Generative Sequential Recommendation with GPTRec

Aleksandr V. Petrov
University of Glasgow
United Kingdom
a.petrov.1@research.gla.ac.uk

Craig Macdonald
University of Glasgow
United Kingdom
craig.macdonald@glasgow.ac.uk

ABSTRACT

Sequential recommendation is an important recommendation task that aims to predict the next item in a sequence. Recently, adaptations of language models, particularly Transformer-based models such as SASRec and BERT4Rec, have achieved state-of-the-art results in sequential recommendation. In these models, item ids replace tokens in the original language models. However, this approach has limitations. First, the vocabulary of item ids may be many times larger than in language models. Second, the classical Top-K recommendation approach used by these models may not be optimal for complex recommendation objectives, including auxiliary objectives such as diversity, coverage or coherence. Recent progress in generative language models inspires us to revisit generative approaches to address these challenges. This paper presents the GPTRec sequential recommendation model, which is based on the GPT-2 architecture. GPTRec can address large vocabulary issues by splitting item ids into sub-id tokens using a novel SVD Tokenisation algorithm based on quantised item embeddings from a SVD decomposition of the user-item interaction matrix. The paper also presents a novel Next-K recommendation strategy, which generates recommendations item-by-item, considering already recommended items. The Next-K strategy can be used for producing complex interdependent recommendation lists. We experiment with GPTRec on the MovieLens-1M dataset and show that using sub-item tokenisation GPTRec can match the quality of SASRec while reducing the embedding table by 40%. We also show that the recommendations generated by GPTRec on MovieLens-1M using the Next-K recommendation strategy match the quality of SASRec in terms of NDCG@10, meaning that the model can serve as a strong starting point for future research.

1 INTRODUCTION

Sequential recommender systems are a class of recommender systems that use the order of user-to-item interactions. These systems are beneficial when the nature of user-item interactions includes sequential patterns. For example, it is natural to watch movies in a particular order (from the first to the last movie in a series). Usually, the sequential recommendation task is cast as a next-item prediction task: the goal of the recommender system is to predict the next item in the sequence of user-item interactions. Most of the recent results in sequential recommendation were achieved by deep learning models [11, 28, 34], and in particular on Transformer [30] architecture [13, 17–19, 26]. Most of the Transformer-based recommender models, such as SASRec [13] and BERT4Rec [26] adapt original language models to recommendation field by using item ids instead of tokens in the language model architecture.

The rapid progress of large generative models, such as T5 [22] and the GPT family [1, 16, 20, 21] 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 [8]. Similarly, in a search task, Tay et al. showed that generative models can replace the traditional reverse index-based retrieval and directly generate relevant document ids by a given query. This progress motivates us to re-examine the role generative of models for the sequential recommendation task. This paper discusses some of the challenges in sequential recommendation and how these challenges can be addressed using generative models. We identify and discuss two main challenges and discuss the ramifications of our findings for other related information retrieval (IR) tasks.

First, existing sequential models usually employ a score-and-rank approach to generate recommendations (we also call this the *Top-K strategy*). The problem with this method is that items are scored independently, and similar items are likely to have similar scores. In this case, 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 a user different types of items. Instead, we propose a generative *Next-K strategy*, where recommendations are generated item-by-item. In that case, when the model generates a recommendation in position K, it already knows what items are recommended in positions $1..K-1$, and may adjust the result accordingly. We discuss the details of these recommendation generation strategies in Section 3.

Second, a substantial contribution to the size of existing recommendation models is the item embeddings table. For example, in a dataset with 10M items and 256-dimensional item embeddings, the embedding table in a typical recommendation model would require more than 10GB of GPU memory only for storing embeddings (even without considering gradients and model weights). When we consider gradients, model weights and intermediate scores, GPU memory requirements increase even more, making model training infeasible. Realistically, training state-of-the-art sequential recommender models, such as BERT4Rec [26], is problematic when there are more than 1M items in the catalogue (see also [17]). We show that this problem can be solved by splitting item ids into sub-item *tokens* using the proposed SVD Tokenisation approach. To obtain sub-item tokens, SVD Tokenisation quantises item embeddings obtained from the SVD decomposition of the user-item interaction

Gen-IR@SIGIR2023, July 27, 2023, Taipei, Taiwan

matrix. Then, the items can be generated token-by-token (similar to how words are generated from sub-word tokens in texts). Section 4.2 covers the details of the SVD Tokenisation algorithm.

To show the feasibility of solving both challenges, we build a generative GPTRec recommendation model based on the GPT-2 [21] architecture. Architecturally, GPTRec is similar to the existing SASRec model [13]. However, GPTRec supports the Next-K recommendation strategy, allowing for more flexible objectives. In addition, GPTRec can work with sub-item tokenisation, which is beneficial for GPU memory consumption. GPTRec also uses a different loss function compared with SASRec (Cross-Entropy instead of Binary Cross Entropy); empirically, we show that this change is beneficial. We evaluate GPTRec on the MovieLens-1M dataset [10] and show the viability of generative techniques for solving the discussed challenges. In particular, we observe that although we do not specifically tune GPTRec for Next-K generation, it achieves a strong ranking quality similar to SASRec (0.105 NDCG@10 for GPTRec-NextK vs. 0.108 SASrec). We also observe that using SVD Tokenisation with GPTRec can reduce the embedding table size by 40%, while achieving comparable ranking quality with SASRec (both have NDCG@10 of 0.108). In short, we summarise the contributions of this paper 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 achieve results similar to those of SASRec.
2. We propose a novel SVD-based sub-item item tokenisation and token-by-token item generation approach. We demonstrate the potential to reduce the embedding table size. We also show that our approach can achieve similar results to SASRec while requiring less storage for embedding tables.
3. We propose GPTRec, a generative sequential recommendation model based on the GPT-2 architecture. We show that it achieves results similar to transformer-based models (better than SASRec, and comparable with BERT4Rec). However, in contrast with BERT4Rec, GPTRec is a generative model, uses SVD Tokenisation for memory efficiency, and more flexible using the Next-K generation strategy.
4. We identify the limitations of proposed approaches and identify strategies to overcome these limitations in future work.

The rest of the paper is organised as follows: Section 2 covers existing work related to generative sequential recommendation; Section 3 discusses limitations of existing Top-K recommendation strategies and proposes Next-K as an alternative; in Section 4 we introduce GPTRec; Section 5 compares GPTRec with existing Transformer-based models; Section 6 covers the experimental evaluation of GPTRec; Section 7 contains directions for further work, ramifications for related IR tasks and concluding remarks.

2 RELATED WORK

This section covers existing work related to the generative sequential recommendation: Section 2.1 discusses adaptations of language models for sequential recommendation; Section 2.2 covers the recent line of work of using pre-trained language models for recommendation task.

2.1 Adaptations of Language Models for Sequential Recommender Systems

Natural Language Processing (NLP) is a prominent and established research field focusing on text-related tasks, such as question answering, machine translation or text summarisation. A text document consists of a sequence of words, and models that work with text have to be able to work with these sequences. Sequences of user-item interactions in sequential recommender systems have a very similar structure to the sequences of words in texts. Therefore recommender systems researchers frequently adapt models originally developed for language processing to the sequential recommendation task. For example, GRU4Rec [11], one of the first neural architectures for the sequential recommendation, is based on the GRU model [4], originally designed for machine translation tasks.

Recently, many recommendation models have adopted variations of the Transformer [30] architecture, which was also originally designed for addressing machine translation. For example, two of the most popular Transformer-based sequential recommenders are (i) SASRec [13], which uses the decoder part of the Transformer model and is trained to shift the input sequence of items one element to the left (in language models, this task is also known as the Language Modelling task, LM) and (ii) BERT4Rec [26], which, in contrast, is trained to recover masked items from the sequence (this is known as Masked Language Modelling, MLM).

Other more recent Transformer-based sequential models include DuoRec [19], CBiT [7], ALBERT4Rec [18] and many others. While these works slightly change the architecture or augment the model with additional training tasks, they all share a Transformer encoder or a Transformer decoder as the main component of their architecture. Despite achieving state-of-the-art results in ranking metrics, such as NDCG@K, the Top-K recommendation strategy employed by these models is not flexible enough to optimise for other metrics such as diversity or serendipity; these models also suffer from high GPU requirements when the number of items in the catalogue is high. These problems and the recent advances in generative models motivate us to apply a generative approach for sequential recommendation.

2.2 Recommendations as Text Generation

The arrival of Large Language Models, such as T5 [22] and GPT-3 [1], 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 [8] uses text generation to generate item and user ids directly as text strings. The M6-Rec [5] model directly generates item titles as recommendations. TIGER [23] introduces the idea of *semantic id*, where the item is represented as a set of tokens derived from its side information, such as product category. While using pre-trained models for recommendation is an interesting research direction, it differs from ours. Indeed, these models rely on the existence of 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, we do not rely on any

side information and research a more classical sequential recommendation setting where the only data available to the model is the interaction sequences. This allows us to understand the properties of the generative approach better and decouple it from the benefits of the availability of side information.

While this paper focuses on the sequential recommendation task, the discussed problems are relevant for a broader field of Generative IR. Indeed, in the Generative IR tasks, which directly generate document ids (docids) for a given query, a large number of docids (the equivalent of items) and the techniques for splitting atomic document ids into tokens are frequently mentioned as one of the central problems in the recent Generative IR publications [29, 36]. Some of the recent papers have specifically focused on sub-docid tokenisation [15, 27] and inspire us to develop a similar technique for recommender systems. SVD-based tokenisation (or, more broadly, embedding-based tokenisation) described in this paper can be adapted to the IR domain and help to solve scaling issues in the IR. Complex interdependent search results are also an active area of IR research, including, for example, results diversity [33] and fairness [12]. The Next-K generation strategy proposed in this paper can help to resolve these issues.

In summary, existing Transformer-based sequential recommendation models achieve high quality in ranking metrics, but they are inflexible to optimise for other metrics and have high GPU memory requirements. On the other hand, recent works show the potential of pre-trained language models for the sequential recommendation problem. However, they rely on side information and pre-existing knowledge, which is not always available or reliable. To address the objective flexibility problem, in the next section, we propose a Next-K recommendation strategy as an alternative to Top-K, specifically designed to work well with generative models. We then describe GPTRec in Section 4, which can utilise the Next-K strategy and may use SVD Tokenisation to reduce GPU memory consumption.

3 TOP-K AND NEXT-K RECOMMENDATIONS

In this section, we discuss the classical Top-K recommendation approach (used by BERT4Rec and SASRec) and its fundamental limitations and propose a novel alternative Next-K approach for generative models.

3.1 Top-K strategy

The ultimate goal of a recommender system is to provide a user with a list of items which are likely to be of user interest. The most typical approach for solving this is the *Top-K* recommendation strategy [14, 37]. In this strategy, the recommendation model $f(user, item) \rightarrow s$ returns an estimated relevance score s for each $(user, item)$ pair (in the case of sequential recommendation, $user$ is actually represented as a sequence of interactions). To generate the recommendation for a particular user u, the recommender system uses model f to generate all item relevance scores and then selects k items with the highest scores as the recommendations. Algorithm 1 illustrates this strategy using pseudo-code.

The fundamental problem with the Top-K strategy is that it assumes that every item can be scored independently from the other recommended items. However, the Top-K strategy fails when the recommendations should be interdependent. For example, there is a redundancy problem: if a user has recently bought a coffee machine, they will likely buy coffee beans as their next purchase. Therefore, it is a good strategy to recommend coffee beans to the user. However, there can be many different variants of coffee beans on sale. Each variant will likely have a high recommendation score, so most 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 [35]. Diversification re-ranking methods, such as MMR [3], and multi-aspect methods [2] 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 [24]. Instead of re-ranking approaches, training a single model with an interdependent objective can provide a more efficient solution.

Algorithm 1 Top-K Recommendation Strategy

Require: $user$ is the target user for recommendations
Require: $items$ is the set of all available items
Require: K is the number of recommendations to generate
Require: f is an existing scoring model

```
procedure TopKRecommend(user, items, K, f)
    item_scores ← empty list
    for item ∈ items do
        score ← f(user, item)
        item_scores.append((item, score))
    end for
    sorted_items ← sorted(item_scores,
            key = score, reverse = True)
    recommended_items ← [item
             for (item, score) in sorted_items[: K]]
    return recommended_items
end procedure
```

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 which fit well with each other), serendipity (sometimes we want to help users discover new items, but only if we already recommended items of their interest), and coverage (we may want to be sure that our recommendations provide good coverage of the system's catalogue) etc. We now introduce the Next-K recommendation strategy, capable to resolve these problems.

3.2 Next-K strategy

Next-K is the recommendation strategy, where the recommender system decides what to recommend at position k only after generating recommendations at position $1..k-1$. More formally, the item score depends not only on the user and item but also recommendations on previous positions: $f(user, recommended\_items, item) \rightarrow score$. The model iteratively scores all items (excluding already recommended) to generate recommendations and adds the highest-scored item to the $recommended\_items$ list. Algorithm 2 illustrates the strategy in pseudo-code. Because the Next-K strategy considers already recommended items, it may address the issues of the classic

Algorithm 2 Next-K Recommendation Strategy

Require: $user$ is the target user for recommendations
Require: $items$ is the set of all available items
Require: K is the number of recommendations to generate
Require: f is an existing scoring model
procedure NextKRecommend($user, items, K, f$)
  $recommended\_items \leftarrow$ empty list
  for $k = 1$ to K **do**
    $max\_score \leftarrow -\infty$
    $best\_item \leftarrow$ None
    for $item \in items \setminus recommended\_items$ **do**
      $score \leftarrow f(user, recommended\_items, item)$
      if $score > max\_score$ **then**
        $max\_score \leftarrow score$
        $best\_item \leftarrow item$
      end if
    end for
    $recommended\_items$.append($best\_item$)
  end for
  return $recommended\_items$
end procedure

Top-K strategy, such as lack of diversity and poor representation of user interests.

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. Another problem is that the scoring function now has more input arguments (i.e. in addition to the user id and the item, it takes into account already recommended items), so training such a function may be challenging.

In summary, in this section, we described a popular Top-K recommendation strategy and identified the limitations of this strategy. We also proposed the Next-K strategy as a solution to the Top-K strategy problems and also identified the limitations of this strategy. In the next section, we introduce the GPTRec model, which can be used with both Top-K and Next-K strategies. We now turn to GPTRec, a generative recommendation model that can use either Top-K or Next-K recommendation strategies.

4 GPTRec

In this section, we introduce GPTRec, a generative sequential recommendation model we use as a backbone for our experiments.

4.1 Architecture

GPTRec,[1] designed for the generative sequential recommendation, utilises the GPT-2 architecture [21], which in turn is based on the decoder part of the Transformer model [30]. 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;
- 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 [21, 30].

4.2 Tokenisation

Similarly to existing Transformer-based sequential recommendation models such as SASRec [13] and BERT4Rec [26], GPTRec can use item ids instead of tokens in the original GPT model. We call this *token-per-item mode*. However, as discussed in Section 1, when the number of items in the catalogue is large, this approach leads to a massive embedding table, which is hard to fit into a single GPU.

Language models, such as BERT [6] and GPT-2 [21] solve a similar problem by splitting whole words into sub-word tokens. Example strategies for sub-word tokenisation include Word Piece encoding [32] (used by BERT) and BytePair encoding [25] (used by GPT family). Inspired by this idea, GPTRec can also split items into sub-items to reduce memory footprint. In this case, each item is represented using t tokens, where each of the t tokens can be chosen from v alternatives. We call this *multi-token-per-item mode*.

As a simple heuristic to decompose items into sub-item tokens, we propose an SVD-based Tokenisation algorithm. Figure 1 illustrates the main three steps of the algorithm.

First (Step 1 of the figure), the SVD Tokenisation algorithm builds a user-item interaction matrix M and performs a truncated SVD decomposition of this matrix with t latent components:

$$M \approx U \times \Sigma \times E^T \tag{1}$$

where U is the matrix of user embeddings, E is the matrix of item embeddings, and Σ is the diagonal matrix with t largest singular eigenvalues on the diagonal. Both user and item embeddings have t latent components.

In Step 2 of Figure 1, the item embeddings E are normalised, so that each embedding dimension is in the [0..1] interval and a small amount of Gaussian noise is added to ensure that no two items have equal embeddings (equal embeddings may happen if exactly the same users interacted with two items). In our experiments, we use Gaussian noise with zero mean and a standard deviation of 10^{-5}, which is several orders of magnitude less than the scale of the embedding values themselves (they are in the [0..1] range after normalisation). After that, SVD Tokenisation quantises each dimension of item embeddings into v values; each value in the quantised embeddings represents one token in the final representation of the item.

Finally, (Step 3 of Figure 1) the algorithm offsets i^{th} dimension of the quantised embeddings by $v * (i - 1)$ to ensure that every dimension of the item representation has its range of tokens (i.e. the first dimension is represented by tokens $[0 \,..\, v - 1]$, the second dimension is represented by tokens $[v \,..\, 2v - 1]$ and so on.

Algorithm 3 illustrates this procedure in pseudo-code. In this algorithm, tokenised items are, in fact, discretised embeddings from an SVD matrix decomposition of the user-matrix interactions matrix. Similar items are likely to have similar embeddings and, therefore, similar multi-token representations. As for memory consumption, the model now only needs to store $t \times v$ embeddings rather

[1] GPT stands for Generative Pre-trained Transformer. In fact, in our adaptation, we don't use the pre-trained versions of GPT, but we saved the letter P in the model name in order to give credit to the GPT authors.

Table 1: GPU memory requirements for the multi-token-per-item model, as a percentage of the one-token-per-item model, for selected datasets. t corresponds to the number of tokens per item, and v corresponds to the number of possible items per token in the SVD Tokenisation algorithm. The number of embeddings and GPU memory requirements in each multi-token-per-item configuration is shown in the brackets. We assume that each embedding is stored as 256 float32 numbers.

Dataset	Num Items	GPU Memory (one-token-per-item mode)	t=2 v=128 (256 embs; 0.25 MB)	t=2 v=512 (1024 embs; 1.00 MB)	t=2 v=2048 (4096 embs; 4.00 MB)	t=4 v=128 (512 embs; 0.50 MB)	t=4 v=512 (2048 embs; 2.00 MB)	t=4 v=2048 (8192 embs; 8.00 MB)	t=8 v=128 (1024 embs; 1.00 MB)	t=8 v=512 (4096 embs; 4.00 MB)	t=8 v=2048 (16384 embs; 16.00 MB)
MovieLens-1M	3,416	3.34 MB	7.494%	29.977%	119.906%	14.988%	59.953%	239.813%	29.977%	119.906%	479.625%
MovieLens-20M	138,493	135.25 MB	0.185%	0.739%	2.958%	0.370%	1.479%	5.915%	0.739%	2.958%	11.830%
Yelp	150,346	146.82 MB	0.170%	0.681%	2.724%	0.341%	1.362%	5.449%	0.681%	2.724%	10.898%
Gowalla	1,280,969	1.22 GB	0.020%	0.080%	0.320%	0.040%	0.160%	0.640%	0.080%	0.320%	1.279%
Amazon Books	5,264,307	5.02 GB	0.005%	0.019%	0.078%	0.010%	0.039%	0.156%	0.019%	0.078%	0.311%
LastFM-1b	32,291,134	30.80 GB	0.001%	0.003%	0.013%	0.002%	0.006%	0.025%	0.003%	0.013%	0.051%

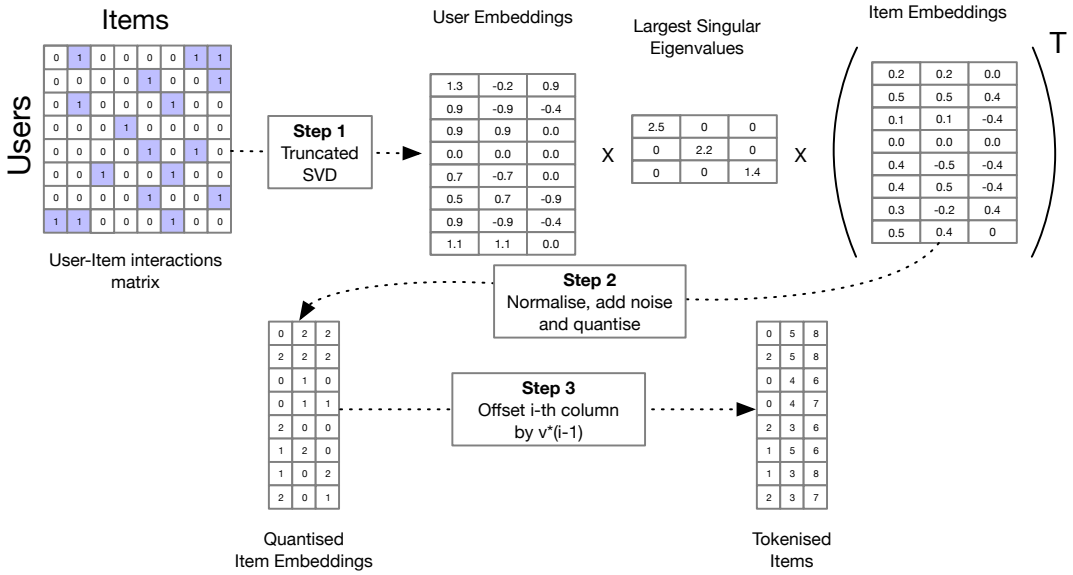


Figure 1: Step-by-step example of the SVD Tokenisation algorithm

than storing one embedding per item. Therefore, the amount of required GPU memory for storing embeddings does not depend on the total number of items in the catalogue; this is similar to how fixed-size token vocabulary can represent millions of different words by combining different tokens in language models. For example, as we mentioned earlier, storing the embedding table in the one-token-per-item mode for 10M items requires more than 10GB of GPU memory. However, embedding a table for the same dataset in multi-token-per-item mode with $t = 8$ and $v = 2048$ requires only 16MB GPU Memory (only 0.16% of the original embedding table size). Table 1 illustrates similar calculations for several available recommender systems datasets and different configurations of the SVD Tokenisation algorithm.

Algorithm 3 SVD Tokenisation Algorithm

Require: M is the user-item interaction matrix
Require: t is the number of tokens per item
Require: v is the number of alternatives per token
Ensure: Sub-item token representation of items

```
procedure SVDTokenise(M, t, v)
    Compute item embeddings E with truncated SVD on M
    for i = 1 to t do
        Normalise E_i to range [0, 1]
        Add small Gaussian noise to E_i
        Quantise E_i into v bins
        Offset quantised values by (i − 1) ∗ v
    end for
    Concatenate quantised components for sub-item tokens
    return Sub-item token representation
end procedure
```

4.3 Training Objective and Loss Function

Following GPT-2 [21], we train the model using the Language[2] Modelling objective with Cross-Entropy loss. The language modelling approach factorises the probability of a sequence of tokens $S = \{s_1, s_2, ...s_n\}$ (recall that in GPTRec, s_i represents an item or a sub-item token) as the product of conditional probabilities:

$$p(S) = \prod_{i=1}^{n} p(s_i|s_1, s_2, ...s_{i-1}) \tag{2}$$

The loss function is then derived from Equation (2) using the Maximum Log-likelihood principle [9, Ch.5]:

$$\mathcal{L}_{LM} = -\frac{1}{n}\sum_{i=1}^{n} \log(p(s_i|s_1, s_2, ...s_{i-1})) \tag{3}$$

Similarly to GPT-2, in GPTRec, $p(s_i|s_1, s_2, ...s_{i-1})$ is modelled as softmax$(\cdot)$ operation over i^{th} output of the model, so overall the model is trained to shift its input one token to the left. The last token predicted by the model is a new token that did not appear in the input sequence. This shifted sequence can be fed back into the model, which enables us to implement the Next-K recommendation approach for increased flexibility and utilise the multi-token-per-item mode for improved GPU efficiency.

In short, GPTRec is trained with standard Language modelling (LM) training objective, however, instead of token ids, we use item ids or sub-id values obtained from quantised item embeddings. We now turn to the generation of recommendations.

4.4 Generating Recommendations with GPTRec

In this section, we describe possible options to generate a list of recommendations using GPTRec.

4.4.1 Top-K generation, one-token-per-item. GPTRec supports the Top-K recommendation strategy. In the simplest case, when one item corresponds to one token, the generation of recommendations is similar to that of SASRec. GPTRec outputs the probability distribution $p(s_i|s_1, s_2, ..s_{i-1})$ of the next token for each position in the sequence, meaning that the last probability distribution corresponds to the next most likely item in the sequence. In this generation mode, GPTRec uses this last probability distribution as item scores and applies standard Top-K scoring (see Section 3.1 and Algorithm 1).

4.4.2 Top-K generation, multi-token-per-item. The next generation strategy supported by GPTRec is *Top-K generation with multi-token-per-item*. In this case, the model outputs K candidate items using standard GPT-2 autoregressive generation: each time, the model predicts the probability distribution of the next token and then samples a token from the distribution. This token is added to the end of the input sequence, and the procedure repeats until the model generates t tokens, where t is the number of tokens in each item representation. After generating the candidate, the model scores the candidate using the chain rule described by Equation (2). Finally, the model applies the standard Top-K strategy Algorithm 1 to the set of items, assuming that any item that was not generated as a candidate has $-\infty$ score.

Note that using this procedure, some generated sequences may not correspond to any valid item id; the model ignores these invalid sequences. Some of the generated candidates may also repeat. In practice, this means that the number of candidates K that has to be chosen is relatively high compared to the required number of recommended items; For example, we use $K = 50$ even though we only needed ten items to generate recommendations.

4.4.3 Next-K generation, one-token-per-item. In this case, the recommendation list is generated using the Next-K procedure (Algorithm 2). At each iteration, the model concatenates the sequence of the user's interactions with the sequence of already generated recommendations and generates the next most likely item. Next-K strategy is also equivalent to the *greedy search* token generation strategy in language models.[3]

Note that the training objective, in this case, is somewhat misaligned with the generation procedure. Indeed, the model is trained to predict one next token and not K next tokens. Therefore we expect quality degradation with increased K. To align the training procedure with the Next-K generation procedure, the model requires more advanced training objectives, such as reinforcement learning techniques described in the InstructGPT [16] paper. Nevertheless, these techniques require a model that can already generate some plausible sequences. Therefore, in this paper, we examine if a model trained for Top-K generation can return meaningful recommendations in the Next-K mode, and leave reinforcement learning fine-tuning for future work. This concludes the description of the GPTRec model. In the next section, we compare GPTRec with existing SASRec and BERT4Rec models before moving to experimental analysis of GPTRec in Section 6.

5 COMPARISON OF GPTRec WITH SASRec AND BERT4Rec

In this section, we compare GPTRec with the two most popular transformer-based sequential models, namely with SASRec [13] and BERT4Rec [26]. SASRec [13] is the most similar model to GPTRec. Indeed, both models use the decoder part of the Transformer [30] architecture and are trained to shift the input sequence one element to the left. However, GPTRec differs from SASRec by using the Cross-Entropy loss (aka Softmax Loss), while SASRec is trained using Binary Cross Entropy loss. The use of Cross-Entropy loss is directly derived from the Maximum Log likelihood principle. In contrast, Binary Cross-Entropy used by SASRec is a heuristic, which, as we show in Section 6, underperforms compared to Cross-Entropy loss. BERT4Rec, on the other hand, uses Cross-Entropy loss, but it uses a Masked Language Modelling task (MLM; also known as Cloze, or Item Masking), which is less aligned with the next item prediction task [18]. BERT4Rec also differs from GPTRec by using the encoder part of the Transformer, whereas SASRec and GPTRec use the decoder part.

Second, SASRec only supports Top-K recommendations in one-token-per-item mode. In contrast, GPTRec also supports a multi-token-per-item mode to reduce memory footprint and a Next-K

[2]Even though our models do not work with texts, we retain the "Language" in the name of the objective to stay consistent with the NLP literature that serves as a source of inspiration for us.

[3]To use greedy search (the equivalent of the Next-K strategy) in HuggingFace Transformers library, one can call the `.generate()` method with parameters `num_beams=1` and `do_sample=False`.
See also https://huggingface.co/docs/transformers/v4.30.0/en/generation_strategies#greedy-search.

recommendation strategy, which can be used for learning more advanced recommendation objectives using techniques such as reinforcement learning.

In summary, compared with existing models, GPTRec is more GPU-memory efficient when trained in multi-token-per-item mode. It also allows for more flexible recommendation generation using the Next-K generation strategy. The left side of Table 3 also portrays the salient characteristics of these models. In the next section, we evaluate GPTRec experimentally, compare it with SASRec and BERT4Rec, and experimentally study its performance in multi-token-per-item and Next-K generation modes.

6 EXPERIMENTS

This section covers the experimental evaluation of GPTRec. We aim to answer the following research questions:

RQ1 How does GPTRec perform in one-token-per-id mode compared to BERT4Rec and SASRec?
RQ2 What are the effects of the number of tokens and the number of values per token in the multi-token-per-item mode in GPTRec?
RQ3 What is the effect of cutoff level K on the GPTRec performance in the Next-K recommendation mode?

6.1 Experimental Setup

We implement GPTRec using the GPT-2 architecture from HuggingFace Transformers [31] and the aprec[4] framework from a recent replicability study [18]. We follow the experimental setup described in the same replicability study [18] and use SASRec's and BERT4Rec's results reported in the paper as baselines.

We use the MovieLens-1M dataset [10] for the experiments. While there are some known issues with the dataset (e.g. users don't rate movies in the dataset in the same order as they watch them), the dataset is used consistently across many papers, allowing compare the published results across papers. The salient characteristics of the dataset are listed in Table 2.

We employ a leave-one-out strategy for model testing, holding out each user's latest interaction in the test set. In addition, for 128 randomly selected users, we hold out their second-to-last action in a separate validation set. This validation set is used for the early stopping mechanism and to control model quality during training, ensuring that our model generalises well to new data.

Following common practice [13, 18, 26] we use a relatively shallow transformer model with three transformer blocks. We set the maximum sequence length to 100 and truncate the sequence to the last 100 interactions if the user interacted with more items. We use 256-dimensional token embeddings in our experiments. We use an early stopping mechanism that terminates training if validation NDCG@10 does not improve for 300 epochs.

6.2 RQ1. Comparison with existing transformer-based models

We first evaluate the performance of GPTRec in comparison with popular Transformer-based sequential recommendation models. For this comparison, we choose BERT4Rec [26] and SASRec [13] as the baselines and compare them with two variations of the GPTRec model: GPTRec-TopK (GPTRec with the Top-K recommendation generation strategy) and GPTRec-NextK (GPTRec with the Next-K generation strategy). In this experiment, we use all models, including GPTRec, in the one-token-per-item mode. Table 3 reports the result of our comparison.

Table 2: Salient characteristics of the MovieLens-1M dataset

Number of users	6040
Number of items	3416
Number of interactions	999611
Average sequence length	165.49
Median sequence length	96

The table shows that the results achieved by GPTRec-TopK are similar to the results of SASRec and BERT4Rec: e.g. it achieves NDCG@10 of 0.146, which is better than SASRec's result (0.108, +35%) but comparable to BERT4Rec's result (0.152, -4%). The results measured by the Recall@10 metric also follow the same pattern: GPTRec achieves Recall@10 of 0.254, slightly worse than BERT4Rec's result (0.282) but better than SASRec's result (0.199).

Overall we can say that the results of GPTRec-TopK are comparable with BERT4Rec and better than SASRec.

The observation that GPTRec-TopK performs better than SASRec is specifically interesting. As discussed in Section 5, when GPTRec is used in one-token-per-item mode with Top-K generation strategy, its main difference with SASRec is the loss function (Cross-Entropy in GPTRec vs. Binary Cross-Entropy in SASRec). This confirms that Cross-Entropy loss is better for the next-item recommendation.

From Table 3, we also see that GPTRec-NextK performs worse than other models. This is not surprising: in Section 4.4.3, we discussed that the model would likely require more complex tuning techniques, such as reinforcement learning, to perform well in the Next-K generation mode. However, as can be seen from the table, the NDCG@10 achieved by the model is very similar to SASRec's, which means that the model can serve as a strong starting point for further tuning, for example using reinforcement learning[5].

Overall in answer to RQ1, we conclude that GPTRec-TopK achieves similar results to BERT4Rec and outperforms SASRec. On the other hand, GPTRec-NextK performs worse than other models but is still comparable with SASRec's results and may serve as a starting point for future research.

6.3 RQ2. Effect of the number of tokens and the number of tokens per item in multi-token-per-item mode

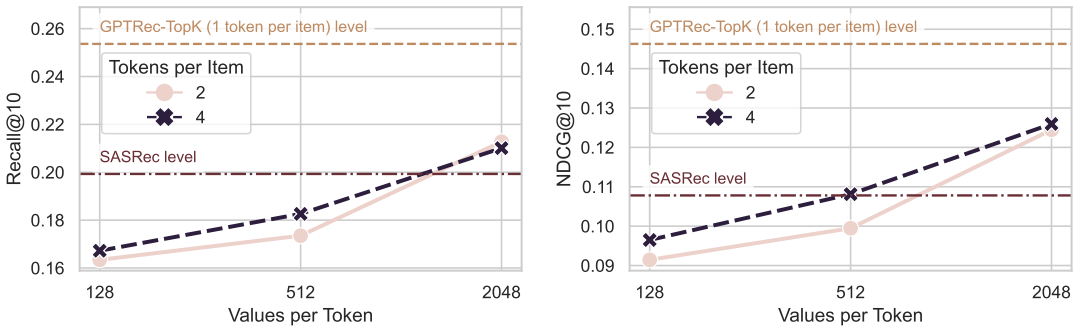
To answer our second research question, we train GPTRec in multi-token-per-item mode. To split item ids into sub-item tokens, we use the SVD Tokenisation algorithm, as described in Section 4.2. We select the number of tokens per item from {2, 4} and the number of values per token from {128, 512, 2048}.

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

[5] Reinforcement learning is needed because, in the Next-K strategy, we can only compute our utility function (e.g. NDCG) after conducting inference using the model K times. Therefore, it is hard to optimise the model using classic gradient descent, which assumes that it is possible to compute the loss function after every model inference.

Table 3: Model performances on the MovieLens-1M dataset. Bold values indicate the best results, and underlined values indicate the second best results. SASRec and BERT4Rec results are copied from the reproducibility paper [18].

Model name	Generation Strategy	Architecture	Training Task	Loss	Recall@10	NDCG@10
BERT4Rec	TopK	Encoder	MLM (Item Masking)	Cross Entropy (Softmax Loss)	**0.282**	**0.152**
GPTRec-TopK	TopK	Decoder	LM (Sequence Shifting)	Cross Entropy (Softmax Loss)	0.254	0.146
SASRec	TopK	Decoder	LM (Sequence Shifting)	Binary Cross-Entropy	0.199	0.108
GPTRec-NextK	NextK	Decoder	LM (Sequence Shifting)	Cross Entropy (Softmax Loss)	0.157	0.105



(a) Recall@10 **(b) NDCG@10**

Figure 2: GPTRec performance in multi-token-per-item mode. The figure also includes the performance of SASRec for comparison, as well as GPTRec in one-token-per-item mode.

Figure 2 provides the Recall@10 and NDCG@10 metrics of these models and compares them with GPTRec-TopK trained in one-token-per-item mode and SASRec. The figure shows that the model's performance degrades when compared to GPTRec-TopK in one-token-per-item mode but remains competitive with SASRec. For example, with four tokens per item and 512 values per token GPTRec almost exactly matches the performance of SASRec in terms of NDCG@10 (0.108 for both models) but is worse when measured on Recall@10 (0.182 for GPTRec with four tokens and 0.199 for SASRec). However, this is substantially worse than GPTRec-TopK in one-token-per-item, which achieves NDCG@10 of 0.253. On the other hand, multi-token-per-item mode uses less GPU memory. For example, with four tokens per item and 512 values per token, the model only needs to store 2048 embeddings, 40% less compared to the one-token-per-item mode (see also Table 1).

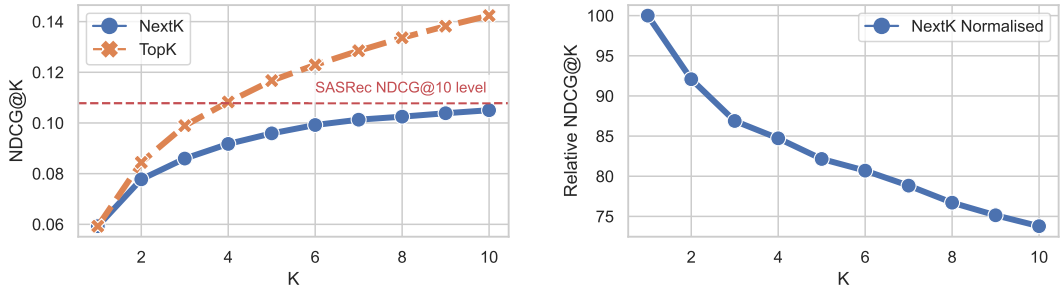
Figure 2 shows that both metrics increase when the discretisation (number of possible values per token) increases. For example, with two tokens per item and 128 values per token, the model only reaches NDCG@10 of 0.0914. When the number of values per token increases to 2048, NDCG@10 increases to 0.124, thereby outperforming SASRec. From the figure, we also see that four tokens per item are better in most cases than two tokens per item, except for the case with 2048 values per token – in this case, the models demonstrate equal performance.

Overall in answer to RQ2, we say that GPTRec in multi-token-per-item mode achieves comparable results with SASRec, but worse than GPTRec in one-token-per-item mode. However, in muti-token-per-item mode, it consumes less GPU memory, which is important when working with larger datasets.

6.4 RQ3. Effect of cutoff K in Next-K generation

We now compare GPTRec performance with Top-K and Next-K recommendation strategies in a one-token-per-item mode to answer our last research question. Figure 3 compares these two-generation strategies: Figure 3a shows the absolute value of the NDCG@K metric at different cutoffs, whereas Figure 3b shows the relative value of NDCG@K metric of the Next-K strategy, using Top-K as a baseline. On analysis of Figure 3b, we observe that at cutoff $K = 1$, GPTRec with Top-K strategy and GPTRec with Next-K strategy have the same performance. Indeed, when $K = 1$, both strategies generate the item with the highest probability and therefore are equal. However, with higher cutoffs, Next-K performs worse, decreasing gradually decreasing to 75% of the Top-K quality at cutoff K=10; this is an expected effect as the model in our experiment was not specifically tuned for Next-K recommendation, and therefore its training task is not aligned with the generation task. Nevertheless, as we see from Figure 3a, even at cutoff K=10, GPTRec with Next-K strategy performs similarly to the SASRec model (0.108 NDCG@10 in both cases), which means that this model can be a strong starting point for further tuning using reinforcement learning or other techniques.

In summary, for RQ3, we find that at low cutoffs, Next-K and Top-K strategies used with GPTRec demonstrate similar performance. However, with increased K, the quality of the Next-K strategy gradually degrades. Nevertheless, GPTRec with the Next-K strategy retains competitive quality even with K=10, forming a strong basis for future tuning.



(a) Absolute **(b) Relative**

Figure 3: Comparison of GPTRec quality with Top-K and Next-K recommendation generation strategies. (a) shows the absolute value of the NDCG@K metric at different cutoff K. (b) shows the normalised quality of the Next-K strategy as the percentage of the Top-K strategy at the same cutoff K.

7 CONCLUSIONS AND FUTURE WORK

This paper presented GPTRec, a generative transformer model for the sequential recommendation problem that supports a novel GPU memory-efficient SVD tokenisation and a novel Next-K recommendation generation strategy suitable for complex interdependent objectives. This paper presented only early experiments with GPTRec and generative recommendations in general. In future work, we plan to expand model evaluation to more datasets (specifically large datasets, where multi-token-per-item promises the largest gains in memory). Additionally, we plan to explore more advanced generation techniques, such as Beam Search, to mitigate the diversity problems in the multi-token-per-item generation. We also plan to further research techniques to tune models for the Next-K generation and tune the model for complex objectives, such as diversity and coverage. In addition, we plan to evaluate the model using more hyperparameter settings. Another line of future work includes adaptations of proposed methods to the field of Generative IR. For example, adaptations of the proposed SVD tokenisation method can help resolve large document id vocabulary issues in Generative IR tasks. Similarly, GPTRec with the Next-K strategy may help resolve complex search results diversity and fairness problems.

To summarise this paper's findings, we showed that in the simplest case, when the model is used in one-token-per-item mode and uses a Top-K recommendation generation strategy, it showed strong performance similar to BERT4Rec (4% lower NDCG@10 on MovieLens-1M dataset). Compared to existing models, such as BERT4Rec, the model supports a multi-token-per-item recommendation mode, which reduces memory footprint with large detests while retaining competitive quality (e.g. we showed that using an SVD-based tokeniser, the model can match SASRec's performance while storing 40% fewer embeddings). We also presented the Next-K recommendation generation strategy, a promising direction for tuning models for complex training objectives that include diversity and coverage. We showed that GPTRec could, in principle, be used with Next-K recommendations. While we observed quality degradation compared with traditional Top-K recommendations, the recommendation quality remains competitive with SASRec. Overall we conclude that generative sequential recommendation is a promising research direction, which can be beneficial for reducing the model's memory consumption and be advantageous for complex training objectives. We also believe that the ideas presented in this paper can benefit research areas beyond sequential recommender systems, such as Generative IR.

REFERENCES

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models Are Few-Shot Learners. In *Proc. NeurIPS*, Vol. 33. 1877–1901.

[2] Wei Cai, Weike Pan, Jingwen Mao, Zhechao Yu, and Congfu Xu. 2022. Aspect Redistribution for Learning Better Item Embeddings in Sequential Recommendation. In *Proc. RecSys*. 49–58.

[3] Jaime Carbonell and Jade Goldstein. 1998. The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries. In *Proc. SIGIR*. ACM, Melbourne Australia, 335–336.

[4] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. arXiv:arXiv:1409.1259

[5] Zeyu Cui, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. 2022. M6-Rec: Generative Pretrained Language Models Are Open-Ended Recommender Systems. arXiv:2205.08084 [cs]

[6] 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.

[7] 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*. 396–405.

[8] 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*. 299–315.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[10] 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.

[11] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-Based Recommendations with Recurrent Neural Networks. In *Proc. ICLR*.

[12] Thomas Jaenich, Graham McDonald, and Iadh Ounis. 2023. ColBERT-FairPRF: Towards Fair Pseudo-Relevance Feedback in Dense Retrieval. In *Proc. ECIR*. 457–465.

[13] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *Proc. ICDM*. 197–206.

[14] 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 (2021), 114649–114658.

[15] Sanket Vaibhav Mehta, Jai Gupta, Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Jinfeng Rao, Marc Najork, Emma Strubell, and Donald Metzler. 2022. DSI++: Updating Transformer Memory with New Documents. arXiv:2212.09744 [cs]

[16] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training Language Models to Follow Instructions with Human Feedback. arXiv:2203.02155 [cs]

[17] Aleksandr Petrov and Craig Macdonald. 2022. Effective and Efficient Training for Sequential Recommendation Using Recency Sampling. In *Proc. RecSys*. 81–91.

[18] Aleksandr Petrov and Craig Macdonald. 2022. A Systematic Review and Replicability Study of BERT4Rec for Sequential Recommendation. In *Proc. RecSys*. 436–447.

[19] Ruihong Qiu, Zi Huang, Hongzhi Yin, and Zijian Wang. 2022. Contrastive Learning for Representation Degeneration Problem in Sequential Recommendation. In *Proc. WSDM*. 813–823.

[20] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. [n. d.]. Improving Language Understanding by Generative Pre-Training. ([n. d.]).

[21] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models Are Unsupervised Multitask Learners. *OpenAI blog* (2019).

[22] 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 (2020), 1–67.

[23] Rajput, Shashank, Mehta, Nikhil, Singh, Anima, Keshavan, Raghunandan, Vu, Trung, Heldt, Lukasz, Hong, Lichan, Tay, Yi, Tran, Vinh Q., Samost, Jonah, Kula, Maciej, Chi, Ed H., and Sathiamoorthy, Maheswaran. 2023. Recommender Systems with Generative Retrieval.

[24] Ozan Sener and Vladlen Koltun. 2018. Multi-task learning as multi-objective optimization. In *Proc. NeurIPS*.

[25] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. arXiv:1508.07909 [cs]

[26] 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*. 1441–1450.

[27] Weiwei Sun, Lingyong Yan, Zheng Chen, Shuaiqiang Wang, Haichao Zhu, Pengjie Ren, Zhumin Chen, Dawei Yin, Maarten de Rijke, and Zhaochun Ren. 2023. Learning to Tokenize for Generative Retrieval. arXiv:2304.04171 [cs]

[28] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *Proc. WSDM*. 565–573.

[29] Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. 2022. Transformer Memory as a Differentiable Search Index. (2022). arXiv:2202.06991 [cs.CL]

[30] 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*.

[31] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv:1910.03771 [cs]

[32] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv:1609.08144 [cs]

[33] Long Xia, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. 2015. Learning Maximal Marginal Relevance Model via Directly Optimizing Diversity Evaluation Measures. In *Proc. SIGIR*. 113–122.

[34] 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*. 582–590.

[35] 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 (2009).

[36] Yu-Jia Zhou, Jing Yao, Zhi-Cheng Dou, Ledell Wu, and Ji-Rong Wen. 2022. DynamicRetriever: A Pre-trained Model-based IR System Without an Explicit Index. *Machine Intelligence Research* 20, 2 (2022), 276–288.

[37] Ziwei Zhu, Jianling Wang, and James Caverlee. 2019. Improving Top-K Recommendation via Joint Collaborative Autoencoders. In *Proc. WWW*.