* between training data and the actual desired outcome. A typical desired use-case for generative models is to work in a "chat-bot" mode and follow the instructions of the user. However, the training data for such interactive communication and instruction-following is limited and very expensive to gather. In contrast, "regular" text data is abundant and practically unlimited in the modern days: texts can be crawled from websites, books, public datasets, etc. Earlier generative models, such as GPT-2 [192] and T5 [193], were trained only using "regular" training data and therefore had limited use for building interactive applications.

Notably, Oyang et al. [165] proposed a solution for the misalignment problem using the idea of "Reinforcement Learning with Human Feedback" (RLHF). The authors proposed a three-step solution for the problem:

1. Train a language model for the next token prediction task using a "regular" training dataset and fine-tune it using a small amount of available "high-quality" dialogue data that was generated with human labellers.
2. Use the model from step (1) to generate different possible outputs to the same model input and ask the labellers to compare generated outputs. These labels are then used to train a *reward* model, which essentially associates a numerical value with the generated output: the larger the reward, the better the output, according to the model.
3. Use a reinforcement learning approach from step (1) to fine-tune the language model using the Proximal Policy Optimisation [214] algorithm.

We argue that a similar misalignment problem arises in the recommender systems when the recommendation objectives include beyond-accuracy components, such as diversity. Indeed, Sequential Recommendation datasets typically contain a large number of user-item interaction sequences that can be used for training Sequential Recommendation models, such as BERT4Rec or SASRec. Indeed, as the main goal of these models is to predict the next item in the sequence of interactions, the training data is *aligned*, with the target goal of accurately predicting the next item, and these models can achieve state-of-the-art results when trained appropriately using different training objectives, such as sequence shifting, item masking or recency sampling of sequences (see also Chapter 4 for a comprehensive study of training objectives). However, the datasets typically do not contain "perfect" model outputs for beyond-accuracy goals, such as diversity, and, therefore, supervised training for such goals is challenging due to data scarcity.

Usually, there is no source of high-quality training data for beyond-accuracy metrics; the typical solution involves training a model for accuracy (for which training data is abundant) and then re-rank using a heuristic approach, such as MMR [25] or Serendipity Oriented Greedy [116]. While these heuristics somewhat allow for the mitigation of the misalignment problem, there is no evidence that such heuristics lead to a solution that may be close to optimal. Therefore, inspired by the success of the RLHF for language models, we propose to apply a similar approach to recommender systems. The difference with the RLHF of our approach is that we assume that the target quality metric is known and measurable (in our experiments, we use a linear combination between accuracy (measured by NDCG) and a secondary objective, such as diversity (for example, measured by Intra-List-Distance [7]). This simplifies our approach, as we do not need human labellers to generate high-quality recommendations and compare generated recommendations. Instead, we directly use a known quality measure as a reward and train the model to maximise it using reinforcement learning.

8.3 Embedding Similarity and Softmax Bottleneck

A typical modern Sequential Recommender System, such as BERT4Rec or SASRec, computes item scores by first computing an embedding representation of a sequence and then computing scores using a similarity function between an item embedding and the sequence embedding. Formally, a score for an item v is computed as:

$$score[v] = \text{sim}(\hat{f}(h), W[v]) \tag{8.3}$$

where $\hat{f}$ is a sequential neural model that produces an embedding given a sequence of interaction (e.g., Transformer or Recurrent Neural Network), W is a learnable matrix of item embeddings, and sim is a similarity function. The most common choices for the similarity function include dot product [100, 223] or cosine similarity [70, 248]. Intuitively, similarity-based approaches try to put item embeddings and sequence embeddings in the same space. The items returned as recommendations are the items located close to the sequence embedding in the embedding space. Using this approach, the item scores are generated independently from each other, allowing for a simple "score-and-rank" Top-K recommendation strategy (see Section 8.4.1). Nonetheless, by design, this approach only looks for items in the neighbourhood of a sequence embedding in the embedding space, which only works for simple recommendation goals. However, these similarity-based approaches struggle to generate recommendations that should include many different items that are not similar to each other.

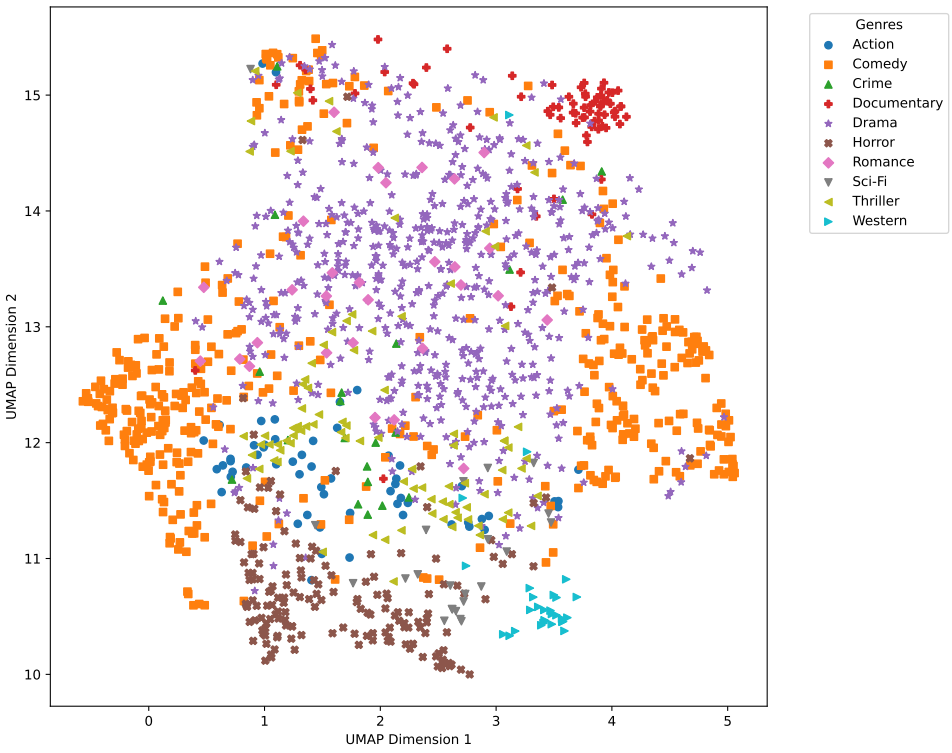


Figure 8.1: UMAP projection [155] of SASRec's item embeddings; MovieLens-1M dataset. No location in this space is close to Horror, Western and Documentary genres simultaneously.

To illustrate this problem, Figure 8.1 visualises item embeddings learned by a version of SASRec on the MovieLens-1M dataset. We project the item embedding into 2-dimensional space using a UMAP [155] transformation. We then visualise item embeddings from different genres using different colours and different markers. As we can see from the figure, different movie genres tend to occupy different regions in the embedding space (for example, Documentary movies are located in the top right corner and Western movies are located in the bottom left part of the plot). To recommend movies from a specific genre, a recommender system can generate an embedding that is located in the middle of one of the regions (or pointing in the same direction if the similarity function is based on dot product or cosine similarity). However, imagine that a hypothetical user of the system who, depending on their mood, prefers different movie genres, for example, Horror, Documentary and Western. Hence, as the system does not know what exact mood the user has right now, a "perfect" recommendation list for the user should contain at least one Western movie, at least one Documentary movie and at least one Horror movie. As can be seen from the figure, there is no location in the embedding space corresponding to this mixture of genres, and produced recommendations are unlikely to contain this mixture of genres.

The problem that the recommendations generated using Equation (8.3) (using the Top-K strategy) cannot represent complex inter-dependent item sets is known as the *SoftMax Bottleneck [267]*. The name SoftMax bottleneck is related to the SoftMax transformation, which is used to convert raw scores into probabilities in many Deep-Learning-based models. The problem arises from the fact that the distribution of the scores obtained from Equation (8.3) is uni-model, i.e. all highly-scored items are located in the proximity (or pointing in the single direction) as a single location in the embedding space.

Recently, Chang et al. [26] showed that the SoftMax bottleneck problem arises in a Sequential Recommendation when the interaction sequences contain repeated interactions; however, we argue that the same problem arises, for example, when modelling diverse recommendation. Indeed, as we have shown before, when our goal is to produce diverse recommendations covering a broad range of users' interests, we need to build a multi-modal score distribution covering multiple areas. Therefore, to build complex recommendation lists with item sets containing items not necessarily similar to each other, we need to design an alternative strategy for producing recommendations rather than just use "score-and-rank" Top-K recommendations. In the next section, we discuss the scoring strategies in more detail and describe the Next-K generation as an alternative to the Top-K recommendation.

8.4 Top-K and Next-K recommendations

In this section, we discuss the classical Top-K recommendation approach (used by BERT4Rec and SASRec) and its limitations and propose a novel alternative Next-K approach for generative models.

8.4.1 Top-K strategy

The ultimate goal of a recommender system is to provide a user with a list of items that are likely to be of user interest. The most typical approach for solving this is the *Top-K* recommendation strategy [122, 287]. In this strategy, the recommendation model $f(h, v) \to s$ returns an estimated relevance score $s \in \mathbb{R}$ for each interaction sequence $h \in H$ and item $v \in I$. Typically, in the modern Sequential Recommender methods, such as SASRec [100] and BERT4Rec [230], the score is estimated as a dot product or cosine similarity between sequence representation embedding and the item embedding, as described by Equation (8.3).

Algorithm 5 illustrates the strategy in the form of pseudo-code. To generate the recommendation for a particular sequence h, the recommender system uses a model f to calculate all item relevance scores and then selects K items with the highest scores as the recommendations. Note that Algorithm 5 can be vectorised very efficiently. Indeed, the iterations of the main loop (lines 3-5 in the Algorithm) are independent of each other and can be computed in parallel. Top-K selection (line 6 in the algorithm) is also a vectorisation-friendly operation, and popular vectorisation frameworks provide efficient implementations for it, for example `tf.math.top_k` in TensorFlow or `torch.topk` in PyTorch.

Algorithm 5 Top-K Recommendation Strategy

Input: h is the sequence of user-item interactions
Input: I is the set of all available items
Input: K is the number of recommendations to generate
Input: $f(h, v) \to \mathbb{R}$ is a scoring function

1: **procedure** TopKRecommend(h, I, K, f)
2: $scores \leftarrow$ empty array of scores for all items in V, initialised to 0
3: **for** $v \in I$ **do**
4: $scores[v] \leftarrow f(h, v)$
5: **end for**
6: $g = \texttt{TopK}(scores, K)$ ▷ TopK returns K items with highest scores, ordered by score in descending order
7: **return** g
8: **end procedure**

While achieving state-of-the-art results for accuracy-based recommendations, the Top-K strategy may fail for beyond-accuracy goals due to the fundamental SoftMax bottleneck problem described in Section 8.3. For example, the Top-K strategy may generate redundant recommendations: if a user has recently bought a coffee machine, they will likely buy coffee beans as their next purchase. However, there can be many different variants of coffee beans. Each variant will likely have a high recommendation score, so most of the recommended items will be coffee

beans. This leads to redundant recommendations (it is enough to recommend only 1-2 types of coffee) and poor representation of the user's other interests [285]. Diversification re-ranking methods, such as MMR [25], and multi-aspect methods [21] can address redundancy and diversity, but they pose challenges such as increased complexity, computational demands, and difficulty in training. Additionally, these approaches may require extensive hyperparameter tuning to achieve optimal performance [215]. Instead of re-ranking approaches, training a single model with an interdependent objective can provide a more efficient solution.

Other problems that Top-K recommender systems will likely fail to solve include complementary item recommendations (e.g., we may want to recommend fashion items that fit well with each other) and serendipity (sometimes we want to help users discover new items). In the next section, we introduce the Next-K recommendation strategy, which can mitigate these problems.

8.4.2 Next-K strategy

Next-K is an autoregressive recommendation strategy, where the recommender system decides what to recommend at position i only after generating recommendations at position $1..i-1$. More formally, the Next-K recommendation strategy uses the scoring function $\phi(h, g^{(i)}, v) \rightarrow s$, which depends on historical interactions $h \in H$, partially generated recommendation list $g^{(i)}$ and an item $v \in I$, and returns relevancy score $s \in R$. Algorithm 6 shows the strategy in the form of pseudo-code. It starts with an empty recommendation list (line 2 in Algorithm 6), then iteratively selects the next item with the highest score according to scoring function ϕ (line 4), and adds them to the partially generated recommendations list (line 5). The iterations continue until all K recommended items are generated.

Algorithm 6 Next-K Recommendation Strategy

Input: h is sequence of user-item interactions
Input: I is the set of all available items
Input: K is the number of recommendations to generate
Input: $\phi(h, g^{(i)}, v)$ is an existing scoring model

```
1: procedure NextKRecommend(h, I, K, φ)
2:     g ← empty list                ▷ g is the list of already generated recommendations
3:     for i = 1 to K do
4:         g_i ← arg max_{v∈I} φ(h, g, v)
5:         g.append(g_i)
6:     end for
7:     return g
8: end procedure
```

The Next-K recommendation strategy is equivalent to greedy autoregressive generation in generative language models. Indeed, in our experiments, we were able to use the Next-K generation both by implementing Algorithm 6 from scratch and by using the `.generate()` API call in the Hugging Face Transformers [255] library.

A limitation of the Next-K strategy is that it is more computationally expensive: it requires the generation of the full scores distribution K times for a user, whereas the Top-K strategy needs only one inference per user. However, autoregressive generation is actively used in language models of much larger size compared to typical recommendation models, and these models still remain interactive; therefore, we argue that the Next-K generation is still a viable strategy for recommender systems.

Another problem is that the Next-K recommendation strategy requires special model training. While it is possible to use Next-K generation with traditional models, such as SASRec, by adding the generated item to the end of the input sequence, that would lead to degradation of the model quality. Indeed, in Sequential Recommendation models, items positioned closer to the end of the input sequence have the highest impact on the predictions (see Section 4.5.6). Hence, if a recommender system makes an incorrect recommendation at position i, the error will be propagated to positions $i+1, i+2, ..., K$, causing error accumulation at higher ranking cutoffs K. In Section 8.8.2, we empirically show the effectiveness degradation at higher cutoffs K of the traditional models, such as SASRec and BERT4Rec, when they are used with the Next-K strategy compared with the more traditional Top-K recommendation. In contrast, the teacher-student training technique, described in Section 8.6.2 allows our proposed GPTRec to avoid this degradation and show effectiveness similar to the Top-K models across different ranking cutoffs K.

In summary, in this section, we described Top-K and Next-K recommendation strategies. We identified that Top-K suits accuracy-based effectiveness measures well, but its application for beyond-accuracy measures, such as diversity, is limited, as it scores items independently from each other. In contrast, we proposed that the Next-K strategy generates recommendations item-by-item and, therefore, can account for intra-list item dependencies. However, the Next-K strategy requires a special training scheme.

We now describe GPTRec, a generative recommendation model specifically designed for recommendation strategy. We then describe the training scheme that we use to achieve high effectiveness for both accuracy-based and beyond-accuracy training measures.

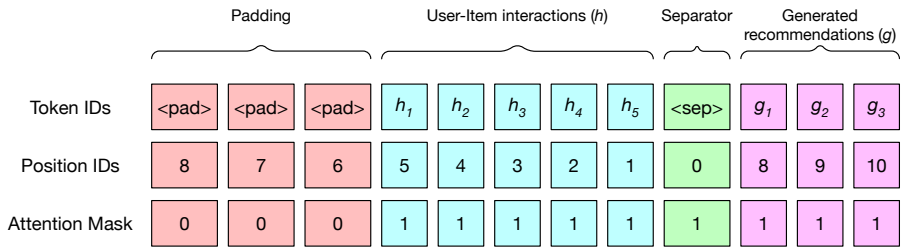


Figure 8.2: GPTRec's sequence structure.

8.5 GPTRec

In this section, we introduce GPTRec, a generative Sequential Recommendation model we use as a backbone for our experiments.

8.5.1 Architecture

GPTRec,[2] designed for the generative Sequential Recommendation utilises the GPT-2 architecture [192], which in turn is based on the Decoder part of the Transformer model [247]. There are minor modifications to the original Transformer model in GPT-2, such as moving the layer normalisation to the beginning of the transformer block, implementing a modified initialisation with scaled residual weights, and employing learnable positional encodings instead of sine-based encodings. For brevity, we omit the details of the Transformer model and refer readers to the original publications [192, 247]. Finally, note that while GPTRec's architecture is similar to SASRec, which also uses a Transformer Decoder as a backbone, SASRec uses the standard Top-K strategy, while GPTRec uses Next-K.

8.5.2 Sequence Structure

Similar to the original GPT-2 model, GPTRec generates recommendations autoregressively using the Next-K recommendations strategy. This means model outputs at the i-th recommendations stage become model inputs at the $i+1$-th stage (alongside the original user-item interaction history). GPTRec's input sequence consists of 4 parts:

2. GPT stands for Generative Pre-trained Transformer. In our adaptation, we don't use the pre-trained versions of GPT, but we saved the letter P in the model name to give credit to the GPT authors.

1. optional padding to equalise lengths of all sequences;
2. user-item historical interactions $h = \{h_1, h_2..h_n\}; h_i \in I$;
3. separator symbol that signals that the "history" part has ended;
4. generated recommendations. $g = \{g_1, g_2..g_k\}; g_i \in I$

Each position in the sequence consists of three integer numbers: (1) a token ID (the item ID, or one of two special tokens for padding and separator); (2) a position ID used by the Transformer architecture to preserve positional information; and (3) an attention mask, which tells GPTRec the positions in the sequence it can ignore.

Initially, the sequence consists of the user history h followed by a separator token; then, at each next generation step i, the sequence shifts left, and a recommended item r_i is added to the end of the sequence. Because GPTRec shifts the same sequence multiple times during generation, the same position may have different semantics. For example, the second last position belongs to the "history" part at the first generation step, after the first sequence shift separator moves to the second to last place, and at the third step, it is occupied by the first recommended item. Therefore, to help the model distinguish between different parts of the sequence, we adopt the following position ID scheme:

- The separator token always has position ID 0;
- The position IDs of historical user-item interactions are in reverse chronological order. This way, the last action (which is the most important for recommendation, see Section 4.3.4) always has position ID 1; the second last has position ID 2, etc.;
- The position IDs associated with recommended items are in ascending order starting from position ID $n + 1$ where n is the maximum length of the user-item interactions history; i.e. first recommended item has position ID $n + 1$, second item $n + 2$ etc. This way, each position in the recommendation is associated with a specific position ID that does not change during the generating process;
- The model ignores item padding tokens, so their position IDs can be anything.

This concludes the description of the GPTRec model. We now discuss its training using a 2-stage pre-training/fine-tuning approach.

8.6 Training GPTRec

As discussed in Section 8.4.2, training the GPTRec model for Next-K generation that is aligned with the beyond-accuracy goals is a challenging task, as training high-quality training samples are rarely available. In this section, we describe a two-stage process that is capable of addressing this challenge. To do this, we first describe the training objectives in Section 8.6.1. We then describe the 2-stage pre-training/fine-tuning approach we use to train the model in Section 8.6.2. Finally, we describe an efficient process decomposition for the reinforcement learning fine-tuning in Section 8.6.4.

8.6.1 Training Objectives

In this section, we formalise the optimisation problem for training GPTRec. Consider a sequence of user-item interactions $h = \{h_1, h_2..h_n\}; h_i \in I$. The goal of the recommender system is to *generate* a list of recommendations $g = \{g_1, g_2...g_K\}; r_i \in I$, which maximises an *effectiveness measure* $R(h, g)$. R may have a complex structure and depend not only on accuracy but also on item popularity, diversity, etc. Note that R depends on the whole recommendation list, and therefore, it can not be optimised using a pointwise loss functions, such as Binary Cross-Entropy. It also is not required to be differentiable, limiting our ability to optimise R using gradient-based methods. Therefore, instead of relying on standard supervised learning, we propose to optimise it using a reinforcement learning approach, which we describe in Section 8.6.2.

Without loss of generality, we can say that metric M can be decomposed into the sum of *immediate rewards* (a part of overall effectiveness measure R, which can be calculated after generating each individual item in the recommendation list):

$$R(h, g) = \sum_{i=1}^{K} \mathrm{r}(h, g_1, g_2, ...g_i) = \sum_{i=1}^{K} \mathrm{r}(h, g^{(i)}) \tag{8.4}$$

where $g^{(i)}$ denotes a partially generated list of recommendations up to position i: $g^{(i)} = \{g_1, g_2, ..g_i\}$.

Even if the metric is only meaningful over the whole list of items, we can say that the immediate reward is 0 for each step, except for the last item in the list; this is also known as *delayed reward*:

$$\mathrm{r}(h, g^{(i)}) = \begin{cases} 0; i \in 1..K-1 \\ R(h, g); i = K \end{cases} \tag{8.5}$$

However, in many cases, the immediate reward may have a non-zero value even for a partially generated list. For example, if accuracy is part of the overall metric, the immediate reward may be positive when the model generates a relevant item; for decreasing popularity bias, the immediate reward may be negative when generating a popular item. Optimising the model for immediate reward is easier, as it requires the model to do less planning, which is generally a challenging task [158]. We discuss examples of possible metrics for R and their decompositions into immediate rewards in Section 8.7. We now discuss how GPTRec can be trained to optimise any given effectiveness metric R.

8.6.2 Pre-Train/Fine Tune approach

When GPTRec is used with the Next-K strategy, it outputs recommendations item-by-item. Consider a partially generated list of recommendations $g^{(i)}$ for the sequence h. As GPTRec is a probabilistic model, it estimates the conditional probability of an item appearing next in the recommendation list:

$$\pi_\Theta(h, g^{(i)}, g_{i+1}) \approx P(g_{i+1}|g^{(i)}; h) \tag{8.6}$$

where Θ is the set of learnable model parameters. Here, we use the symbol π for the GPTRec model itself to highlight that this is used as a policy model in the Reinforcement Learning setup. To understand which item to put to the position $i+1$, the model obtains the conditional probability distribution π_Θ and selects the item with the maximum probability:

$$g_{i+1} = \arg\max_{g_{i+1} \in I} \pi_\Theta(h, g^{(i)}, g_{i+1}) \tag{8.7}$$

Consider that we have a training set containing "perfect" recommendations $\hat{g} = \{\hat{g}_1, \hat{g}_2, \hat{g}_3, ..\hat{g}_k\}$ for the sequences of historical interactions h (the lists of recommendations, which maximise a given effectiveness measure $R(h, g)$ given h:

$$\hat{g} = \arg\max_{g} R(h, g) \tag{8.8}$$

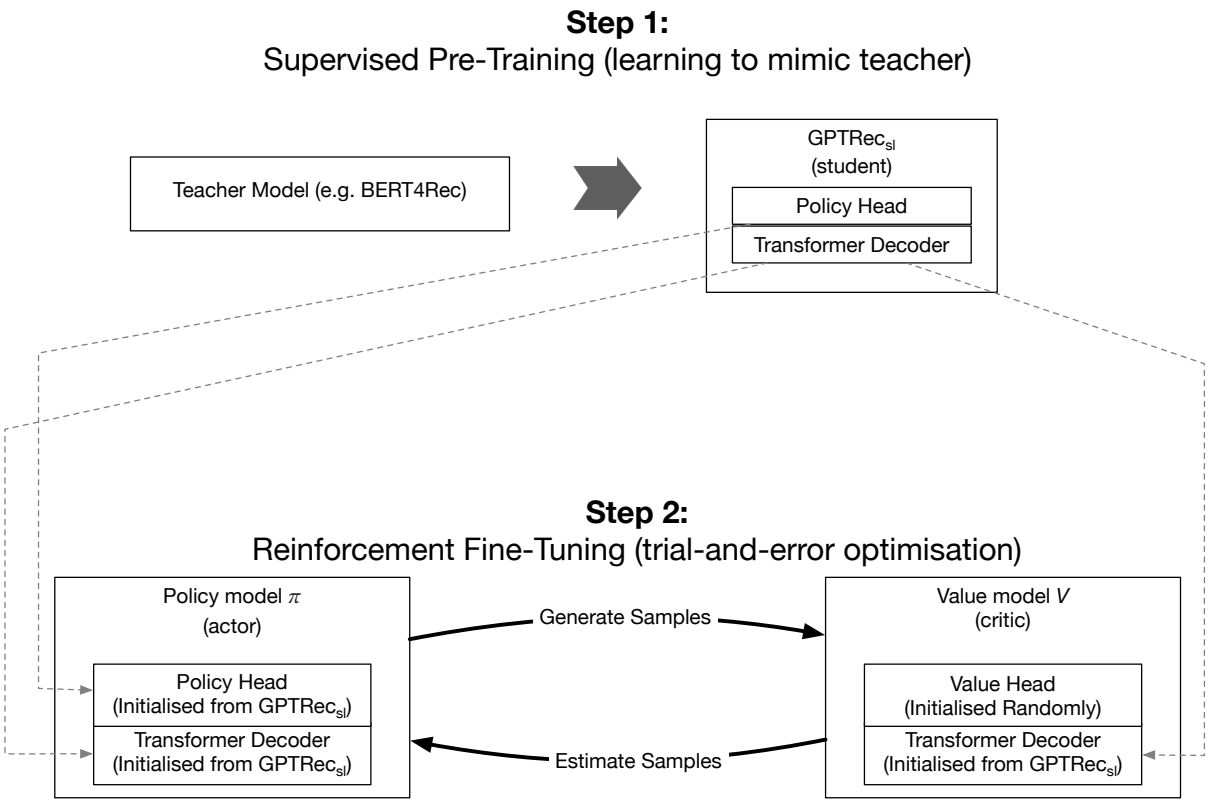


Figure 8.3: GPTRec's Pre-Training/Fine-Tuning scheme. Pre-training (Step 1) takes the form of using a Top-K model like BERT4Rec (teacher) to pre-train a GPTRec model checkpoint (student). In Step 2 (Fine-tuning), the Policy model π is the GTPRec model itself initialised by the student model checkpoint from Stage 1; the Value model is a Transformer Decoder-based model with a regression head. The Transformer Decoder layer of the Value model is initialised from the Transformer Decoder layer of the student model, and the regression head is initialised randomly.

Then, GPTRec can be optimised using the standard Language Modelling (LM) loss (the same way, as GPT [192] is optimised given a large dataset of texts):

$$\mathcal{L}_{LM}(\Theta) = -\sum_{i \in 1..k} \log \pi_\Theta(h, \hat{g}^{(i-1)}, \hat{g}_i) \tag{8.9}$$

Unfortunately, the "perfect" list of recommendations, $\hat{g}$, is usually unknown. Indeed, given the complex structure of many possible quality metrics for R, in the general case, we have to score every possible recommendation list to find the perfect list g according to Equation (8.8), which is not feasible due to the combinatorial size of the set of all possible recommendation lists.

To solve this problem and inspired by the success of InstructGPT [165] for solving a similar problem in the language processing domain, we propose a two-stage pre-training/fine-tuning approach. Figure 8.3 illustrates the overall idea of the two-stage approach. As the figure shows, at the first stage, we use supervised pre-training to train the model to mimic the behaviour of

a traditional Top-K recommendation model (a *teacher* model), such as BERT4Rec [230]. To do that, we first train the teacher using the training set; then, for each training sequence h_j, we generate a list of teacher recommendations $\tilde{g}_j$; we then use $\tilde{g}_j$ as the "perfect" recommendation $\hat{g}_j$ in Equation (8.9) and optimise GPTRec using the LM loss. This process drives the model to generate recommendations that are as similar as possible to the ones generated by the teacher model[3]. The problem is that this learning goal is not necessarily aligned with optimising the target metric R. Indeed, most traditional Top-K models are trained to optimise accuracy. At the same time, R may include other components, such as diversity; the model pre-trained this way is likely to be sub-optimal with respect to R.

Therefore, in the second stage, as illustrated in Figure 8.3, we use a reinforcement learning-based approach to align GPTRec with the target metric R. To achieve this, following the InstructGPT paper [165], we use the Proximal Policy Optimisation (PPO) algorithm [214]. PPO is an Actor-Critic type of approach [233, Ch. 15.1], which jointly trains the main *policy model* π ("Actor") and an auxiliary *value model* V ("Critic"). Similar to other reinforcement learning methods, PPO is a trial-and-error approach, where the Actor iteratively performs actions (generates recommendations) with the goal of beating the expectations of the Critic, and the Critic tries to predict the performance of the Actor in each case.

In our case, we use GPTRec itself in Figure 8.3) itself as the policy model π and another GPTRec-based value model as the value model V. The difference between the policy and value models is that the policy generates a probability distribution over all possible items in the catalogue, whereas the value model generates a single number: a predicted recommendation effectiveness according to metric R.

Despite having different outputs, the input to both models has the same structure (illustrated in Figure 8.2). Therefore, in both cases, we use the same GPT-2-based Transformer Decoder as the model backbone; however, in the value model, we use a simple feed-forward layer to predict a single number instead of a standard GPT-2 token prediction head. Following best practices [221], we initialise the policy model π from the supervised checkpoint trained in the first stage ($GPTRec_{sl}$). We also initialise the backbone Transformer Decoder of the value model V

3. The process of teaching one model to mimic the behaviour of another model is also often called *distillation* [82] in the literature. However, the main goal of distillation is usually to make the student model smaller and more efficient compared to the teacher model. Our goal is different: we want the student model to mimic the behaviour of the teacher model but do it in an autoregressive, generative way regardless of the models' sizes. Therefore, we call this the teacher process "teacher-student pre-training" to avoid confusion.

model from the backbone Transformer Decoder of the same supervised pre-trained model. We then fine-tune both policy and value models using the PPO algorithm; we provide a high-level overview of the PPO algorithm in the next section and Section 8.6.3 and refer to the original PPO paper [214] for more details.

8.6.3 Details of the Proximal Policy Optimisation (PPO) Algorithm

Proximal Policy Optimisation (PPO) [214] is a state-of-the-art algorithm for many Reinforcement Learning problems that have been shown effective for aligning generative recommendation models with complex goals [165]. While the algorithm is well-known in the Reinforcement Learning community, it is rarely used in recommender systems literature[4]. As PPO is an integral part of our methodology, we provide an overview of the algorithm here. As we describe in Section 8.2.1, the goal of Reinforcement Learning is to train an *agent* (a recommender system in our case) to perform *actions* (choosing an item to recommend at position i) given a *state* (a user with history h and a partially generated list $g^{(i)} = [g_1, g_2, ..g_{i-1}]$) to optimise the long-term *reward* (metric R over the fully generated recommendation list g). PPO assumes that the agent follows a stochastic *policy* π_Θ parametrised by a set of parameters Θ: the agent selects actions from a probability distribution over the set of all possible actions, and PPO iteratively improves the policy by updating its parameters Θ.

In our case, the policy defined by the GPTRec model itself defines the policy π_Θ, and therefore PPO iteratively updates GPTRec's parameters to maximise effectiveness R.

PPO follows the Actor-Critic optimisation scheme: it uses an auxiliary model, V_Ψ, parametrised by a set of parameters Ψ, which predicts the expected discounted long-term reward given the current state. The idea is to increase the probabilities of actions that increase our expected reward at each training iteration and decrease the probabilities of actions that decrease our expected reward. Therefore, in addition to training the main policy model, we train the value model:

$$V_\Psi(h, g^{(i)}) \approx \mathbb{E}_{\pi_\Theta}\left[\sum_{j \in 0..K-i-1} \gamma^j \mathrm{r}(h; g^{(i+j)})\right] \tag{8.10}$$

4. Notable exceptions where PPO had been used for Recsys include: Padhye et al. [166] use PPO for non-Sequential Recommendations, Feng et al. [54] use PPO for explainable knowledge graph-based recommendations, Chen et al. [28] use PPO for online learning. In all these cases, the PPO application is different from ours (i.e. non-sequential models, not used for aligning generative models with beyond-accuracy goals).

here, $0 < \gamma \leq 1$ is the discount hyperparameter, which controls how much the model prefers immediate reward over delayed reward, and $\mathbb{E}_{\pi_\Theta}$ means expectation over all possible generated recommendations with the GPTRec model. To optimise the value model, PPO uses a standard mean squared loss:

$$\mathcal{L}_{MSE}(\Psi) = -\sum_{i=1}^{K}\left(V_\Psi(h, g^{(i)}) - \sum_{j=0}^{K-i-1}\gamma^j \mathrm{r}(h; g^{(i+j)})\right)^2 \tag{8.11}$$

PPO uses the value function to compute *advantages*, which measures how much better or worse our model selected an item at each generation step compared to the expected value (according to metric Q and expected reward V). PPO uses the *Generalised Advantage Estimators* (GAE) [213]. To compute the GAE advantages, we first compute the temporal difference (TD) residual. The TD residual, denoted as δ_i, at position i is given by:

$$\delta_i = \mathrm{r}(h, g^{(i+1)}) + \gamma V_\Psi(h, g^{(i+1)}) - V_\Psi(h, g^{(i)}) \tag{8.12}$$

where $\mathrm{r}(h, g^{(i+1)})$ is the immediate reward at position $i+1$ (Equation (8.4)), and γ is the same discount factor as used in Equation (8.10). The GAE advantage fo position i, denoted as A_i, computes the advantage at each position i using a decay factor λ (which is also a hyperparameter that similarly to γ controls the model focus on long/short term rewards) and TD residuals, is then by:

$$A_i = \sum_{l=0}^{K-i-1}(\gamma\lambda)^l \delta_{i+l} \tag{8.13}$$

To drive the model towards better rewards, PPO optimises the surrogate objective, which depends on the Ratio$(\cdot)$ function:

$$\text{Ratio}(\Theta, h, g^{(i)}) = \frac{\pi_\Theta(h, g^{(i-1)}, g_i)}{\pi_{\Theta_{old}}(h, g^{(i-1)}, g_i)} \tag{8.14}$$

where Θ_{old} is the set of model parameters used at the time of generating the recommendations list $g^{(i)}$. Note that PPO uses the same generated recommendations for multiple optimisation steps, and therefore, Θ_{old} is not equal to Θ, and overall, the proportion in Equation (8.14) is not equal to 1. The PPO surrogate objective (also known as the CLIP objective) is then defined as:

$$\mathcal{L}_{CLIP}(\Theta) = -\sum_{i=1}^{K}\min\left(\text{Ratio}(\Theta, h, g^{(i)})A_i, \text{clip}\left(\text{Ratio}(\Theta, h, g^{(i)}), 1-\epsilon, 1+\epsilon\right)A_i\right) \tag{8.15}$$

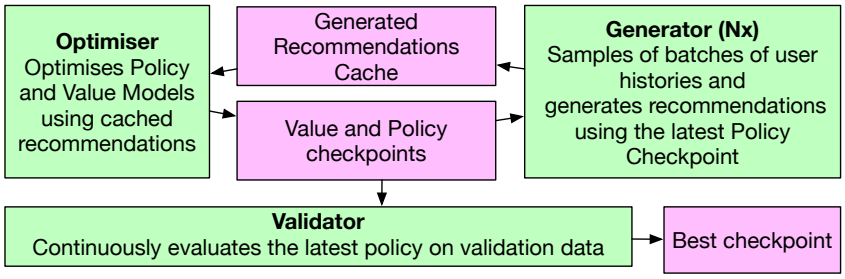


Figure 8.4: GTPRec fine-tuning processes diagram. Green boxes are processes; purple boxes are data.

where the clip$(\cdot)$ function limits the minimum and maximum of the first argument, and ϵ is a hyperparameter that controls maximum divergence from old model parameters Θ_{old}. Without clipping, the Ratio function (Equation (8.14)) is known to diverge from 1 very quickly; therefore, clipping is required to prevent an explosion of gradients of the CLIP objective.

Overall, the loss function optimised by PPO can be written as:

$$\mathcal{L}_{PPO}(\Psi, \Theta) = \mathcal{L}_{MSE}(\Psi) + \mathcal{L}_{CLIP}(\Theta) \tag{8.16}$$

The original PPO paper also includes an entropy bonus in the loss function (which encourages the model to perform exploration). However, in our initial experiments, we observed no improvement in the overall quality of GPTRec when using an entropy bonus; therefore, we omit it for training GPTRec. Overall, PPO can be summarised as repeating the following steps:

1. Sample a batch of user histories $h_1, h_2 ... h_B$; $h_i \in H$
2. For each history h in the batch, generate recommendations g using the current version of GPTRec π_Θ. Note that at this step, we generate recommendations stochastically, drawing items from the distribution stochastically rather than selecting the item maximum score;
3. For each pair history-recommendations pair $\langle h, g \rangle$, compute advantages $A_1, A_2 \ldots, A_K$ using Equation (8.13);
4. Perform multiple optimisation steps with respect to value function parameters Ψ and GPTRec parameters Θ, using gradient descent with $\mathcal{L}_{PPO}$ loss (Equation (8.16)).

This concludes the overview of the PPO algorithm for GPTRec optimisation. For more details, we refer to the original PPO and GAE papers [213, 214]. In the next section, we discuss how we implement PPO in practice to allow efficient tuning of GPTRec using limited hardware.

8.6.4 Efficient Asynchronous Decomposition of Reinforcement Fine-Tuning

As we discussed in Section 8.6.2, the PPO algorithm alternates between generating recommendations using the current generation of the policy model and improving the policy and value models using gradient descent. Improving the policy and the value models using gradient descent can be efficiently parallelised using hardware acceleration on a GPU and a deep learning framework (we use Tensorflow [1])[5]; however, generating recommendations with the Next-K strategy is an iterative process (recommendation for position $i + 1$ can be only generated after generating the recommendation for position i). This iterative nature of generation limits the efficiency of using the GPU for generation. In initial experiments, we observed that directly alternating between generation and optimisation steps, the GPU becomes under-utilised, and model training becomes too slow to be feasible in practice (in our experiments, fine-tuning a single mode could take up to a week). However, with PPO, we can make generation and optimisation asynchronous. Therefore, to efficiently utilise available resources, we decompose sample generation into a separate group of processes and perform the generation using the Next-K strategy on the CPU. We also use a separate validation process, which continuously evaluates the latest model checkpoint on a validation dataset. The validation results are used to monitor metrics during fine-tuning as well as in order to select the final model checkpoint.

Figure 8.4 summarises the processes involved in fine-tuning GPTRec. As can be seen from the figure, to train GPTRec, we use 3 processes:

1. *A Generator* process that continuously samples batches of user histories from the training set and generates recommendations for these histories using the latest version of the policy model. After generating the recommendation, it saves the batch of $\langle h, g \rangle$ pairs into the generated recommendations cache. The recommendation cache stores the last M generated batches, where M is a hyperparameter. Multiple Generators can work simultaneously.
2. *An Optimiser* that continuously samples batches of $\langle h, g \rangle$ pairs from the recommendations cache and performs optimisation steps for policy and value models using these batches. It then saves checkpoints of the Value and Policy models so the Generator and Validator processes can use them.
3. *A Validator* that continuously takes the latest checkpoint of the Policy model and evaluates it using the validation dataset.

---

5. The code for this chapter: `https://github.com/asash/gptrec_rl`

This scheme allows us to use both CPU and GPU system resources efficiently and reduces the training time of a single model from a few days to a few hours.

8.6.5 Comparison of GPTRec's and SASRec's Training and Inference

As we noted in Section 8.5.1, GPTRec is not the first recommender model based on the Transformer Decoder architecture. Indeed, SASRec's [100] architecture is similar, and the differences in architectures between the two are unimportant. Indeed, the main differences between SASRec and GPTRec are not in the architecture but in their training and inference: SASRec is a Top-K recommendation model, and its training scheme is specifically designed for the Top-K recommendation strategy. As we show in Section 8.8.2, SASRec's performance degrades when it uses the Next-K Strategy.

In contrast, GPTRec uses the Next-K strategy and further uses special techniques (namely, the teacher-student pre-training scheme and a specially designed sequence structure) that allow it to improve ranking quality at deep ranking cutoffs compared to Top-K. Indeed, as argued in Section 3.2, the Next-K recommendation strategy allows the model to account for intra-list recommendation dependencies when generating a recommendations list. Moreover, the reinforcement learning-based fine-tuning allows the model to utilise such dependencies and optimise for complex recommendation objectives, such as diversity.

We also note that the sequence structure and the 2-stage training scheme are not specific to the GPT-2 backbone. Indeed, as mentioned before, we do not use GPT's pre-trained weights; therefore, the training and inference can be used with any Transformer Decoder-based architecture, including SASRec, or other language model architectures, such as LLaMA [243]. However, reinforcement-learning-based optimisation is computationally expensive and does not allow us to perform exhaustive architecture searches. Therefore, we use one of the best and most studied generative model architectures, leaving an exhaustive architecture search for generative recommendations for future work.

With this, we conclude the description of GPTRec and its training procedure. We now turn to the experimental evaluation of the model.

8.7 Experimental Setup

We aim to answer the following research questions:

GPTRec Research Questions

RQ8.1: How effective is GPTRec trained as a supervised student compared to state-of-the-art baselines?

RQ8.2: What is the effect of the teacher-student pretraining scheme on the effectiveness of the Next-K strategy at different ranking cutoffs K?

RQ8.3: What is the effect of RL-based tuning of GPTRec with Next-K generation when optimising for NDCG?

RQ8.4: How does RL-based tuning affect beyond-accuracy metrics compared to greedy reranking?

8.7.1 Implementation

We implement GPTRec using the GPT-2 architecture from HuggingFace Transformers [255] and the aprec framework. We follow our common experimental setup described in Section 2.4.

8.7.2 Datasets

As discussed in Section 8.1, the scope of this work is limited to the datasets with a few thousand items in the catalogue. Indeed, there are solutions to allow Transformer models to scale to large catalogues, for instance, by using multiple tokens to represent each item (*sub-item representation*)[6].

Hence, we use the MovieLens-1M and Steam-2M datasets for our experiments, which both have < 4000 items. We pre-process the datasets as described in Section 2.4.1.

6. Specifically, see our early experiments in our workshop paper [177], where we already showed that GPTRec could be used with sub-item representations. Adding sub-item representations is a further experimental variable outside the scope of this chapter.

Table 8.3: Evaluation results. The best results are highlighted in bold; second best are underlined; * denotes significant difference with BERT4Rec ($pvalue < 0.05$, Bonferroni multi-test correction), arrows indicate the metric improvement direction

(a) MovieLens-1M

Model Type	Model	Secondary Metric	Secondary metric weight (λ)	Recall@1 ↑	Recall@10 ↑	NDCG@10 ↑	Diversity (ILD@10) ↑	Popularity Bias (nPCOUNT@10) ↓
Baselines	Popularity			0.0056$^<$	0.0363$^<$	0.0178$^<$	0.3222$^>$	1.0000$^>$
	MF-BPR			0.0113$^<$	0.0719$^<$	0.0366$^<$	0.2745	0.2867$^<$
	SASRec			0.0464$^<$	0.2482$^<$	0.1306$^<$	0.2703$^<$	0.2389$^<$
	GPTRec-Shifting			0.0603	0.2647$^<$	0.1486$^<$	0.2749	0.2985$^<$
	BERT4Rec			0.0608	**0.2921**	0.1617	0.2746	0.3222
Supervised	GPTRec-Supervised (MC Teacher)			0.0791$^>$	0.2690$^<$	0.1638	0.2798$^>$	0.4470$^>$
NDCG Tuned	GPTRec-Reinforcement-NDCG			**0.0829**$^>$	0.2785	**0.1682**	0.2878$^>$	0.4205$^>$
Diversity tuned	GPTRec-Reinforcement-Diversity-0.2	ILD	0.2	0.0798$^>$	0.2298$^<$	0.1499$^<$	0.3621$^>$	0.5212$^>$
	GPTRec-Reinforcement-Diversity-1.0	ILD	1.0	0.0795$^>$	0.1748$^<$	0.1272$^<$	**0.4349**$^>$	0.5016$^>$
Tuned to decrease popularity bias	GPTRec-Reinforcement-pCOUNT-3.0	PCOUNT	3.0	0.0820$^>$	0.2733$^<$	0.1669	0.2774	0.3936$^>$
	GPTRec-Reinforcement-pCOUNT-6.0	PCOUNT	6.0	0.0710	0.2154$^<$	0.1392$^<$	0.2648$^<$	**0.1742**$^<$

(b) Steam-2M

Model Type	Model	Secondary Metric	Secondary metric weight (λ)	Recall@1 ↑	Recall@10 ↑	NDCG@10 ↑	Diversity (ILD@10) ↑	Popularity Bias (nPCOUNT@10) ↓
Baselines	Popularity			0.0113$^<$	0.0725$^<$	0.0368$^<$	0.3282$^>$	1.0000$^>$
	MF-BPR			0.0103$^<$	0.0570$^<$	0.0294$^<$	0.2923$^<$	0.1093$^<$
	SASRec			0.0197$^<$	0.1139$^<$	0.0588$^<$	0.3135	0.4826$^<$
	GPTRec-Shifting			0.0329	0.1719	0.0912	0.3114$^<$	0.4301$^<$
	BERT4Rec			0.0331	**0.1745**	**0.0923**	0.3147	0.4973
Supervised	GPTRec-Supervised (MC Teacher)			0.0187$^<$	0.1060$^<$	0.0546$^<$	0.3210$^>$	0.8790$^>$
	GPTRec-Supervised (BERT4Rec Teacher)			0.0326	0.1699	0.0908	0.3149	0.5041$^>$
NDCG Tuned	GPTRec-Reinforcement-NDCG			0.0328	0.1709	0.0912	0.3148	0.5029$^>$
Diversity tuned	GPTRec-Reinforcement-Diversity-1.0	ILD	1.0	0.0325	0.1530$^<$	0.0837$^<$	0.3636$^>$	0.4825$^<$
	GPTRec-Reinforcement-Diversity-3.0	ILD	3.0	0.0303	0.1174$^<$	0.0699$^<$	**0.4205**$^>$	0.4744$^<$
Tuned to decrease popularity bias	GPTRec-Reinforcement-pCOUNT-3.0	PCOUNT	3.0	**0.0346**	0.1606$^<$	0.0878$^<$	0.3141	0.3355$^<$
	GPTRec-Reinforcement-pCOUNT-6.0	PCOUNT	6.0	0.0245$^<$	0.0964$^<$	0.0563$^<$	0.3182$^>$	**0.1049**$^<$

8.7.3 Pre-Training/Fine-Tuning configuration

At the supervised pre-training stage, we use a 200-epoch early stopping mechanism to ensure model convergence. At the fine-tuning stage, we tune the model for 64000 steps. We select the best model according to the target metric Q on validation data.

8.7.4 Effectiveness Metrics

We use NDCG@10 as the main accuracy metric of the models. We use PCOUNT [15] (mean recommended item popularity) as the popularity bias metric. To make PCOUNT independent of the dataset, we also use nPCOUNT — a normalised version of the PCOUNT metric, in which we divide the PCOUNT metric by the PCOUNT metric of the popularity models. This way, the item popularity model (which has maximum possible popularity bias) always has an nPCOUNT value of 1, and the smaller values nPCOUNT of nPCOUNT correspond to the smaller popularity bias of the model (i.e., lower is better). Additionally, we use the Intra-List Distance (ILD) as the metric of the diversity of the recommended items [7]. ILD is measured as a sum of pair-wise distances of

the recommended items, and as the distance metric, we use the cosine distance between movie genre vectors for MovieLens-1M and game genres for Steam-2M. As the immediate reward $q(h; g^{(i)})$ for NDCG@10 we use discounted relevance ($\frac{y_i}{\log(i+1)}$), where y_i is the ground truth relevance or i^{th} recommendation g_i); for PCOUNT as an immediate reward we use the negative item popularity and for diversity as an immediate reward we use the sum of pairwise diversities $\frac{1}{k \cdot (k-1)} \sum_{j \in 1..i-1} \text{CosDist}(g_i, g_j)$. For each immediate reward, the sum over all recommendation positions returns the value of the full metric. For PCOUNT, the immediate reward is negative, as we want to minimise the metric instead of maximising it.

8.7.5 Baselines

The goal of this chapter is to determine the feasibility of the universal alignment approach of the generative Next-K strategy with different recommendation goals, including increasing diversity and decreasing popularity bias. Note that we do not aim to establish state-of-the-art in each of these individual recommendation goals. Hence, in our baseline selection, we focus on the most popular and well-studied models rather than the most recent models that are less studied and frequently have reproducibility issues [55, 78, 174].

In particular, as the baseline models, we use two simple models: Item Popularity and BPR [199] (we use a version of BPR from the LightFM library). Additionally, we use several Transformer-based models[7]:

1. SASRec and BERT4rec, key Transformer-based models used across this paper; (SASRec is Transformer Decoder-based and BERT4Rec is Transformer Encoder-based);
2. GPTRec-Shifting is another Transformer Decoder-based baseline. Similar to SASRec, it is trained using the sequence shifting objective, but similarly to BERT4Rec, it uses the Softmax Cross-Entropy the loss function instead of the Binary Cross-Entropy. Indeed, Softmax Cross-Entropy has been shown to be more effective for Sequential Recommendation(Section 5.6). Similar to GPTRec, GPTRec-Shifting uses GPT-2 architecture as the backbone. The motivation for us to include GPTRec-Shifting as a baseline is to decouple the effect of model architecture change from the training scheme change when comparing GPTRec with the traditional models, such as SASRec and BERT4Rec.

---

7. In Section 8.2.2 we also highlight SMORL [226] as a closely related work. However, our initial attempts to use SMORL as a baseline failed: the code is not publicly available, and the paper omits some of the important training details. For example, the SMORL paper [226] mentions that the training batches are obtained from "experience buffer", and this is the only place in the paper where "experience buffer" is mentioned in the paper.

Note that in addition to using "vanilla" versions of the models that are only optimised for accuracy, we also use BERT4Rec (as the strongest accuracy baseline) with MMR Reranking to optimise diversity and with Greedy reranking to decrease popularity bias. These models are dubbed as "BERT4Rec-MMR" and "BERT4Rec-Greedy" in our experiments.

8.7.6 Teacher Models for Supervised Learning.

For the first stage of the training process, we use two teacher models. First, we use a simple Markov Chain-based (MC) model, which calculates the empirical probability of item a appearing in the sequence after item b (immediately or after several steps). It then computes the final score of the weighted sum of empirical probabilities: $s(r_i) = \sum_{j \in 1..n} \beta^j \log p(r_i | h_{n-j+1})$, where β is a decay hyperparameter, which we set to 0.6 (we find this to work the best experimentally). This teacher model is very simple, and training it only takes a few seconds on both datasets. Second, for the Steam-2M dataset, where using Markov Chain as a teacher resulted in weaker performance compared to the baselines, we also use BERT4Rec as a teacher.

8.8 Results

We now report our experimental findings for RQs 1-4 in turn.

8.8.1 RQ8.1 Supervised-Only training

We first analyse the effectiveness of GPTRec when it is trained only using the first stage of the training process, without reinforcement learning. Table 8.3 contains the results of this comparison (see model types "Supervised" and "Baselines") for (a) MovieLens-1M and (b) Steam-2M. As we can see from Table 8.3(a), for MovieLens-1M, the model with this simple teacher performs surprisingly well, achieving similar accuracy metrics to BERT4Rec (NDCG@10 0.1638 for GPTRec, 0.1617 for BERT4Rec, difference not significant). Interestingly, on this dataset GPTRec also achieves much higher Recall@1 compared to BERT4Rec (0.0791, +30.1%). We speculate that the improvement is due to a better alignment of GPTRec's training task with the next item prediction task; (see Section 5.6) for similar findings. On the other hand, on the

Steam-2M dataset (Table 8.3(b)), GPTRec with the Markov Chain teacher has a lower effectiveness than BERT4Rec (NDCG@10 0.0546 vs. 0.0923, significant); however, it is still comparable with SASRec, which has a slightly higher NDCG@10 of 0.0588. Due to the weaker effectiveness of GPTRec compared to BERT4Rec on the Steam-2M dataset, we additionally train GPTRec using BERT4Rec as a teacher. In this case, GPTRec achieves a similar NDCG@10 to BERT4Rec (1.6% difference, not significant). Hence, for steam dataset in all following experiments we use the version of GPTRec pre-trained with the BERT4Rec teacher.

Overall, in answer to RQ8.1, we say that GPTRec can achieve similar accuracy (measured by NDCG@10) to the state-of-the-art BERT4Rec model and outperforms other baselines. Importantly, it achieves good effectiveness using the *generative* Next-K strategy, meaning that the models from the first training stage can serve as a good starting point for the fine-tuning at the second stage.

8.8.2 RQ8.2 Effect of Teacher-Student pertaining on the Next-K recommendation.

In Section 8.4.2, we argued that the Next-K strategy can be used with any Sequential Recommendation model, by iteratively adding a generated item to the end of the input sequence. However, we hypothesise the model needs to be trained specifically for Next-K generation – i.e., using a Top-K model checkpoint leads to effectiveness degradation at large cutoffs K for Next-K generation. To empirically validate the hypothesis, we compare the NDCG@K metric at different cutoffs K when using the Top-K and the Next-K inference. We analyse the three baseline Transformer-based models (SASRec, BERT4Rec and GTPRec-Shifting) trained using their classical training objectives and GPTRec trained with the teacher-student scheme. We compare these baseline models using both Top-K and Next-K generation inference strategies. For all four models, we compare the same model checkpoint under different inference strategies; the only change is switching from Top-K inference (Algorithm 5) to Next-K inference (Algorithm 6).

Figure 8.5 shows the results of this analysis on both experimental datasets. As we can see from the figure, in all cases at cutoff $K = 1$, both Top-K and Next-K strategies always achieve the same NDCG@1. Indeed, when we are only interested in one item, both strategies select the item with the maximum score according to the model, and hence both strategies are equal in case $K = 1$. However, at deeper ranking cutoffs, we observe quality effectiveness degradation when using the Next-K strategy with the baseline models. For example, with $K = 10$, SASRec shows

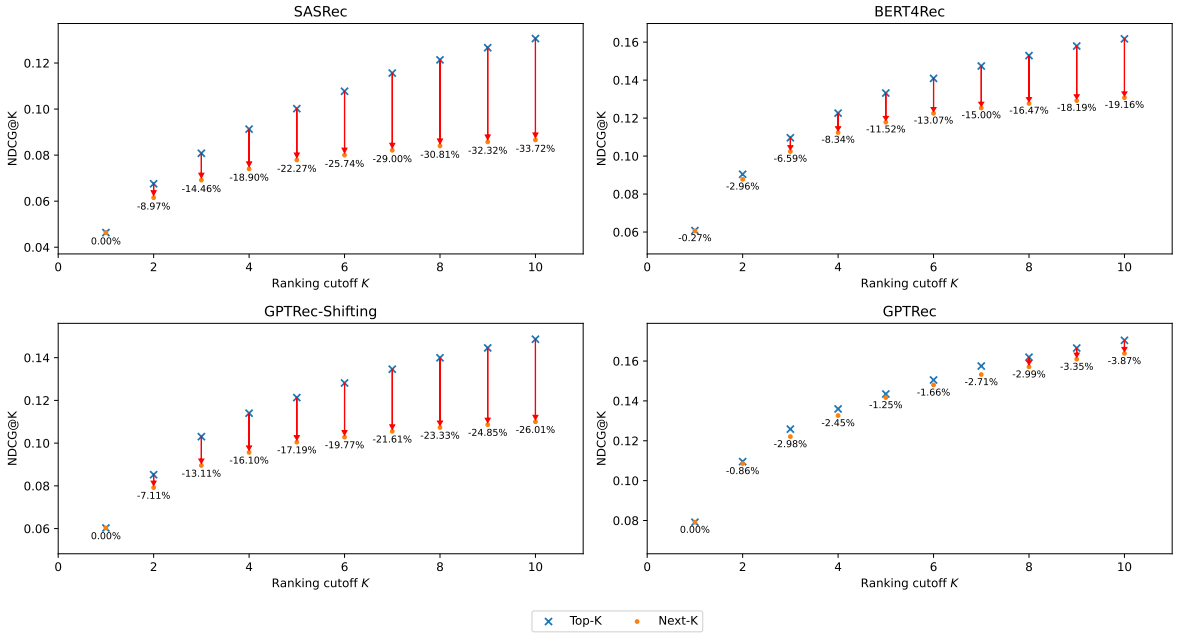
33.7% lower NDCG@10 when using the Next-K strategy compared to the Top-K. In contrast, for GPTRec pre-trained with the teacher-student, the difference between Top-K and Next-K is usually small and sometimes positive. For example, when using ranking cutoff $K = 8$, GPTRec with Next-K strategy achieves a slightly higher NDCG@8 (+0.35%) on the Steam-2M dataset while achieving slightly lower NDCG@8 on MovieLens-1M dataset (-2.99%).

When comparing the plots of the GPTRec with GPTRec-Shifting (which uses the same backbone architecture but classic sequence shifting training), we observe markedly different trends. Indeed, on both datasets, similarly to other baselines, GPTRec-Shifting exhibits effectiveness degradations when switching from the Top-K to the Next-K inference strategy. However, for GPTRec, Top-K and Next-K exhibit widely similar effectiveness across K values. For example, we observe a -16.14% NDCG@8 drop when switching from Top-K to Next-K at ranking cutoff $K = 8$ on Steam-2M, which has a similar magnitude to the other baseline models (-15.59% for SASRec, -18.93% for BERT4Rec). In contrast, for the same dataset and at the same cutoff $K = 8$, GPTRec exhibits a +0.35% improvement. This highlights that the absence of effectiveness degradation at deeper cutoffs K when using the Next-K strategy compared to Top-K in GPTRec is caused by the training scheme from sequence shifting to the teacher-student pre-training and not by the architecture (GPTRec and GPTRec-Shifting share the backbone architecture but perform differently).

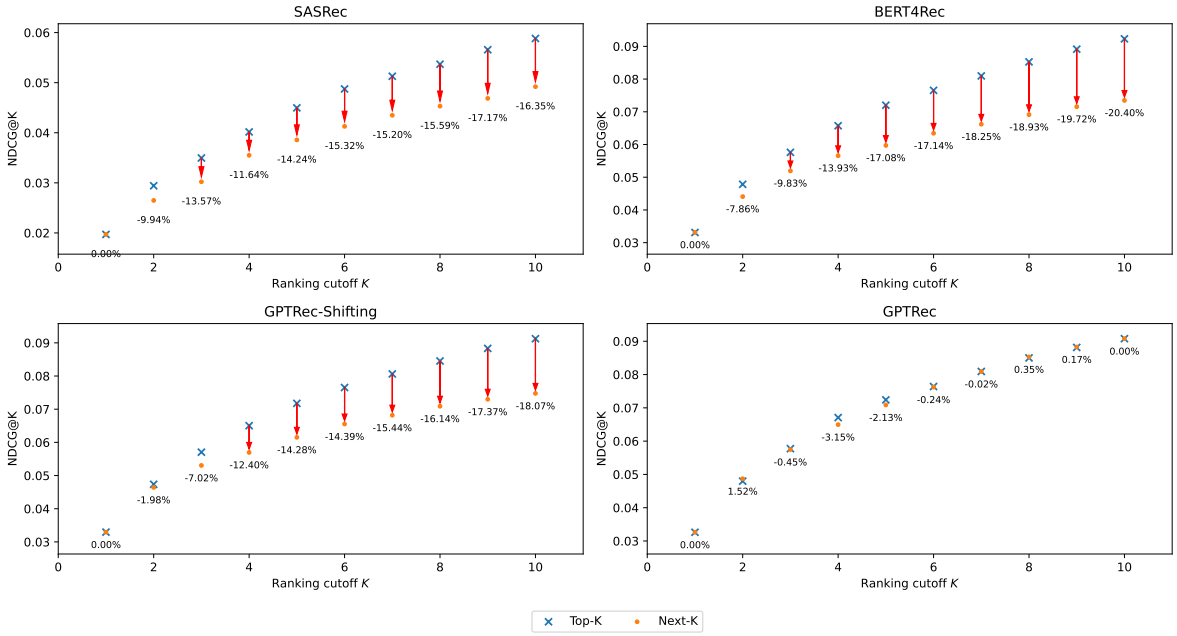
Overall, in answer to RQ8.2, we conclude that a teacher-student training scheme is necessary to achieve the same effectiveness, using the Next-K strategy, as it is possible to achieve with the Top-K generation. This shows that Next-K is a viable strategy for recommendation, but requires an appropriate training scheme. In the next sections, we empirically demonstrate its benefits compared Top-K for accuracy and beyond-accuracy metrics as a result of applying reinforcement learning.

8.8.3 RQ8.3. Reinforcement Learning for Accuracy

We now analyse the effectiveness of the full 2-stage training scheme when our goal is to optimise accuracy, i.e., using NDCG@10 as the effectiveness measure Q. For MovieLens-1M, we start from the checkpoint which was trained using the Markov Chain teacher, and for Steam-2M, we use the checkpoint trained using the BERT4Rec teacher; both these checkpoints achieve competitive results to BERT4Rec even without fine-tuning. Table 8.3 summarises the results (see "NDCG-Tuned" Model Type). As we can see from the table, fine-tuning increases NDCG@10



(a) MovieLens-1M



(b) Steam-2M

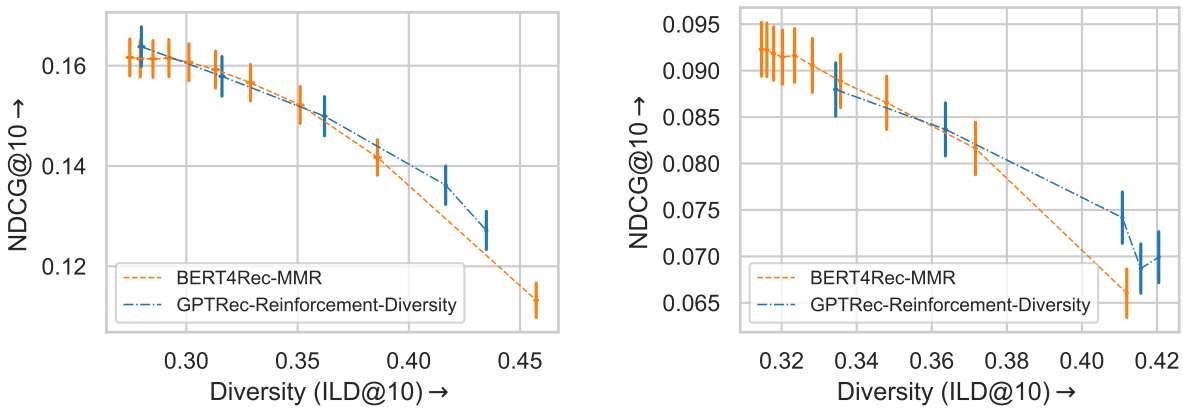
Figure 8.5: Models' NDCG@K with Top-K and Next-K recommendation strategies when varying ranking cutoff K. Red arrows demonstrate the effectiveness gap between the Top-K and the Next-K inference strategies at a given ranking cutoff K.

in both cases, but this improvement is very small (and not statistically significant). For example, on MovieLens-1M, NDCG@10 improved from 0.1638 to 0.1682 (+2.6%, not significant), and on Steam-2M, it improved from 0.0908 to 0.0912 (+0.4%, not significant). In both cases, the result is on par with BERT4Rec (statistically indistinguishable). In summary, for RQ8.3, we observe a small positive effect of reinforcement learning tuning for NDCG, but this effect is small and not statistically significant. In practice, it is enough to use only first-stage training if the goal is to optimise the model for accuracy only (but not for more complex goals; see the next section).

8.8.4 RQ8.4. Reinforcement Learning for Beyond-Accuracy Measures

In our last research question, we analyse the effect of the 2nd-stage tuning when the effectiveness metric R includes additional components. In our experiments, we use the compositional effectiveness metric $R = NDCG + \lambda \cdot R_{Secondary}$, where $R_{Secondary}$ measures diversity (using ILD) or popularity bias (using negative PCOUNT). In our experiments, we vary λ between 0 and 3 for ILD and between 0 and 6 for PCOUNT. As the baseline for secondary metric optimisation, we use greedy reranking techniques over the BERT4Rec results: Maximal Marginal Relevance (MMR) [25] as the diversity baseline, and greedy re-ranking [103] for decreasing popularity bias. In both cases, we control the tradeoff between primary and secondary metrics in the baselines by varying the weight of the secondary metric during the re-ranking stage.

Figures 8.6 and 8.7 illustrate the results of our experiments. As can be seen from the figures, the results achieved by GPTRec are not worse compared to MMR for diversity and compared to greedy re-ranking for PCOUNT. Indeed, in all cases, when placing most optimisation emphasis on NDCG, the results are not distinguishable between BERT4Rec and GPTRec (we already discussed that in RQ2). However, when we increase the importance of the secondary metrics, GPTRec outperforms greedy techniques over BERT4Rec in 3 out of 4 cases, with the only exception being tuning for decreasing popularity bias on Steam-2M. For example, when GPTRec is tuned to decrease popularity bias on MovieLens-1M, it achieves NDCG@10 of 0.139 and nPCOUNT of 0.1742. At the same time, BERT4Rec can only achieve lower NDCG@10 of 0.1267 (-8.8 %, significant), with higher popularity bias (nPCOUNT 0.1893, +8.6%, significant).



(a) MovieLens-1M **(b)** Steam-2M

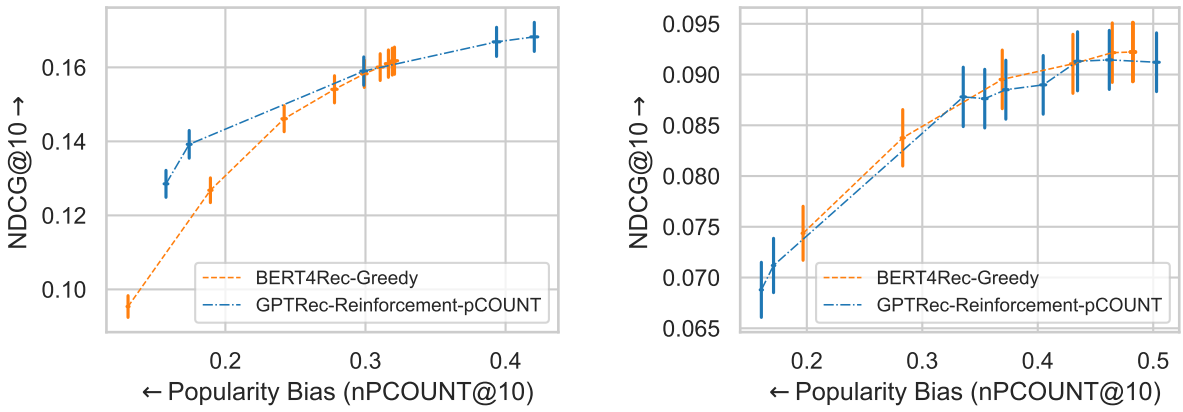
Figure 8.6: Accuracy (NDCG@10) / Diversity(ILD@10) tradeoff. Arrows represent the direction of metric improvement, and horizontal and vertical lines represent standard errors.

We also provide the quantitative results of tuning the models for beyond-accuracy metrics in Table 8.3. As can be seen from the table, when the model is tuned for a specific secondary metric, the model is able to achieve good results according to this metric while retaining reasonably high NDCG@10. For example, when we tune GPTRec for diversity on the Steam-2M dataset (Table 8.3(b)), with the weight of secondary metric $\lambda = 6.0$, we obtain the best ILD (0.4205, +33% over BERT4Rec, significant) while still retaining NDCG@10 comparable to SASRec.

Overall, in answer to RQ8.4, we conclude that fine-tuning allows us to optimise GPTRec with the Next-K generation strategy for complex recommendation goals, and the results are better (in $\frac{3}{4}$ cases) or statistically indistinguishable ($\frac{1}{4}$ case) compared to greedy reranking of the BERT4Rec results.

8.9 Discussion and future work

In this chapter, we analysed the feasibility of a universal approach for aligning generative Sequential Recommender systems for accuracy and beyond-accuracy metrics. We found that an autoregressive model can match the performance of strong classic baselines, such as BERT4Rec or SASRec. We also find that when the goals include beyond-accuracy goals (e.g. diversity), the generative Next-K strategy coupled with the Reinforcement Learning-based alignment offers a



(a) MovieLens-1M **(b)** Steam-2M

Figure 8.7: Accuracy (NDCG@10) / Popularity Bias (nPCOUNT@10) tradeoff. Arrows represent the direction of metric improvement, and horizontal and vertical lines represent standard errors.

universal and promising approach that outperforms popular greedy reranking methods, such as MMR. While we believe that the results of this work can be directly used in particular scenarios in production systems (e.g. a small-size e-commerce store), in this section, we also want to suggest several future directions for developing our method.

8.9.1 Scaling to Large Datasets

In this work, due to practical considerations, we focused on datasets with a relatively small number of items. While there are scenarios where such an approach may be directly used in practice, scaling the approach to larger datasets is an important future direction; however, given the success of the Large Language Models that use a similar alignment pipeline, we believe that it is possible to scale the described approach to larger dataset sizes. In particular, *item tokenisation* (i.e. splitting items into several sub-item tokens, similarly as words are split into sub-word tokens in LLMs) offers a path forward for scaling. Splitting items into sub-items is an important and big topic in Recommender Systems, and we would like to highlight our own RecJPQ method (Chapter 6) that uses collaborative information for item tokenisation as well as Google's semantic IDs [194, 222] that are based on semantic information.

8.9.2 Investigating architectures beyond GPT-2.

In this work, we focused on GPT-2 as a representative, one of the most cited and well-studied generative transformer architectures. However, we note that the field of generative models is developing at an unprecedented speed, and new architectures and models are published almost every day. State-of-the-art models language models include several improvements compared to GPT-2, such as rotary embeddings [229] for better position encoding and sparse transformer layers [30] for processing longer sequences. Given that these improvements have allowed better results in generative models, it is quite possible that the same improvements may work for generative recommendations as well. Additionally, in this work, we focused on the decoder-only GPT-2 architecture – as future work, we will consider other variants of generative models, e.g. encoder-decoder models, such as the T5 model [193], and models based on other-then-transformer architectures, such as Mamba [66].

8.10 Conclusions

In this chapter, we proposed the Next-K recommendation strategy, where the recommendation list is produced item-by-item, as an alternative to the standard score-and-rank approach. We also proposed GPTRec, a generative recommendation model benefiting from the Next-K strategy and optimisable for beyond-accuracy metrics, such as diversity. To train GPTRec, we proposed a 2-stage pre-training/fine-tuning approach, where at the pre-training stage, GPTRec learns to mimic the behaviour of traditional Top-K recommenders, and during the fine-tuning stage, it can be optimised for any metric using reinforcement learning. We demonstrated that the first stage (supervised pre-training) is enough to achieve NDCG@10 comparable to the state-of-the-art BERT4Rec model; however, fine-tuning can improve secondary objectives, such as diversity and popularity bias. In 3 out of 4 experiments, GPTRec fine-tuned for complex metrics, including diversity and popularity bias, outperformed the greedy re-ranking over BERT4Rec results. In 1 out of 4, the results were indistinguishable. For example, a fine-tuned version of GPTRec allowed us to simultaneously achieve 8.8% better NDCG and 8.6% lower popularity bias compared to greedy reranking over BERT4Rec results.

The methodology proposed in this allows direct optimisation of GPTRec (or, indeed, any other Transformer Decoder model) for any effectiveness measure without requiring the measure to be differentiable or computable at a single-item level. We demonstrated the applicability of this methodology using such beyond-accuracy objective targets as increasing diversity and decreasing popularity bias; however, the methodology can be used with other metrics, including variations of fairness measures, measures that balance the needs of various stakeholders of recommender systems and so on. We believe that this methodology is beneficial in industrial deployments of recommender systems, where optimisation criteria may also include such components as profitability, brand safety, sensitivity, etc. Currently, these goals are usually achieved through the complex pipelines of re-rankings and filters that apply heuristics and business rules, whereas the methodology presented in this chapter allows simplifying these pipelines and optimising for these goals directly – the only requirement is that the goals are measurable.

There are a number of possible future research directions that can build upon this chapter's methodology: Firstly, the scope of applicability of the methodology can be extended to larger datasets by applying sub-item representations instead of using atomic item IDs; Secondly, the quality of the generated rankings could possibly be further be improved by enhancing the model architecture; Thirdly, the recommendation goal can be used as the model input, allowing a single model to generate different kinds of recommendations and providing the user control over the desired output. Overall, exploring these future research directions could further refine the effectiveness, efficiency and applicability of recommendation systems in many scenarios.

This was the last methodological chapter of this thesis. Indeed, we now addressed all limitations of Transformer-based sequential recommender systems that we outlined Section 2.1.4. In the next chapter, we will summarise the key contributions and conclusions of this thesis, reflect on how the work validates the central thesis statement, and outline directions for future research.

Chapter 9

Conclusions and Future Work

9.1 Contributions

In this thesis, we argued that Transformer-based models can be used efficiently and effectively for large-scale Sequential Recommender Systems, both in training and inference, even when effectiveness extends beyond accuracy-based objectives. Our approach builds on the key observation that user behavioural sequences can be viewed as sequences of tokens in a specialised "behavioural language" (Section 2.3.1). Consequently, Transformer models originally designed for Natural Language Processing can be adapted for sequential recommendation. Prior work introduced Transformer-based models for this task, including SASRec (Section 2.3.3) and BERT4Rec (Section 2.3.4), where sequences of items replace the token sequences found in language models.

However, in Section 2.3.5, we argued that there are substantial differences between sequences of tokens in natural language and sequences of interactions in our "behavioural language," which hinder the widespread adoption of Transformer-based sequential recommender models in real-world applications. In particular, in Section 2.3.5, we outlined five key limitations of existing Transformer-based sequential recommender systems:

L2.1: **Training is slow** because each recommendation dataset has a unique set of items, making pre-training and fine-tuning approaches infeasible.

L2.2: **Computing all item scores is expensive during training** due to the large catalogue sizes typical of real-world recommender systems.

L2.3: **The item embedding tensor is too large**, primarily due to the extensive item catalogues in real-world deployments.

L2.4: **Computing all scores is expensive during inference**, again due to the large item catalogue.

L2.5: **Complex, beyond-accuracy objectives with limited training data** make standard supervised training techniques inefficient.

These limitations guided the direction of this thesis. Specifically, our contributions include:

1. **A systematic review and replicability study of SASRec and BERT4Rec** (Chapter 3), which helped us better understand the "state-of-the-art" in Sequential Recommender Systems and led to the development of **the aprec evaluation framework**, which we used throughout this thesis. Using the aprec framework, we also developed **ALBERT4Rec** and **DeBERTa4Rec** that can be more efficient than BERT4Rec. For example, on the MovieLens-1M dataset, ALBERT4Rec achieved 6% better NDCG@10 than BERT4Rec.
2. **The RSS training objective** (Chapter 4), which addressed Limitation L2.1 by enabling more efficient training of Transformer-based recommender systems. For example, an RSS-enhanced SASRec trained for just one hour achieves the same quality as a BERT4Rec model trained for 16 hours.
3. **The gBCE loss function and gSASRec/gBERT4Rec models** (Chapter 5), which addressed Limitation L2.2 by enabling effective training of Transformer-based recommender models with negative sampling. On the Gowalla dataset (with over 1M items), gSASRec is 47% more effective than standard SASRec, which suffers from the overconfidence side effect of negative sampling. Additionally, training a standard BERT4Rec model on this dataset is infeasible due to its lack of negative sampling support.
4. **The RecJPQ Item Embedding Tensor compression technique** (Chapter 6), which addressed Limitation L2.3 by adapting the Joint Product Quantization (JPQ) technique from Information Retrieval. For instance, on the Gowalla dataset, RecJPQ compressed the item embedding tensor by a factor of 50× without compromising model effectiveness. If necessary, RecJPQ can be combined with both the RSS training objective and the gBCE loss function.
5. **The RecJPQPrune Dynamic Pruning technique** (Chapter 7), which addressed Limitation L2.4 by accelerating model inference using RecJPQ's salient properties. On the Tmall dataset (2.2M items), it reduced the median model scoring time by 64× compared to the default Transformer scoring method used in SASRec and BERT4Rec.
6. **The Next-K recommendation strategy and the GPTRec model** (Chapter 8), which addressed Limitation L2.5 by generating recommendations autoregressively and hence allowing the next item in the recommendations list to be conditioned on already generated recommendations. In our experiments, we found that in 3 out of 4 cases, GPTRec's Next-K generation approach offers a better tradeoff between accuracy and secondary metrics (e.g. diversity) than classic greedy re-ranking techniques, such as Maximal Marginal Relevance.

Most of the techniques presented in this thesis can be combined. For example, in Section 6.6, we demonstrated that RecJPQ, gBCE, and RSS can be integrated to train an efficient and effective SASRec-based model. Similarly, in Section 7.2.4, we showed that RecJPQPrune enables efficient inference for both BERT4Rec- and SASRec-based models trained with gBCE and RecJPQ.

By addressing key limitations (Limitation L2.1-Limitation L2.5) of Transformer-based sequential recommendation models, this thesis introduces a set of techniques that not only improve efficiency and effectiveness but can also be combined for scalable, real-world deployment.

We now discuss the impact of this thesis on real-world applications and future work directions.

In summary, our contributions validate our Thesis Statement (stated in Section 1.1). Specifically, our proposed RSS training objective (Chapter 4) was shown to improve the efficiency of training Transformer-based recommender models, validating the claim of efficient training through recency-based sampling. The effectiveness of gSASRec and gBERT4Rec models trained with our gBCE loss function (Chapter 5) validated the claim of training effective Transformer-based models by reducing the overconfidence problem associated with negative sampling. The effectiveness and the efficiency of our RecJPQ embedding compression technique and our RecJPQ-Prune dynamic pruning method (Chapters 6 and 7) validated the claim that item embedding quantisation allow the scaling of Transformer-based recommender models to millions of items while significantly reducing memory footprints and accelerating inference. Finally, the GPTRec model and autoregressive Next-K recommendation generation strategy (Chapter 8) validated our claim that Transformer-based models can be aligned with beyond-accuracy goals–such as improved diversity–by leveraging generative models and reinforcement learning. Together, these contributions validate our Thesis Statement that Transformer-based models can be used for large-scale sequential Recommender Systems efficiently and effectively for both training and inference, even when effectiveness includes beyond-accuracy objectives. We now discuss the practical impact of these contributions on real-world deployments and outline promising directions for future research.

9.2 Impact and Future work

The main goal of this thesis is to bridge the gap between academic research and real-world industrial applications of Transformer-based recommender systems. Hence, the methodology presented in this thesis has a direct impact on the adoption of Transformer-based recommender systems in the industry. Indeed, we already have evidence of the methodology presented in the thesis being used in industrial applications. In particular:

- Wildberries, a large e-commerce company, deployed our ALBERT4Rec model (Chapter 3) in its recommendation pipeline, optimizing for large-scale personalized recommendations despite hardware constraints [89].
- Tencent Music, one of the largest music streaming platforms in China, used an approach inspired by our RSS (Chapter 4), specifically adopting its probabilistic item selection mechanism to enhance sequence augmentation [110]. By leveraging the probability function proposed in RSS, they designed swap and removal operations that selectively modify training sequences in a controlled manner. This method enriches training samples without requiring extra data, leading to improved recommendation performance while maintaining model simplicity.
- ZDF, one of the largest broadcasting companies in Germany, integrated a version of the SASRec model with our gBCE loss function (Chapter 5) to mitigate popularity bias [110]. This enhancement enables its streaming platform, ZDFmediathek, to provide more personalized and diverse recommendations.
- Our gSASRec model (Chapter 5) had been integrated into a number of industrial recommendation libraries, including Shaped.AI[1] and RecTools[2].

These real-world adoptions demonstrate the clear impact of the methodologies developed in this thesis. By addressing key challenges such as training efficiency and scaling to large catalogues, this work has already influenced the deployment of Transformer-based recommender systems at scale, supporting our claims in the thesis statement (Section 1.1).

Beyond these specific applications, this thesis's contributions provide a foundation for further advancements in Transformer-based recommender systems, encouraging both academic and industry researchers to explore new frontiers in personalization and efficiency.

This thesis also opens a number of possible future research directions. In particular:

- **Applications beyond Sequential Recommendation**. Our main focus in this thesis was on Sequential recommendation; however, we believe that many of our proposed techniques can be used in other domains, such as Information Retrieval or Natural Language Processing. For example, we have already shown that the gBCE loss function can be used for training document retrieval models [173]. However, we believe that other methodologies, including RecJPQ (Chapter 6) and RecJPQPrune, can also be applied to Information Retrieval tasks.

---

1. hhttps://news.ycombinator.com/item?id=41235733
2. https://github.com/MobileTeleSystems/RecTools/blob/main/examples/tutorials/transformers_tutorial.ipynb

- **Multi-modal item representation**. Our RecJPQ (Chapter 6) only uses collaborative signals to derive sub-item ID representations. However, we believe that it is possible to also use content-based item features, such as item descriptions or images, to achieve better item representations in RecJPQ.
- **Fusing Language and Recommendation Models.** Our research was only focused on using user behaviour sequences as the model input. However, we believe that the strong text-processing capabilities of modern language models can enhance recommender systems, for example, by allowing users to provide free-text guidance for recommendations. Our RecJPQ item tokenization technique (Chapter 6) can help expand the vocabularies of Language Models with item-related tokens, while our Reinforcement Learning-based optimization (Chapter 8) can help align these models with recommendation goals.

These directions highlight the potential for further improving Transformer-based recommender systems by expanding their applicability, enhancing item representations, and integrating them more effectively with language models. Addressing these challenges could lead to more efficient, interpretable, and user-centric recommendation systems, bridging the gap between research and real-world deployment even further.

9.3 Concluding Remarks

This thesis examined the challenges of Sequential Recommendation in real-world settings, particularly with large item catalogues and objectives beyond ranking accuracy. While Transformer-based models like SASRec and BERT4Rec demonstrated strong ranking accuracy, they also exhibited key limitations, including slow training, inefficiency with large catalogues, and limited effectiveness for beyond-accuracy goals. To address these challenges, we introduced several novel techniques: RSS, a training objective for improved efficiency; gBCE, a loss function designed for negative sampling; RecJPQ, a sub-item representation technique for reducing GPU memory requirements; RecJPQPrune, a dynamic pruning approach for efficient inference; and Next-K recommendation generation, an autoregressive strategy for optimising beyond-accuracy objectives. These methodologies have already been applied in industrial settings, demonstrating their practical value. Finally, in Section 9.2, we outlined directions for future work to further advance the adoption of Transformer-based models in real-world applications.

Bibliography

[1] Martin Abadi et al. 2016. Tensorflow: A System for large-scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 265–283.

[2] Antonio Acquavia, Nicola Tonellotto and Craig Macdonald. 2023. Static pruning for multi-representation dense retrieval. In *Proc. DocEng*.

[3] Panagiotis Adamopoulos. 2013. Beyond rating prediction accuracy: on new perspectives in recommender systems. In *Proc. RecSys*, 459–462.

[4] M. Mehdi Afsar, Trafford Crump and Behrouz Far. 2023. Reinforcement Learning based Recommender Systems: A Survey. *ACM Computing Surveys*, 55, 7, 1–38.

[5] Mehrnaz Amjadi, Seyed Danial Mohseni Taheri and Theja Tulabandhula. 2021. Katrec: knowledge aware attentive sequential recommendations. In *Proc. ICDS*, 305–320.

[6] Vo Ngoc Anh, Owen de Kretser and Alistair Moffat. 2001. Vector-space ranking with effective early termination. In *Proc. SIGIR*, 35–42. ISBN: 1-58113-331-6.

[7] Arda Antikacioglu, Tanvi Bajpai and R. Ravi. 2019. A New System-Wide Diversity Measure for Recommendations with Efficient Algorithms. *SIAM Journal on Mathematics of Data Science*, 1, 4, 759–779.

[8] Nabiha Asghar. 2016. Yelp dataset challenge: review rating prediction. *arXiv preprint arXiv:1605.05362*.

[9] Xueying Bai, Jian Guan and Hongning Wang. 2019. A model-based reinforcement learning with adversarial training for online recommendation. In *Proc. NeurIPS*, 10735–10746.

[10] Marko Balabanović and Yoav Shoham. 1997. Fab: content-based, collaborative recommendation. *Commun. ACM*, 40, 3, (Mar. 1997), 66–72.

[11] Jan Van Balen and Mark Levy. 2019. PQ-VAE: Efficient Recommendation Using Quantized Embeddings. In *Proc. RecSys*, 46–50.

[12] Yoshua Bengio, Réjean Ducharme, Pascal Vincent and Christian Jauvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3, 1137–1155.

[13] James Bennett and Stan Lanning. 2007. The netflix prize. In Proceedings of KDD Cup and Workshop. Vol. 2007, 35.

[14] Shuqing Bian, Wayne Xin Zhao, Kun Zhou, Jing Cai, Yancheng He, Cunxiang Yin and Ji-Rong Wen. 2021. Contrastive curriculum learning for sequential user behavior modeling via data augmentation. In *Proc. CIKM*, 3737–3746.

[15] Rodrigo Borges and Kostas Stefanidis. 2021. On mitigating popularity bias in recommendations via variational autoencoders. In *Proc. SAC*, 1383–1389.

[16] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *Proc. CIKM*, 426–434.

[17] Tom Brown et al. 2020. Language Models are Few-Shot Learners. In *Proc. NeurIPS*. Vol. 33, 1877–1901.

[18] Chris Buckley and Alan F. Lewit. 1985. Optimization of inverted vector searches. In *Proc. SIGIR*, 97–110. ISBN: 0-89791-159-8.

[19] Christopher Burges. 2010. From RankNet to LambdaRank to LambdaMART: An overview. *Learning*, 11, (Jan. 2010).

[20] Michael Busch, Krishna Gade, Brian Larson, Patrick Lok, Samuel Luckenbill and Jimmy Lin. 2012. Earlybird: real-time search at twitter. In *Proc. ICDE*, 1360–1369.

[21] Wei Cai, Weike Pan, Jingwen Mao, Zhechao Yu and Congfu Xu. 2022. Aspect Re-distribution for Learning Better Item Embeddings in Sequential Recommendation. In *Proc. RecSys*. (Sept. 2022), 49–58.

[22] B. Barla Cambazoglu, Hugo Zaragoza, Olivier Chapelle, Jiang Chen, Ciya Liao, Zhaohui Zheng and Jon Degenhardt. 2010. Early exit optimizations for additive machine learned ranking systems. In *Proc. WSDM*, 411–420.

[23] Rocío Cañamares and Pablo Castells. 2020. On Target Item Sampling in Offline Recommender System Evaluation. In *Proc. RecSys*, 259–268.

[24] Iván Cantador, Peter Brusilovsky and Tsvi Kuflik. 2011. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *Proc. RecSys* (RecSys 2011). Chicago, IL, USA.

[25] Jaime Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proc. SIGIR*, 335–336.

[26] Haw-Shiuan Chang, Nikhil Agarwal and Andrew McCallum. 2024. To Copy, or not to Copy; That is a Critical Issue of the Output Softmax Layer in Neural Sequential Recommenders. In *Proc. WSDM*.

[27] Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview. *Proceedings of Machine Learning Research*, 1–24.

[28] Weilong Chen, Shaoliang Zhang, Ruobing Xie, Feng Xia, Leyu Lin, Xinran Zhang, Yan Wang and Yanru Zhang. 2024. CIPPO: Contrastive Imitation Proximal Policy Optimization for Recommendation Based on Reinforcement Learning. *IEEE Trans. on Knowledge and Data Engineering*, 36, 11, (Nov. 2024), 5753–5767.

[29] Yongjun Chen, Jia Li, Zhiwei Liu, Nitish Shirish Keskar, Huan Wang, Julian McAuley and Caiming Xiong. 2022. Generating Negative Samples for Sequential Recommendation. (2022). arXiv: 2208.03645 [cs].

[30] Rewon Child, Scott Gray, Alec Radford and Ilya Sutskever. 2019. Generating Long Sequences with Sparse Transformers. arXiv: 1904.10509.

[31] Jin Yao Chin, Yile Chen and Gao Cong. 2022. The datasets dilemma: how much do we really know about recommendation datasets? In *Proc. WSDM*, 141–149.

[32] Eunjoon Cho, Seth A. Myers and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *Proc. KDD*, 1082.

[33] Sung Min Cho, Eunhyeok Park and Sungjoo Yoo. 2020. Meantime: mixture of attention mechanisms with multi-temporal embeddings for sequential recommendation. In *Proc. RecSys*, 515–520.

[34] Kevin Clark, Minh-Thang Luong, Quoc V. Le and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *Proc. ICLR*. arXiv: 2003.10555 [cs].

[35] Gordon V. Cormack, Ondrej Lhotak and Christopher R. Palmer. 1999. Estimating precision by random sampling. In *Proc. SIGIR*, 273–274.

[36] Paul Covington, Jay Adams and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proc. RecSys*, 191–198.

[37] Paolo Cremonesi, Yehuda Koren and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *Proc. RecSys*. Barcelona Spain, (26th Sept. 2010), 39–46.

[38] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song and Serge Belongie. 2019. Class-Balanced Loss Based on Effective Number of Samples. In *Proc. CVPR*, 9268–9277.

[39] Zeyu Cui, Jianxin Ma, Chang Zhou, Jingren Zhou and Hongxia Yang. 2022. M6-Rec: Generative Pretrained Language Models are Open-Ended Recommender Systems. (May 2022). Retrieved 27th Apr. 2023 from arXiv: arXiv:2205.08084 (cs).

[40] Alexander Dallmann, Daniel Zoller and Andreas Hotho. 2021. A Case Study on Sampling Strategies for Evaluating Neural Sequential Item Recommendation Models. In *Proc. RecSys*. (Sept. 2021), 505–514.

[41] Kalyanmoy Deb and Kalyanmoy Deb. 2014. Multi-objective optimization. In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Boston, MA, 403–449.

[42] Romain Deffayet, Thibaut Thonet, Jean-Michel Renders and Maarten De Rijke. 2023. Generative Slate Recommendation with Reinforcement Learning. In *Proc. WSDM*, 580–588.

[43] Wei Deng, Junwei Pan, Tian Zhou, Deguang Kong, Aaron Flores and Guang Lin. 2021. DeepLight: Deep Lightweight Feature Interactions for Accelerating CTR Predictions in Ad Serving. In *Proc. WSDM*, 922–930.

[44] Luke de Oliveira and Alfredo Láinez. 2018. Bayesian Personalized Ranking for Spark. https://github.com/alfredolainez/bpr-spark. [Online; accessed 14-December-2023]. (2018).

[45] Gabriel de Souza Pereira Moreira, Sara Rabhi, Jeong Min Lee, Ronay Ak and Even Oldridge. 2021. Transformers4rec: bridging the gap between nlp and sequential/session-based recommendation. In *Proc. RecSys*, 143–153.

[46] Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. of NAACL-HLT*, 4171–4186.

[47] 2023. Dimensionality Reduction - RDD-based API. https://spark.apache.org/docs/latest/mllib-dimensionality-reduction. [Online; accessed 14-December-2023]. (2023).

[48] Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In *Proc. SIGIR*, 993–1002.

[49] Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell and Matt Gardner. 2021. Documenting Large Webtext Corpora: A Case Study on the Colossal Clean Crawled Corpus. In *Proc. EMNLP*, 1286–1305.

[50] Hanwen Du, Hui Shi, Pengpeng Zhao, Deqing Wang, Victor S. Sheng, Yanchi Liu, Guanfeng Liu and Lei Zhao. 2022. Contrastive Learning with Bidirectional Transformers for Sequential Recommendation. In *Proc. CIKM*. (Oct. 2022), 396–405.

[51] Yu-Chen Fan, Yitong Ji, Jie Zhang and Aixin Sun. 2024. Our Model Achieves Excellent Performance on MovieLens: What Does It Mean? *ACM Transactions on Information Systems*, 42, 6, (30th Nov. 2024), 1–25.

[52] Xinyan Fan, Zheng Liu, Jianxun Lian, Wayne Xin Zhao, Xing Xie and Ji-Rong Wen. 2021. Lighter and better: low-rank decomposed self-attention networks for next-item recommendation. In *Proc. SIGIR*, 1733–1737.

[53] Ziwei Fan, Zhiwei Liu, Jiawei Zhang, Yun Xiong, Lei Zheng and Philip S Yu. 2021. Continuous-time sequential recommendation with temporal graph collaborative transformer. In *Proc. CIKM*, 433–442.

[54] Qian Feng, Geyang Xiao, Yuan Liang, Huifeng Zhang, Linlin Yan and Xiaoyu Yi. 2022. Proximal Policy Optimization for Explainable Recommended Systems. In *Proc. DOCS*, 1–6.

[55] Maurizio Ferrari Dacrema, Paolo Cremonesi and Dietmar Jannach. 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proc. RecSys*, 101–109.

[56] Norbert Fuhr. 2021. Proof by experimentation? towards better IR research. In *ACM SIGIR Forum* number 2. Vol. 54, 1–4.

[57] Funk. 2006. Netflix Update: Try This at Home (Funk SVD Blog POST). https://sifter.org/simon/journal/20061211.html.

[58] Philip Gage. 1994. A new algorithm for data compression. *C Users J.*, 12, 2, (1st Feb. 1994), 23–38.

[59] Mouzhi Ge, Carla Delgado-Battenfeld and Dietmar Jannach. 2010. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proc. RecSys*, 257–260.

[60] Tiezheng Ge, Kaiming He, Qifa Ke and Jian Sun. 2014. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36, 4, 744–755.

[61] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge and Yongfeng Zhang. 2022. Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5). In *Proc. RecSys*. (Sept. 2022), 299–315.

[62] David Goldberg, David Nichols, Brian M. Oki and Douglas Terry. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35, 12, (1st Dec. 1992), 61–70.

[63] Dmitri Goldenberg and Pavel Levin. 2021. Booking.com multi-destination trips dataset. In *Proc. SIGIR*, 2457–2462.

[64] Ian Goodfellow, Yoshua Bengio and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[65] Robert M. Gray. 1984. Vector quantization. *IEEE Assp*, 1, 2.

[66] Albert Gu and Tri Dao. 2023. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. arXiv: 2312.00752 [cs].

[67] Asela Gunawardana, Guy Shani and Sivan Yogev. 2022. Evaluating Recommender Systems. In *Recommender Systems Handbook*. Francesco Ricci, Lior Rokach and Bracha Shapira, editors. Springer US, New York, NY, 547–601.

[68] Chuan Guo, Geoff Pleiss, Yu Sun and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *Proc. ICML*, 1321–1330.

[69] N. Halko, P. G. Martinsson and J. A. Tropp. 2011. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53, 2, 217–288.

[70] Casper Hansen, Christian Hansen, Lucas Maystre, Rishabh Mehrotra, Brian Brost, Federico Tomasi and Mounia Lalmas. 2020. Contextual and Sequential User Embeddings for Large-Scale Music Recommendation. In *Proc. RecSys*, 53–62.

[71] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5, 4, (Dec. 2015), 19:1–19:19.

[72] Pengcheng He, Xiaodong Liu, Jianfeng Gao and Weizhu Chen. 2020. DeBERTa: decoding-enhanced bert with disentangled attention. In *Proc. ICLR*.

[73] Ruining He and Julian McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *Proc. WWW*. International World Wide Web Conferences Steering Committee, Montréal Québec Canada, (Apr. 2016), 507–517.

[74] Ruining He and Julian McAuley. 2016. Ups and downs: modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proc. WWW*, 507–517.

[75] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proc. WWW*, 173–182.

[76] Zhankui He, Handong Zhao, Zhe Lin, Zhaowen Wang, Ajinkya Kale and Julian Mcauley. 2021. Locker: locally constrained self-attentive sequential recommendation. In *Proc. CIKM*, 3088–3092.

[77] Harold Stanley Heaps. 1978. *Information Retrieval: Computational and Theoretical Aspects*. Academic Press, Inc., USA.

[78] Balázs Hidasi and Ádám Tibor Czapp. 2023. The Effect of Third Party Implementations on Reproducibility. In *Proc. RecSys*, 272–282.

[79] Balázs Hidasi and Ádám Tibor Czapp. 2023. Widespread Flaws in Offline Evaluation of Recommender Systems. In *Proc. RecSys*, 848–855.

[80] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. In *Proc. CIKM*, 843–852.

[81] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *Proc. ICLR*.

[82] Geoffrey Hinton, Oriol Vinyals and Jeff Dean. 2014. Distilling the Knowledge in a Neural Network. In Deep Learning Workshop @ NeurIPS. arXiv: 1503.02531 [cs, stat].

[83] Yupeng Hou, Zhankui He, Julian McAuley and Wayne Xin Zhao. 2023. Learning Vector-Quantized Item Representation for Transferable Sequential Recommenders. In *Proc. WWW*, 1162–1171.

[84] 2023. How Many Videos Are on YouTube in 2024? https://earthweb.com/how-many-videos-are-on-youtube/. [Online; accessed 14-December-2023]. (2023).

[85] Kaixi Hu, Lin Li, Qing Xie, Jianquan Liu and Xiaohui Tao. 2021. What is next when sequential prediction meets implicitly hard interaction? In *Proc. CIKM*, 710–719.

[86] Ziniu Hu, Yang Wang, Qu Peng and Hang Li. 2019. Unbiased LambdaMART: An Unbiased Pairwise Learning-to-Rank Algorithm. In *Proc. WWW*. (May 2019), 2830–2836.

[87] Liwei Huang, Yutao Ma, Yanbo Liu, Bohong Danny Du, Shuliang Wang and Deyi Li. 2021. Position-enhanced and time-aware graph convolutional network for sequential recommendations. *ACM Transactions on Information Systems (TOIS)*.

[88] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proc. CIKM*, 2333–2338.

[89] Evgeniy Ivanov. 2024. Fear and loathing in recommendations: how to get albert4rec into production on 1080ti. Wildberries Blog. (17th Mar. 2024). https://telegra.ph/Strah-i-nenavist-v-rekomendaciyah-kak-zatashchit-ALBERT4Rec-v-prod-na-1080TI-03-17.

[90] Gautier Izacard, Fabio Petroni, Lucas Hosseini, Nicola De Cao, Sebastian Riedel and Edouard Grave. 2020. A Memory Efficient Baseline for Open Domain Question Answering. (2020). arXiv: 2012.15156 [cs].

[91] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20, 4, (Oct. 2002), 422–446.

[92] Sébastien Jean, Kyunghyun Cho, Roland Memisevic and Yoshua Bengio. 2015. On Using Very Large Target Vocabulary for Neural Machine Translation. In *Proc. ACL-IJCNLP*. (July 2015), 1–10.

[93] Herve Jégou, Matthijs Douze and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33, 1, 117–128.

[94] Myeongjae Jeon, Saehoon Kim, Seung-won Hwang, Yuxiong He, Sameh Elnikety, Alan L. Cox and Scott Rixner. 2014. Predictive parallelization: taming tail latencies in web search. In *Proc. SIGIR*, 253–262.

[95] Yitong Ji, Aixin Sun, Jie Zhang and Chenliang Li. 2020. A Re-visit of the Popularity Baseline in Recommender Systems. In *Proc. SIGIR*. (July 2020), 1749–1752.

[96] Xiang-Fei Jia, Andrew Trotman and Richard O'Keefe. 2010. Efficient accumulator initialisation. In *Proc. ADCS*, 44–51.

[97] Jeff Johnson, Matthijs Douze and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, 7, 3, 535–547.

[98] Taegwan Kang, Hwanhee Lee, Byeongjin Choe and Kyomin Jung. 2021. Entangled bidirectional encoder to autoregressive decoder for sequential recommendation. In *Proc. SIGIR*, 1657–1661.

[99] Wang-Cheng Kang, Derek Zhiyuan Cheng, Ting Chen, Xinyang Yi, Dong Lin, Lichan Hong and Ed H. Chi. 2020. Learning Multi-granular Quantized Embeddings for Large-Vocab Categorical Features in Recommender Systems. In *Proc. WWW*, 562–566.

[100] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *Proc. ICDM*. (Nov. 2018), 197–206.

[101] Jussi Karlgren. 1990. *An Algebra for Recommendations : Using Reader Data as a Basis for Measuring Document Proximity*.

[102] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proc. EMNLP*. arXiv: 2004.04906 [cs].

[103] Mesut Kaya. 2018. Accurate and Diverse Recommendations Using Item-Based SubProfiles. In *Proc. FLAIRS*.

[104] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye and Tie-Yan Liu. 2017. LightGBM: a highly efficient gradient boosting decision tree. In *Proc. NeurIPS*, 3146–3154.

[105] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proc. SIGIR*, 39–48.

[106] Anton Klenitskiy and Alexey Vasilev. 2023. Turning Dross Into Gold Loss: is BERT4Rec really better than SASRec? In *Proc. RecSys*, 1120–1125.

[107] Anton Klenitskiy, Anna Volodkevich, Anton Pembek and Alexey Vasilev. 2024. Does It Look Sequential? An Analysis of Datasets for Evaluation of Sequential Recommendations. In *Proc. RecSys*. Bari Italy, (8th Oct. 2024), 1067–1072.

[108] Anastasiia Klimashevskaia, Dietmar Jannach, Mehdi Elahi and Christoph Trattner. 2024. A survey on popularity bias in recommender systems. *User Modeling and User-Adapted Interaction*, (1st July 2024).

[109] Alexander Kolesnikov et al. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *Proc. ICLR*.

[110] Venkata Harshit Koneru, Xenija Neufeld, Sebastian Loth and Andreas Grün. 2024. Enhancing Recommendation Quality of the SASRec Model by Mitigating Popularity Bias. In *Proc. RecSys*. Bari Italy, (8th Oct. 2024), 781–783.

[111] Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA, (28th June 2009), 447–456.

[112] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Proc. KDD. Las Vegas Nevada USA, (24th Aug. 2008), 426–434.

[113] Yehuda Koren. 2009. The BellKor Solution to the Netflix Grand Prize. `https://www.asc.ohio-state.edu/statistics/dmsl/GrandPrize2009_BPC_BellKor.pdf`.

[114] Yehuda Koren, Robert Bell and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42, 8, (Aug. 2009), 30–37.

[115] Yehuda Koren, Steffen Rendle and Robert Bell. 2022. Advances in Collaborative Filtering. In *Recommender Systems Handbook*, 91–142.

[116] Denis Kotkov, Jari Veijalainen and Shuaiqiang Wang. 2020. How does serendipity affect diversity in recommender systems? A serendipity-oriented greedy algorithm. *Computing*, 102, 2, 393–411.

[117] Walid Krichene and Steffen Rendle. 2022. On sampled metrics for item recommendation. *Communications of the ACM*, 65, 7, (June 2022), 75–83.

[118] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proc. NeurIPS*. Vol. 25.

[119] Maciej Kula. 2015. Metadata embeddings for user and item cold-start recommendations. In *Proc. Workshop on New Trends on Content-Based Recommender @ RecSys* (CEUR Workshop Proc.). Vol. 1448, 14–21.

[120] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *Proc. ICLR*. (Mar. 2020).

[121] Carlos Lassance and Stéphane Clinchant. 2022. An efficiency study for splade models. In *Proc. SIGIR*, 2220–2226. ISBN: 9781450387323.

[122] Hyunsung Lee, Sangwoo Cho, Yeongjae Jang, Jaekwang Kim and Honguk Woo. 2021. Differentiable Ranking Metric Using Relaxed Sorting for Top-K Recommendation. *Proc. IEEE Access*, 9, 114649–114658.

[123] Chenglin Li, Mingjun Zhao, Huanming Zhang, Chenyun Yu, Lei Cheng, Guoqiang Shu, BeiBei Kong and Di Niu. 2022. Recguru: adversarial learning of generalized user representations for cross-domain recommendation, 571–581.

[124] Haoyang Li, Xin Wang, Ziwei Zhang, Jianxin Ma, Peng Cui and Wenwu Zhu. 2021. Intention-aware sequential recommendation with structured intent transition. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*.

[125] Jiacheng Li, Ming Wang, Jin Li, Jinmiao Fu, Xin Shen, Jingbo Shang and Julian McAuley. 2023. Text Is All You Need: Learning Language Representations for Sequential Recommendation. In *Proc. KDD*. (6th Aug. 2023), 1258–1267.

[126] Jiacheng Li, Yujie Wang and Julian McAuley. 2020. Time Interval Aware Self-Attention for Sequential Recommendation. In *Proc. WSDM*. (20th Jan. 2020), 322–330.

[127] Jiacheng Li, Tong Zhao, Jin Li, Jim Chan, Christos Faloutsos, George Karypis, Soo-Min Pantel and Julian McAuley. 2022. Coarse-to-Fine Sparse Sequential Recommendation. In *Proc. SIGIR*. (6th July 2022), 2082–2086.

[128] Roger Zhe Li, Julián Urbano and Alan Hanjalic. 2021. New insights into metric optimization for ranking-based recommendation. In *Proc. SIGIR*, 932–941.

[129] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32, 8, 1475–1488.

[130] Yang Li, Tong Chen, Peng-Fei Zhang and Hongzhi Yin. 2021. Lightweight self-attentive sequential recommendation. In *Proc. CIKM*, 967–977.

[131] Yicong Li, Hongxu Chen, Xiangguo Sun, Zhenchao Sun, Lin Li, Lizhen Cui, Philip S Yu and Guandong Xu. 2021. Hyperbolic hypergraphs for sequential recommendation. In *Proc. CIKM*, 988–997.

[132] Yunzhe Li, Yue Ding, Bo Chen, Xin Xin, Yule Wang, Yuxiang Shi, Ruiming Tang and Dong Wang. 2021. Extracting attentive social temporal excitation for sequential recommendation. In *Proc. CIKM*, 998–1007.

[133] Defu Lian, Haoyu Wang, Zheng Liu, Jianxun Lian, Enhong Chen and Xing Xie. 2020. LightRec: A Memory and Search-Efficient Recommender System. In *Proc. WWW*, 695–705.

[134] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman and Tony Jebara. 2018. Variational autoencoders for collaborative filtering. In *Proc. WWW*, 689–698.

[135] Jimmy Lin, Rodrigo Nogueira and Andrew Yates. 2022. *Pretrained Transformers for Text Ranking: BERT and Beyond*. *Synthesis Lectures on Human Language Technologies*. Springer International Publishing.

[136] Junyang Lin et al. 2021. M6: A Chinese Multimodal Pretrainer. (2021). arXiv: 2103.00823 [cs].

[137] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He and Piotr Dollár. 2018. Focal Loss for Dense Object Detection. (2018). arXiv: 1708.02002 [cs].

[138] Yuanguo Lin, Yong Liu, Fan Lin, Lixin Zou, Pengcheng Wu, Wenhua Zeng, Huanhuan Chen and Chunyan Miao. 2023. A Survey on Reinforcement Learning for Recommender Systems. arXiv: 2109.10665.

[139] G. Linden, B. Smith and J. York. 2003. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7, 1, (Jan. 2003), 76–80.

[140] Chang Liu, Xiaoguang Li, Guohao Cai, Zhenhua Dong, Hong Zhu and Lifeng Shang. 2021. Noninvasive self-attention for side information fusion in sequential recommendation. In *Proc. AAAI*, 4249–4256.

[141] Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3, 3, 225–331.

[142] Zhiwei Liu, Yongjun Chen, Jia Li, Philip S Yu, Julian McAuley and Caiming Xiong. 2021. Contrastive self-supervised sequential recommendation with robust augmentation. *arXiv preprint arXiv:2108.06479*.

[143] Zhiwei Liu, Ziwei Fan, Yu Wang and Philip S. Yu. 2021. Augmenting sequential recommendation with pseudo-prior items via reversely pre-training transformer. In *Proc. SIGIR*, 1608–1612.

[144] Malte Ludewig and Dietmar Jannach. 2018. Evaluation of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*, 28, 4-5, (Dec. 2018), 331–390.

[145] Jianxin Ma, Chang Zhou, Hongxia Yang, Peng Cui, Xin Wang and Wenwu Zhu. 2020. Disentangled self-supervision in sequential recommenders. In *Proc. KDD*, 483–491.

[146] Craig Macdonald, Nicola Tonellotto and Iadh Ounis. 2012. Learning to predict response times for online query scheduling. In *Proc. SIGIR*, 621–630.

[147] Joel Mackenzie, J. Shane Culpepper, Roi Blanco, Matt Crane, Charles L. A. Clarke and Jimmy Lin. 2018. Query driven algorithm selection in early stage retrieval. In *Proc. WSDM*, 396–404.

[148] Joel Mackenzie, Andrew Trotman and Jimmy Lin. 2023. Efficient document-at-a-time and score-at-a-time query evaluation for learned sparse representations. *ACM Trans. Inf. Syst.*, 41, 4, (Mar. 2023).

[149] J. MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Vol. 5.1, 281–298.

[150] Pranava Madhyastha and Rishabh Jain. 2019. On model stability as a function of random seed. *arXiv preprint arXiv:1909.10447*.

[151] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42, 4, 824–836.

[152] Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto and Rossano Venturini. 2017. Faster blockmax wand with variable-sized blocks. In *Proc. SIGIR*, 625–634. ISBN: 978-1-4503-5022-8.

[153] Yusuke Matsui, Hiho Karuta, Calvin McCarter, Fangrui Liu, Hiroyuki Deguchi and Lee Butterman. 2023. Nano Product Quantization (nanopq): a vanilla implementation of Product Quantization (PQ) and Optimized Product Quantization (OPQ) written in pure python without any third party dependencies. `https://github.com/matsui528/nanopq`. [Online; accessed 14-December-2023]. (2023).

[154] Warren S. McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5, 4, (Dec. 1943), 115–133.

[155] Leland McInnes, John Healy and James Melville. 2020. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv: `1802.03426 [cs, stat]`.

[156] Paulius Micikevicius et al. 2018. Mixed Precision Training. In *Proc. ICLR*.

[157] Aleksandr Milogradskii, Oleg Lashinin, Alexander P, Marina Ananyeva and Sergey Kolesnikov. 2024. Revisiting BPR: A Replicability Study of a Common Recommender System Baseline. In *18th ACM Conference on Recommender Systems*. Proc. RecSys. Bari Italy, (8th Oct. 2024), 267–277.

[158] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. (2013). arXiv: 1312.5602 [cs].

[159] Alistair Moffat and Justin Zobel. 1994. Fast ranking in limited space. In *Proc. ICDE*, 428–437.

[160] Alistair Moffat and Justin Zobel. 1996. Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst.*, 14, 4, 349–379.

[161] Kevin P. Murphy. 2022. *Probabilistic Machine Learning: An Introduction*. *Adaptive Computation and Machine Learning Series*. The MIT Press, Cambridge, Massachusetts.

[162] Maxim Naumov et al. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. (2019). arXiv: 1906.00091 [cs].

[163] Rodrigo Nogueira and Kyunghyun Cho. 2020. Passage Re-ranking with BERT. (14th Apr. 2020). arXiv: 1901.04085 [cs].

[164] Kemal Oksuz, Baris Can Cam, Sinan Kalkan and Emre Akbas. 2021. Imbalance Problems in Object Detection: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43, 10, 3388–3415.

[165] Long Ouyang et al. 2022. Training language models to follow instructions with human feedback. (2022). arXiv: 2203.02155 [cs].

[166] Vaibhav Padhye, Kailasam Lakshmanan and Amrita Chaturvedi. 2023. Proximal policy optimization based hybrid recommender systems for large scale recommendations. *Multimedia Tools and Applications*, 82, 13, 20079–20100.

[167] Umaporn Padungkiatwattana, Thitiya Sae-Diae, Saranya Maneeroj and Atsuhiro Takasu. 2022. Arerec: attentive local interaction model for sequential recommendation. In.

[168] Mahdi Pakdaman Naeini, Gregory Cooper and Milos Hauskrecht. 2015. Obtaining Well Calibrated Probabilities Using Bayesian Binning. In *Proc. AAAI*. Vol. 29.

[169] Yoon-Joo Park and Alexander Tuzhilin. 2008. The long tail of recommender systems and how to leverage it. In *Proc. RecSys*, 11–18.

[170] Apurva Pathak, Kshitiz Gupta and Julian McAuley. 2017. Generating and Personalizing Bundle Recommendations on *Steam*. In *Proc. SIGIR*. (Aug. 2017), 1073–1076.

[171] Roberto Pellegrini, Wenjie Zhao and Iain Murray. 2022. Don't recommend the obvious: estimate probability ratios. In *Proc. RecSys*. (Sept. 2022), 188–197.

[172] Aleksandr V Petrov, Craig Macdonald and Nicola Tonellotto. 2025. Efficient recommendation with millions of items by dynamic pruning of sub-item embeddings. (2025).

[173] Aleksandr V. Petrov, Sean MacAvaney and Craig Macdonald. 2024. Shallow Cross-Encoders for Low-Latency Retrieval. In *Proc. ECIR*.

[174] Aleksandr V. Petrov and Craig Macdonald. 2022. A Systematic Review and Replicability Study of BERT4Rec for Sequential Recommendation. In *Proc. RecSys*, 436–447.

[175] Aleksandr V. Petrov and Craig Macdonald. 2024. Aligning GPTRec with Beyond-Accuracy Goals with Reinforcement Learning. In *Proc. GenRec@TheWebConf*. (7th Mar. 2024). arXiv: 2403.04875.

[176] Aleksandr V. Petrov and Craig Macdonald. 2022. Effective and Efficient Training for Sequential Recommendation using Recency Sampling. In *Proc. RecSys*. arXiv: 2207.02643 [cs].

[177] Aleksandr V. Petrov and Craig Macdonald. 2023. Generative Sequential Recommendation with GPTRec. In *Proc. Gen-IR@SIGIR*.

[178] Aleksandr V. Petrov and Craig Macdonald. 2023. gSASRec: Reducing Overconfidence in Sequential Recommendation Trained with Negative Sampling. In *Proc. RecSys*, 116–128.

[179] Aleksandr V. Petrov and Craig Macdonald. 2024. gSASRec: Reducing Overconfidence in Sequential Recommendation Trained with Negative Sampling (Extended Abstract). In Proc. IJCAI. Vol. 9. (1st Aug. 2024), 8447–8449.

[180] Aleksandr V. Petrov and Craig Macdonald. 2024. RecJPQ: Training Large-Catalogue Sequential Recommenders. In *Proc. WSDM*.

[181] Aleksandr V. Petrov and Craig Macdonald. 2025. RSS: Effective and Efficient Training for Sequential Recommendation Using Recency Sampling. *ACM Transactions on Recommender Systems*, 3, 1, 1–32.

[182] Aleksandr V. Petrov, Craig Macdonald and Nicola Tonellotto. 2024. Efficient Inference of Sub-Item Id-based Sequential Recommendation Models with Millions of Items. In *Proc. RecSys*. Bari Italy, 912–917.

[183] Aleksandr V. Petrov and Yuriy Makarov. 2021. Attention-based neural re-ranking approach for next city in trip recommendations. In *Proc. WSDM WebTour*, 41–45.

[184] Aleksandr Vladimirovich Petrov and Craig Macdonald. 2024. Improving Effectiveness by Reducing Overconfidence in Large Catalogue Sequential Recommendation with gBCE loss. *ACM Transactions on Recommender Systems*, (7th Oct. 2024), 3699521.

[185] Luan Thanh Pham, Erdinc Oksum, Thanh Duc Do, Minh Le-Huy, Minh Duc Vu and Vinh Duc Nguyen. 2019. LAS: A combination of the analytic signal amplitude and the generalised logistic function as a novel edge enhancement of magnetic data. *Contributions to Geophysics & Geodesy*, 49, 4, 425–440.

[186] Michael Potter, Hamlin Liu, Yash Lala, Christian Loanzon and Yizhou Sun. 2022. Gru4recbe: a hybrid session-based movie recommendation system (student abstract).

[187] Ronak Pradeep, Kai Hui, Jai Gupta, Adam Lelkes, Honglei Zhuang, Jimmy Lin, Donald Metzler and Vinh Tran. 2023. How Does Generative Retrieval Scale to Millions of Passages? In *Proc. EMNLP*, 1305–1321.

[188] Zhen Qin, Le Yan, Honglei Zhuang, Yi Tay, Rama Kumar Pasumarthi, Xuanhui Wang, Michael Bendersky and Marc Najork. 2021. Are neural rankers still outperformed by gradient boosted decision trees? In *Proc. ICLR*.

[189] Ruihong Qiu, Zi Huang, Hongzhi Yin and Zijian Wang. 2022. Contrastive Learning for Representation Degeneration Problem in Sequential Recommendation. In *Proc. WSDM*. (Feb. 2022), 813–823.

[190] Massimo Quadrana, Paolo Cremonesi and Dietmar Jannach. 2018. Sequence-aware recommender systems. *ACM Computing Surveys (CSUR)*, 51, 4, 1–36.

[191] Alec Radford, Karthik Narasimhan, Tim Salimans and Ilya Sutskever. 2018. Improving Language Understanding by Generative Pre-Training. *OpenAI blog*.

[192] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).

[193] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21, 140, 1–67.

[194] Rajput, Shashank et al. 2023. Recommender Systems with Generative Retrieval. In *Proc. NeurIPS*. Retrieved 27th Apr. 2023 from.

[195] Jérémie Rappaz, Julian McAuley and Karl Aberer. 2021. Recommendation on live-streaming platforms: dynamic availability and repeat consumption. In *Proc. RecSys*, 390–399.

[196] Steffen Rendle. 2010. Factorization Machines. In *Proc. ICDM*. 2010 IEEE 10th International Conference on Data Mining (ICDM). Sydney, Australia, (Dec. 2010), 995–1000.

[197] Steffen Rendle. 2022. Item Recommendation from Implicit Feedback. In *Recommender Systems Handbook*, 143–171.

[198] Steffen Rendle and Christoph Freudenthaler. 2014. Improving pairwise learning for item recommendation from implicit feedback. In *Proc. WSDM*. (Feb. 2014).

[199] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proc. UAI*, 452–461.

[200] Steffen Rendle, Christoph Freudenthaler and Lars Schmidt-Thieme. 2010. Factorizing personalized Markov chains for next-basket recommendation. In *Proc. WWW*, 811.

[201] Steffen Rendle, Walid Krichene, Li Zhang and Yehuda Koren. 2022. Revisiting the Performance of iALS on Item Recommendation Benchmarks. In *Proc. RecSys*, 427–435.

[202] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom and John Riedl. 1994. GroupLens: an open architecture for collaborative filtering of netnews. In *Proc. CSCW*. (22nd Oct. 1994), 175–186.

[203] Francesco Ricci, Lior Rokach and Bracha Shapira. 2022. Recommender Systems: Techniques, Applications, and Challenges. In *Recommender Systems Handbook*. Springer US, 1–35.

[204] F. J. Richards. 1959. A Flexible Growth Function for Empirical Use. *Journal of Experimental Botany*, 10, 2, 290–301.

[205] Cornelis J. van Rijsbergen. 1981. *Information Retrieval*. (2. ed., repr ed.). London. 208 pp.

[206] S.E. Robertson. 1977. THE PROBABILITY RANKING PRINCIPLE IN IR. *Journal of Documentation*, 33, 4, (1st Apr. 1977), 294–304.

[207] Stephen Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu and Mike Gatford. 1994. Okapi at TREC-3. In *Proc. TREC*.

[208] F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, 6, 386–408.

[209] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts and Matei Zaharia. 2022. ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. (2022). arXiv: 2112.01488 [cs].

[210] Rodrygo L. T. Santos, Craig Macdonald and Iadh Ounis. 2015. Search Result Diversification. *Foundations and Trends® in Information Retrieval*, 9, 1, 1–90.

[211] Badrul Sarwar, George Karypis, Joseph Konstan and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*. New York, NY, USA, (1st Apr. 2001), 285–295.

[212] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan and Pieter Abbeel. 2015. Trust region policy optimization. In *Proc. ICML*, 1889–1897.

[213] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan and Pieter Abbeel. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *Proc. ICLR*.

[214] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. (2017). arXiv: `1707.06347 [cs]`.

[215] Ozan Sener and Vladlen Koltun. 2018. Multi-task learning as multi-objective optimization. In *Proc. NeurIPS*.

[216] C. E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal*, 27, 3, (July 1948), 379–423.

[217] C. E. Shannon. 1951. Prediction and entropy of printed English. *The Bell System Technical Journal*, 30, 1, (Jan. 1951), 50–64.

[218] Jiayi Shen, Shupeng Gui, Haotao Wang, Jianchao Tan, Zhangyang Wang and Ji Liu. 2021. UMEC: Unified model and embedding compression for efficient recommendation systems. In *Proc. ICLR*.

[219] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov and Jiyan Yang. 2020. Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems. In *Proc. KDD*, 165–175.

[220] Shaoyun Shi, Weizhi Ma, Min Zhang, Yongfeng Zhang, Xinxing Yu, Houzhi Shan, Yiqun Liu and Shaoping Ma. 2020. Beyond User Embedding Matrix: Learning to Hash for Modeling Large-Scale Users in Recommendation. In *Proc. SIGIR*, 319–328.

[221] David Silver et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 7587, 484–489.

[222] Anima Singh et al. 2024. Better Generalization with Semantic IDs: A Case Study in Ranking for Recommendations. In *Proc. RecSys*. New York, NY, USA, 1039–1044.

[223] Harold Soh, Scott Sanner, Madeleine White and Greg Jamieson. 2017. Deep Sequential Recommendation for Personalized Adaptive User Interfaces. In *Proc. UAI*. Limassol Cyprus, 589–593.

[224] Zihan Song, Jiahao Yuan, Xiaoling Wang and Wendi Ji. 2021. Capturing multi-granularity interests with capsule attentive network for sequential recommendation. In *Proc. WISE*, 147–161.

[225] 2024. Spotify/annoy. Spotify. (2024). `https://github.com/spotify/annoy`.

[226] Dusan Stamenkovic, Alexandros Karatzoglou, Ioannis Arapakis, Xin Xin and Kleomenis Katevas. 2022. Choosing the Best of Both Worlds: Diverse and Novel Recommendations through Multi-Objective Reinforcement Learning. In *Proc. WSDM*, 957–965.

[227] Harald Steck. 2019. Embarrassingly Shallow Autoencoders for Sparse Data. In *Proc. WWW*. (13th May 2019), 3251–3257.

[228] A. Stolcke. 1998. Entropy-based Pruning of Backoff Language Models. In *Proc. Broadcast News Tanscription and Understanding Workshop*, 270–274.

[229] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo and Yunfeng Liu. 2024. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomputing*, 568, 127063.

[230] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In *Proc. CIKM*. (Nov. 2019), 1441–1450.

[231] Zhongchuan Sun, Bin Wu, Youwei Wang and Yangdong Ye. 2022. Sequential graph collaborative filtering. *Information Sciences*, 592, 244–260.

[232] Peter Sunehag, Richard Evans, Gabriel Dulac-Arnold, Yori Zwols, Daniel Visentin and Ben Coppin. 2015. Deep Reinforcement Learning with Attention for Slate Markov Decision Processes with High-Dimensional States and Actions. (2015). eprint: `1512.01124` (cs).

[233] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. (Second ed.). The MIT Press.

[234] Panagiotis Symeonidis. 2016. Matrix and Tensor Decomposition in Recommender Systems. In *Proc. RecSys*. Boston Massachusetts USA, (7th Sept. 2016), 429–430.

[235] Gábor Takács and Domonkos Tikk. 2012. Alternating least squares for personalized ranking. In *Proc. RecSys*, 83–90.

[236] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *Proc. WSDM*. (Feb. 2018), 565–573.

[237] Tianchi. IJCAI-16 Brick-and-Mortar Store Recommendation Dataset. `https://tianchi.aliyun.com/dataset/53`.

[238] Nicola Tonellotto and Craig Macdonald. 2020. Using an inverted index synopsis for query latency and performance prediction. *ACM Trans. Inf. Syst.*, 38, 3, (May 2020).

[239] Nicola Tonellotto, Craig Macdonald and Iadh Ounis. [n. d.] Efficient and effective retrieval using selective pruning. In *Proc. WSDM*, 63–72.

[240] Nicola Tonellotto, Craig Macdonald and Iadh Ounis. 2018. Efficient Query Processing for Scalable Web Search. *Foundations and Trends® in Information Retrieval*, 12, 4-5, 319–500.

[241] Nicola Tonellotto, Craig Macdonald and Iadh Ounis. 2018. Efficient query processing for scalable web search. *Foundations and Trends in Information Retrieval*, 12, 4–5, 319–492.

[242] Xiaohai Tong, Pengfei Wang, Chenliang Li, Long Xia and Shaozhang Niu. 2021. Pattern-enhanced contrastive policy learning network for sequential recommendation. In *IJCAI*.

[243] Hugo Touvron et al. 2023. LLaMA: Open and Efficient Foundation Language Models. (2023). arXiv: `2302.13971 [cs]`.

[244] Quyen Tran, Lam Tran, Linh Chu Hai, Ngo Van Linh and Khoat Than. 2022. From implicit to explicit feedback: a deep neural network for modeling sequential behaviours and long-short term preferences of online users. *Neurocomputing*.

[245] Lewis Tunstall, Leandro von Werra and Thomas Wolf. 2022. *Natural Language Processing with Transformers, Revised Edition*. "O'Reilly Media, Inc.", (May 2022).

[246] Howard Turtle and James Flood. 1995. Query evaluation: Strategies and optimizations. *Information Processing & Management*, 31, 6, 831–850.

[247] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser and Illia Polosukhin. 2017. Attention is All you Need. In *Proc. NeurIPS*.

[248] Chenyang Wang, Weizhi Ma, Chong Chen, Min Zhang, Yiqun Liu and Shaoping Ma. 2023. Sequential Recommendation with Multiple Contrast Signals. *ACM Transactions on Information Systems*, 41, 1, 1–27.

[249] Feng Wang, Miaomiao Dai, Xudong Li and Liquan Pan. 2022. Compressing Embedding Table via Multi-dimensional Quantization Encoding for Sequential Recommender Model. In *Proc. ICCIP*, 234–239.

[250] Ruoxi Wang, Bin Fu, Gang Fu and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proc. ADKDD*. New York, NY, USA, (14th Aug. 2017), 1–7.

[251] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong and Ed Chi. 2021. DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems. In *Proc. WWW*. New York, NY, USA, (3rd June 2021), 1785–1797.

[252] Hongxin Wei, Renchunzi Xie, Hao Cheng, Lei Feng, Bo An and Yixuan Li. 2022. Mitigating Neural Network Overconfidence with Logit Normalization. In *Proc. ICML*. (June 2022).

[253] Kilian Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford and Alex Smola. 2010. Feature hashing for large scale multitask learning. (2010). arXiv: 0902.2206[cs].

[254] Jason Weston, Samy Bengio and Nicolas Usunier. 2011. Wsabie: Scaling Up To Large Vocabulary Image Annotation. In *Proc. IJCAI*.

[255] Thomas Wolf et al. 2020. Transformers: state-of-the-art natural language processing. In *Proc. EMNLP*. (Oct. 2020), 38–45.

[256] Haolun Wu, Yansen Zhang, Chen Ma, Fuyuan Lyu, Bowei He, Bhaskar Mitra and Xue Liu. 2024. Result Diversification in Search and Recommendation: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 36, 10, 5354–5373.

[257] Jiancan Wu, Xiang Wang, Xingyu Gao, Jiawei Chen, Hongcheng Fu and Tianyu Qiu. 2024. On the Effectiveness of Sampled Softmax Loss for Item Recommendation. *ACM Transactions on Information Systems*, 42, 4, 1–26. arXiv: 2201.02327 [cs].

[258] Qitian Wu, Chenxiao Yang, Shuodian Yu, Xiaofeng Gao and Guihai Chen. 2021. Seq2bubbles: region-based embedding learning for user behaviors in sequential recommenders. In *Proc. CIKM*, 2160–2169.

[259] Yonghui Wu et al. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. (Oct. 2016). arXiv: 1609.08144 [cs].

[260] Xin Xia, Junliang Yu, Qinyong Wang, Chaoqun Yang, Nguyen Quoc Viet Hung and Hongzhi Yin. 2023. Efficient On-Device Session-Based Recommendation. *ACM Transactions on Information Systems*.

[261] Jiayi Xie, Shang Liu, Gao Cong and Zhenzhong Chen. 2024. UnifiedSSR: A Unified Framework of Sequential Search and Recommendation. In *Proceedings of the ACM on Web Conference 2024*, 3410–3419.

[262] Xu Xie, Fei Sun, Zhaoyang Liu, Shiwen Wu, Jinyang Gao, Jiandong Zhang, Bolin Ding and Bin Cui. 2022. Contrastive Learning for Sequential Recommendation. In *Proc. ICDE*. (May 2022), 1259–1273.

[263] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu and Jimmy Lin. 2020. DeeBERT: dynamic early exiting for accelerating BERT inference. In *Proc. ACL*, 2246–2251.

[264] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis and Joemon M. Jose. 2020. Self-Supervised Reinforcement Learning for Recommender Systems. In *Proc. SIGIR*, 931–940.

[265] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang and Jiajun Chen. 2017. Deep Matrix Factorization Models for Recommender Systems. In *Proc. IJCAI*, 3203–3209.

[266] Jiyuan Yang et al. 2024. Debiasing Sequential Recommenders through Distributionally Robust Optimization over System Exposure. In *Proc. WSDM*, 882–890. arXiv: 2312.07036 [cs].

[267] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov and William W. Cohen. 2018. Breaking the Softmax Bottleneck: A High-Rank RNN Language Model. In *Proc. ICLR*.

[268] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li and Yuxiong He. 2022. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. In *Proc. NeurIPS*. Vol. 35, 27168–27183.

[269] Fajie Yuan, Guibing Guo, Joemon M. Jose, Long Chen, Haitao Yu and Weinan Zhang. 2016. LambdaFM: Learning Optimal Ranking with Factorization Machines Using Lambda Surrogates. In *Proc. CIKM*. (Oct. 2016), 227–236.

[270] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M. Jose and Xiangnan He. 2019. A Simple Convolutional Generative Network for Next Item Recommendation. In *Proc. WSDM*. (Jan. 2019), 582–590.

[271] Zhenrui Yue, Zhankui He, Huimin Zeng and Julian McAuley. 2021. Black-box attacks on sequential recommenders via data-free model extraction. In *Proc. RecSys*.

[272] Zheni Zeng, Chaojun Xiao, Yuan Yao, Ruobing Xie, Zhiyuan Liu, Fen Lin, Leyu Lin and Maosong Sun. 2021. Knowledge transfer via pre-training for recommendation: a review and prospect. *Frontiers in big Data*, 4.

[273] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang and Shaoping Ma. 2021. Jointly Optimizing Query Encoder and Product Quantization to Improve Retrieval Performance. In *Proc. CIKM*, 2487–2496.

[274] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang and Shaoping Ma. 2021. Optimizing dense retrieval model training with hard negatives. In *Proc. SIGIR*, 1503–1512.

[275] Lingxiao Zhang, Jiangpeng Yan, Yujiu Yang and Li Xiu. 2020. Match4rec: a novel recommendation algorithm based on bidirectional encoder representation with the matching task. In *Proc. ICONIP*. Springer, 491–503.

[276] Qihua Zhang, Junning Liu, Yuzhuo Dai, Yiyan Qi, Yifan Yuan, Kunlun Zheng, Fan Huang and Xianfeng Tan. 2022. Multi-Task Fusion via Reinforcement Learning for Long-Term User Satisfaction in Recommender Systems. In *Proc. KDD*, 4510–4520.

[277] Xiaokun Zhang, Bo Xu, Youlin Wu, Yuan Zhong, Hongfei Lin and Fenglong Ma. 2024. FineRec:Exploring Fine-grained Sequential Recommendation. In *Proc. SIGIR*, 1599–1608. arXiv: 2404.12975 [cs].

[278] Yin Zhang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Lichan Hong and Ed H. Chi. 2021. A model of two tales: dual transfer learning framework for improved long-tail item recommendation. In *Proc. WWW*, 2220–2231.

[279] Yixin Zhang, Lizhen Cui, Wei He, Xudong Lu and Shipeng Wang. 2021. Behavioral data assists decisions: exploring the mental representation of digital-self. *International Journal of Crowd Science*.

[280] Pengyu Zhao, Tianxiao Shui, Yuanxing Zhang, Kecheng Xiao and Kaigui Bian. 2021. Adversarial oracular seq2seq learning for sequential recommendation. In *Proc. ICJAI*.

[281] Wayne Xin Zhao et al. 2021. Recbole: towards a unified, comprehensive and efficient framework for recommendation algorithms. In *Proc. CIKM*, 4653–4664.

[282] Chang Zhou, Jianxin Ma, Jianwei Zhang, Jingren Zhou and Hongxia Yang. 2021. Contrastive learning for debiased candidate generation in large-scale recommender systems. In *Proc. KDD*, 3985–3995.

[283] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang and Ji-Rong Wen. 2020. S3-Rec: Self-Supervised Learning for Sequential Recommendation with Mutual Information Maximization. In *Proc. CIKM*. (Oct. 2020), 1893–1902.

[284] Peilin Zhou, You-Liang Huang, Yueqi Xie, Jingqi Gao, Shoujin Wang, Jae Boum Kim and Sunghun Kim. 2024. Is Contrastive Learning Necessary? A Study of Data Augmentation vs Contrastive Learning in Sequential Recommendation. In *Proceedings of the ACM on Web Conference 2024*, 3854–3863.

[285] Tao Zhou, Ri-Qi Su, Run-Ran Liu, Luo-Luo Jiang, Bing-Hong Wang and Yi-Cheng Zhang. 2009. Accurate and diverse recommendations via eliminating redundant correlations. *New Journal of Physics*, 11, 12.

[286] Xun Zhou, Jing He, Guangyan Huang and Yanchun Zhang. 2012. A personalized recommendation algorithm based on approximating the singular value decomposition (ApproSVD). In *Proc. WI-IAT*. Vol. 2, 458–464.

[287] Ziwei Zhu, Jianling Wang and James Caverlee. 2019. Improving Top-K Recommendation via Joint Collaborative Autoencoders. In *Proc. WWW*. (May 2019).

[288] Shlomo Zilberstein. 1996. Using anytime algorithms in intelligent systems. *AI Magazine*, 17, 3, 73.

[289] Andrew Zimdars, David Maxwell Chickering and Christopher Meek. 2001. Using temporal data for making recommendations. In *Proc. UAI*, 580–588.

[290] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu and Dawei Yin. 2019. Reinforcement Learning to Optimize Long-term User Engagement in Recommender Systems. In *Proc. KDD*, 2810–2818.