# Prompt Distillation for Efficient LLM-based Recommendation

Lei Li*
Hong Kong Baptist University
Hong Kong, China
csleili@comp.hkbu.edu.hk

Yongfeng Zhang
Rutgers University
New Brunswick, USA
yongfeng.zhang@rutgers.edu

Li Chen
Hong Kong Baptist University
Hong Kong, China
lichen@comp.hkbu.edu.hk

## ABSTRACT

Large language models (LLM) have manifested unparalleled modeling capability on various tasks, e.g., multi-step reasoning, but the input to these models is mostly limited to plain text, which could be very long and contain noisy information. Long text could take long time to process, and thus may not be efficient enough for recommender systems that require immediate response. In LLM-based recommendation models, user and item IDs are usually filled in a template (i.e., discrete prompt) to allow the models to understand a given task, but the models usually need extensive fine-tuning to bridge the user/item IDs and the template words and to unleash the power of LLM for recommendation. To address the problems, we propose to distill the discrete prompt for a specific task to a set of continuous prompt vectors so as to bridge IDs and words and to reduce the inference time. We also design a training strategy with an attempt to improve the efficiency of training these models. Experimental results on three real-world datasets demonstrate the effectiveness of our PrOmpt Distillation (POD) approach on both sequential recommendation and top-N recommendation tasks. Although the training efficiency can be significantly improved, the improvement of inference efficiency is limited. This finding may inspire researchers in the community to further improve the inference efficiency of LLM-based recommendation models.

## CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Natural language generation**.

## KEYWORDS

Recommender Systems; Large Language Models; Generative Recommendation; Sequential Recommendation; Top-N Recommendation; Explainable Recommendation; Prompt Distillation

**ACM Reference Format:**
Lei Li, Yongfeng Zhang, and Li Chen. 2023. Prompt Distillation for Efficient LLM-based Recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3615017

*Work was done during the visit at Rutgers University.


*CIKM '23, October 21–25, 2023, Birmingham, United Kingdom*

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3615017

## 1 INTRODUCTION

In recent years, recommender systems have been successfully deployed on various online platforms, such as e-commerce, video-streaming and social media. With recommendations, users can easily find what they are interested in without going through a vast amount of items. In the early days, recommendation models, such as collaborative filtering [34] and matrix factorization [17], are usually shallow and contain a limited number of parameters. Later on, they gradually grow deep [5], since deep neural networks generally have better representation ability than shallow models. Meanwhile, the abundant user-generated data on those platforms in turn give rise to the development of recommendation-related tasks, including review summarization [23], explanation generation [19], etc. More recently, recommendation models are entering a new stage where a single model can perform multiple tasks [6, 9].

These models are based on large language models (LLM), e.g., T5 [31], which refer to models that were trained on a huge amount of data and can adapt to a great number of downstream tasks. LLM have shown astonishing abilities that small models do not possess, e.g., doing arithmetic [3] that they were not trained for, so they have gained much attention and been adopted in many fields, such as natural language processing [3, 7], computer vision [32] and recommender systems [9]. As language can express various concepts, a recommendation task, similar to those in other fields, is also formulated as a textual description, i.e., discrete prompt [9, 26], before being fed into an LLM to enable sequence-to-sequence generation.

However, there are two problems with discrete prompt in recommendation scenarios. On one hand, user and item IDs, which are important identifiers in recommender systems, serve a different purpose from the words in the discrete prompt. Concretely, word embeddings capture the contextual relation between words, while ID embeddings encode users' preferences towards items as well as users' and items' similarity. As a result, LLM-based recommendation models usually need extensive fine-tuning to bridge the gap between IDs and template words and to unleash the power of LLM for recommendation. Besides, the key information to a recommendation model is the IDs, so the discrete prompt could be a little bit noisy when too many words are involved. On the other hand, processing long prompt takes long time, which may cause negative user experience, especially for recommendation services that require low-latency inference.

To cope with the problems, we propose a **PrOmpt Distillation (POD)**[1] approach, where we distill the knowledge in the discrete prompt into continuous prompt vectors. More specifically, we use both an array of continuous prompt vectors and a set of discrete prompt templates when training each recommendation task. As

[1]https://github.com/lileipisces/POD

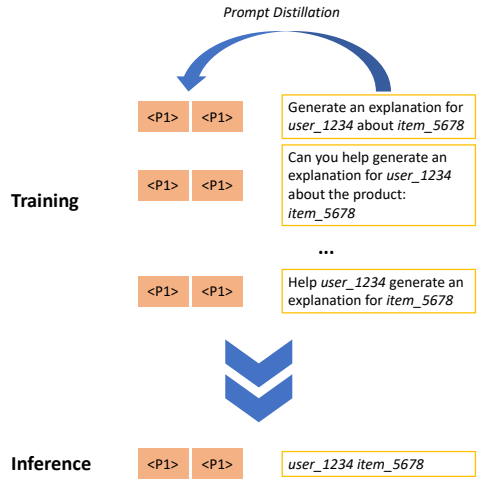



**Figure 1: An illustration of prompt distillation for the explanation generation task.**

the training process goes on, the randomly initialized continuous prompt vectors can learn the expressions in the discrete prompt, as shown in Fig. 1. Since continuous prompt vectors do not map to any concrete words, they can be more flexible and expressive than discrete prompt, and thus help LLM better learn different recommendation tasks. After prompt distillation, we only keep each task's continuous prompt so as to reduce the inference time.

As there are several recommendation tasks, the training efficiency is also a critical issue. Although mixing the samples of these tasks in one batch is viable [9], it is less efficient. Specifically, different tasks' input/output can be of varying length, so it may consume a lot of memory to pad them to the same length. As a result, the batch size would be small, the number of iterations would grow, and the training time would largely increase. To resolve the problem, we propose a strategy called **Task-alternated Training**, where we train the LLM with a batch of samples from the same task, followed by that of another task, and so on. Since each task's data are generally of the same length, this training strategy would not waste much memory on padding, thus improving the training efficiency.

Our key contributions are summarized below:

- As far as we know, our PrOmpt Distillation (POD) is the first approach that can distill the knowledge of discrete prompt to continuous prompt for LLM-based recommendation models. It is model-agnostic, and can be applied to any other LLM.
- We propose a Task-alternated Training strategy to improve the efficiency of training multiple recommendation tasks on LLM. This strategy has the potential to be extended to similar scenarios in other fields.
- Extensive experiments show that our approach POD can outperform state-of-the-art baselines by a large margin on two typical recommendation tasks, including sequential recommendation and top-N recommendation.
- As evidenced by our experiments, LLM's training efficiency can be easily improved, but improving its inference efficiency is not that easy. Hence, the latter might become a new research direction in recommender systems.

In the following, we first review research related to our work in Sec. 2, and then detail our methodology in Sec. 3. The experimental settings are described in Sec. 4, the analysis of results is provided in Sec. 5, and the conclusion with outlooks is given in Sec. 6.

## 2 RELATED WORK

We go through relevant research from large language models (LLM), prompt learning, LLM-based recommendation to prompt transfer.

Recently, LLM have drawn a lot of attention from both academia and industry, owing to its effectiveness on a wide spectrum of tasks. These models are usually trained on massive data, and can be classified into two main categories according to their training objectives, including masked language modeling and auto-regressive language modeling. For the former category, a certain number of tokens in a textual sequence is randomly masked, and the models need to predict them based on the non-masked tokens. A typical example is BERT [7], which is more suitable for natural language understanding tasks, e.g., question answering. For the latter, the models (e.g., T5 [31], LLaMA [37], GPT-3 [3]) are instructed to predict a token based on its proceeding tokens in the sequence.

A common way to adapt LLM to downstream tasks is to fine-tune them on task-specific datasets. However, with the prohibitively increased model scale, fine-tuning could also be computationally expensive. Therefore, there emerges a new paradigm named prompt learning [26]. Instead of adapting the models to the tasks, the tasks are adapted to the LLM in this paradigm. For instance, a sample for sentiment classification can be filled in a pre-defined template to form an input sequence, based on which an LLM can generate a few tokens that can be further mapped to the prediction score. This is termed discrete prompt learning, as the prompt template is comprised of discrete tokens. There is also another typical prompt learning approach called continuous/soft prompt learning, where the prompt is a set of vectors that do not map to any words. With the two types of prompt, LLM can easily perform different tasks, such as domain adaptation [2] and table-to-text generation [24].

Personalizing LLM for recommendation is important, as it can help to better understand a user's intent and address their personalized needs. However, early attempts that adopt LLM for recommendation tasks mostly focus on one single task, such as explainable recommendation [22] and sequential recommendation [13]. More recently, there has been growing interest in integrating several recommendation tasks into one LLM. For example, a collection of personalized prompts are designed in P5 [9] and OpenP5 [42] to handle several recommendation tasks simultaneously or to handle multiple modalities of data as in VIP5 [10]. In these models, all the tasks are formulated as a sequence-to-sequence generation problem. Another example is M6-Rec [6] where users and items are represented by their metadata (e.g., location and product name), but different tasks in this model employ task-specific loss functions. Our approach can also perform multiple recommendation tasks, but we focus on distilling the knowledge in discrete prompt to continuous prompt, which is not explored by previous work.

Besides prompt learning, another line of relevant research is prompt transfer. In SPoT [39], a prompt is first learned from source tasks and then utilized to initialize a target task's prompt. ATTEMPT [1] further employs an attention network to interpolate the

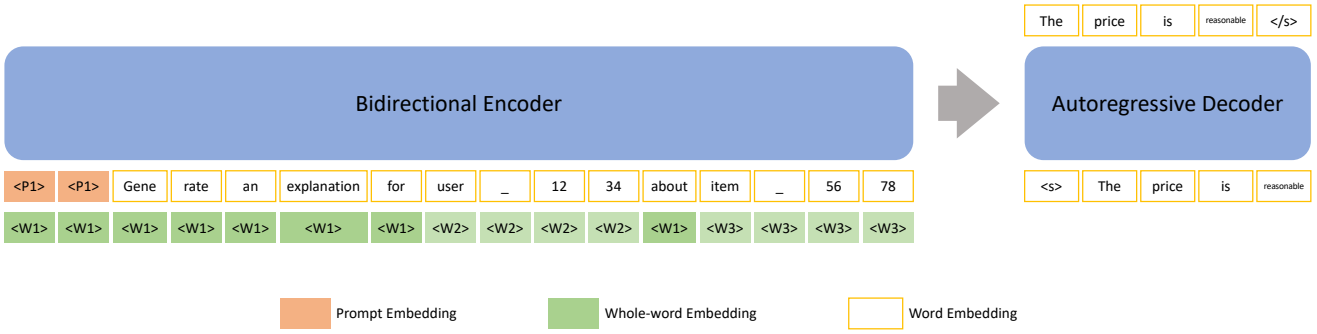



**Figure 2: Overview of our proposed prompt distillation approach for recommendation tasks. The backbone model follows an encoder-decoder architecture. The continuous prompt vectors are task-specific, and each task utilizes a set of discrete prompt templates. The whole-word embeddings are used to connect each ID's tokens.**

prompts of source tasks for better learning a target task's prompt. PANDA [46] first measures prompt transferability, and then transfers the knowledge from a source prompt to a target prompt. This is achieved by knowledge distillation [4, 12], where the predictions of a large teacher model are used to learn a small but effective student model. Similarly, MPT [41] distills the shared knowledge between multiple source tasks to a single prompt so that it can be leveraged by a target task. Although related, our work differs from them in that we perform intra-task prompt distillation, while they do cross-task prompt transfer. UP5 [14] distills an LLM's bias information into the encoder and decoder prompts so as to reduce the bias and improve the fairness of LLM-based recommendation models. This work is quite relevant to ours regarding knowledge distillation, but its research focus is on bias and fairness issue, while we care about effectiveness and efficiency.

## 3 METHODOLOGY

As LLM can handle a variety of tasks, we first give an overview of three typical recommendation tasks that we will test with our approach, including sequential recommendation, top-N recommendation and explainable recommendation. Note that, other recommendation tasks can also be easily incorporated. Then, we briefly introduce discrete prompt in recommendation scenarios. After that, we go through the details of our PrOmpt Distillation (POD) approach, followed by our proposed Task-alternated Training strategy. At last, we show how to generate recommendations and explanations with beam search algorithm.

### 3.1 Tasks Formulation

Before introducing the tasks, we provide some basic notations that will be used throughout the paper. Specifically, we use $\mathcal{U}$ to denote the user set in a dataset, and $\mathcal{I}$ the item set. A particular user in $\mathcal{U}$ is denoted as $u$, and a specific item in $\mathcal{I}$ is represented by $i$. For the task of sequential recommendation, each user $u$ is associated with his/her chronologically ordered interaction history $I^u = \{i_1^u, i_2^u, ..., i_{|I^u|}^u\}$. Based on the user and the item sequence, the LLM needs to predict which item the user is going to interact with next, i.e., $i_{|I^u|+1}^u \in \mathcal{I}/\mathcal{I}^u$, where $\mathcal{I}^u$ is the item set that user $u$ have interacted with. The top-N recommendation task is to recommend the user $u$ an item list that consists of N items which the user never interacted with before, i.e., items from $\mathcal{I}/\mathcal{I}^u$. For training this task, we can randomly sample an item $i$ from $\mathcal{I}^u$ and pair it with random negative items drawn from $\mathcal{I}/\mathcal{I}^u$ to form a candidate item list. As to explanation task, given a pair of user $u$ and item $i$, the LLM needs to generate an explanation $E_{u,i}$ that justifies why the item is recommended to the user (e.g., "the price is reasonable"). Notice that, for all the tasks, there is no side information or additional content provided besides the above mentioned data.

### 3.2 Discrete Prompt for Recommendation

In recommender systems, user and item IDs are essential, because they are the key to distinguishing one user/item from the others. However, IDs can be different from words. To adapt IDs to LLM, most previous work [6, 13] adopt ID-associated text segments (such as user name and item title) as an alternative, such that they can be easily filled in a pre-defined template (e.g., "explain why the movie Guardians of the Galaxy Vol. 3 is recommended to Cheryl"). This type of textual input is formally termed *discrete prompt* [26] because it is comprised of a sequence of discrete tokens.

However, experimental results from recent studies [13, 15] suggest that LLM's recommendation performance could be largely impaired when ID information is unavailable, because ID makes it possible to exactly identify and distinguish different items and conveys important information such as the collaborative relationship between items [15]. To address the problem, our solution is to keep the IDs and represent them as textual strings where the original ID numbers are still retained. For example, the user with ID 1234 can be represented as "user_1234", where the prefix "user_" is used to distinguish user IDs from item IDs (i.e., the latter starts with "item_"). As the composing units in this string are already in the LLM's tokenizer, it can be easily tokenized into a token sequence (i.e., "user", "_", "12" and "34"). After the IDs are broken into several pieces, the model might not be able to recognize which tokens belong to a particular ID. To connect an ID's tokens, we employ an additional set of *whole-word embeddings* [9]. As Fig. 2 illustrated, each ID's token sequence shares one whole-word embedding vector, while all the other non-ID tokens share the same whole-word embedding vector. Such a way can make each ID a whole unit, and IDs distinguishable to words.

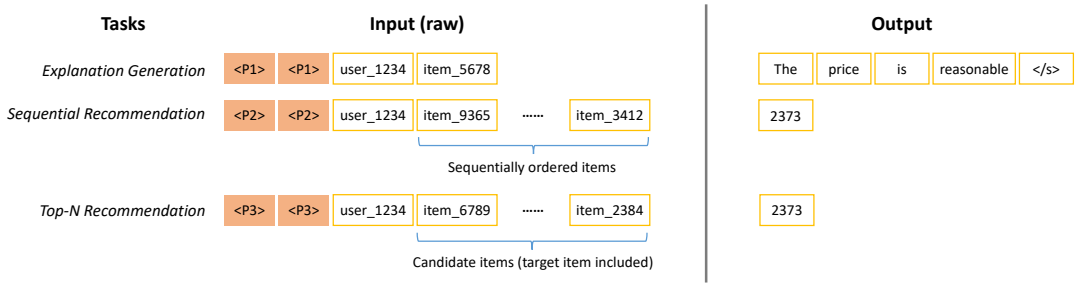



**Figure 3: Varying length of different tasks' input/output. The discrete prompt templates and word tokenization are both omitted for better illustration.**

## 3.3 Prompt Distillation

Although it looks promising, discrete prompt may fail to give an effective instruction to LLM, e.g., "summarize the content in the table" [24]. This could become a serious issue in recommender systems especially when the input data format of two different tasks is quite similar. For example, the input of both sequential recommendation and top-N recommendation is a user and a bunch of items (as discussed in Sec. 3.1). If the model misunderstands the discrete prompt, it may treat one task as another and fail to provide accurate recommendations. Moreover, when the prompt template is long, it may overshadow the key information (i.e., IDs) and also take long time for training and inference. To address the problems, we propose a PrOmpt Distillation (POD) approach, whose formal definition is given below.

Definition 1 (Prompt Distillation). *We call an approach prompt distillation if it can shorten a long prompt without sacrificing an LLM's performance on the testing tasks. The distilled short prompt can either be free text or vectors.*

In this work, we distill discrete prompt templates into multiple continuous prompt vectors. Concretely, we append a set of vectors at the beginning of an input sample that already filled in a discrete prompt template, and allow the vectors to be shared by the samples of the same recommendation task. Fig. 2 shows an example of the explanation generation task. For other tasks, another set of vectors will be utilized. As the training progresses, the continuous prompt vectors can learn the expressions in the discrete prompt through the loss function. Since they do not map to any real words, they can be more expressive and flexible than discrete prompt. After the distillation stage is completed, we only keep the continuous prompt in order to improve the inference efficiency.

Technically, we adopt the encoder-decoder architecture for a fair comparison with a recent work with discrete prompt [9], but our approach can be easily extended to decoder-only models. Fig. 3 shows the example input and output of the three recommendation tasks (the prompt template and word tokenization are both omitted for better illustration). We denote an input-output sequence pair for either of the tasks as $X = [x_1, ..., x_{|X|}]$ and $Y = [y_1, ..., y_{|Y|}]$. After passing the input sequence $X$ through the embedding layer, we can obtain its tokens' representations $[\mathbf{x}_1, ..., \mathbf{x}_{|X|}]$, which will be appended at the end of the $K$ continuous prompt vectors $[\mathbf{p}_1, ..., \mathbf{p}_K]$. This concatenated representation $[\mathbf{p}_1, ..., \mathbf{p}_K, \mathbf{x}_1, ..., \mathbf{x}_{|X|}]$ will then be added to the whole-word representation $[\mathbf{w}_1, ..., \mathbf{w}_{K+|X|}]$, which gives us $\mathbf{S} = [\mathbf{s}_1, ..., \mathbf{s}_{|S|}]$. After passing $\mathbf{S}$ through the LLM's encoder, it will produce a sequence of hidden vectors $\mathbf{H} = [\mathbf{h}_1, ..., \mathbf{h}_{|S|}]$, based on which the LLM's decoder can perform auto-regressive generation. Specifically, at each time step $t$, the decoder outputs a probability distribution $p(y|Y_{<t}, \mathbf{H})$ over the vocabulary $\mathcal{V}$, where $Y_{<t}$ denotes the tokens generated before time step $t$. Since all the tasks in this work are formulated as natural language generation problem, we adopt the commonly used Negative Log-Likelihood (NLL) loss function to optimize the model parameters $\Theta$:

$$\mathcal{L}_\Theta = \frac{1}{|\mathcal{D}|} \sum_{(X,Y)\in\mathcal{D}} \frac{1}{|Y|} \sum_{t=1}^{|Y|} -\log p(y_t|Y_{<t}, X) \tag{1}$$

where $\mathcal{D}$ is the training set that consists of all the input-output pairs $(X, Y)$. $|\mathcal{D}|$ and $|Y|$ denote the amount of training samples and the number of tokens in the output sequence, respectively. $p(y_t|Y_{<t}, X)$ represents the probability of generating token $y_t$, given the input sequence and the already generated tokens.

## 3.4 Task-alternated Training

Unlike the pre-training stage in natural language processing where all the training samples in each batch are of the same length, the input and output in recommendation tasks could be of varying length. For example, the explanation task's input only consists of two IDs (i.e., user and item), while that of sequential recommendation could be hundred-scale owing to the user's historical item sequence (see Fig. 3). Meanwhile, the former's output is an explanation sentence with tens of words, but that of the latter is merely an item ID. The varying length of different tasks' input and output makes it less efficient to train an LLM with mixed samples from different tasks [9], because padding them to the same length would consume a lot of memory, make the batch size quite small, and thus lead to more iterations and longer training time. Our solution is to alternately update the model parameters with a batch of samples from one task, followed by that from another task, and so on. In general, each task has the same data format, so this strategy can save a lot of memory for improving the training efficiency. We dub it *Task-alternated Training* and provide the training procedure as well as the implementation details in Algorithm 1.

## 3.5 Generation with Beam Search

Since all the tasks are formulated as a sequence-to-sequence generation problem, we can instruct the well trained model to generate

**Algorithm 1** Task-alternated Training

**Input:** Explanation set $\mathcal{E} = \{(u, i, E_{u,i})\}$, user set $\mathcal{U}$, prompt template sets for explanation $\mathcal{P}_e$, sequential recommendation $\mathcal{P}_s$ and top-N recommendation $\mathcal{P}_t$, number of negative items $n$
**Output:** Model parameters $\Theta$

1: **repeat**
2: Uniformly draw a batch $\mathcal{B}$ from $\mathcal{E}$ // explanation
3: **for** $(u, i, E_{u,i})$ in $\mathcal{B}$ **do**
4: Draw a template $prompt(\cdot)$ from $\mathcal{P}_e$
5: $x \leftarrow prompt(u, i), y \leftarrow E_{u,i}$
6: **end for**
7: $X \leftarrow [x_1, ..., x_{|\mathcal{B}|}], Y \leftarrow [y_1, ..., y_{|\mathcal{B}|}]$
8: Update $\Theta$ with $\mathcal{L}_\Theta$ in Eq. (1) by feeding $(X, Y)$
9: Uniformly draw a batch $\mathcal{B}$ from $\mathcal{U}$ // sequential
10: **for** $u$ in $\mathcal{B}$ **do**
11: Draw a template $prompt(\cdot)$ from $\mathcal{P}_s$, a segment $\widetilde{I^u}$ from $I^u_{|I^u|-2}$ // last two items for validation and testing
12: $x \leftarrow prompt(u, \widetilde{I^u_{|I^u|-1}}), y \leftarrow \widetilde{i^u_{|I^u|}}$
13: **end for**
14: Execute line 7 and 8
15: Uniformly draw a batch $\mathcal{B}$ from $\mathcal{U}$ // top-N
16: **for** $u$ in $\mathcal{B}$ **do**
17: Draw a template $prompt(\cdot)$ from $\mathcal{P}_t$, an item $i$ from $I^u_{|I^u|-2}$, $n$ negative items $\mathcal{I}^n$ from $\mathcal{I}/\mathcal{I}^u$
18: Add $i$ to $\mathcal{I}^n$ and shuffle
19: $x \leftarrow prompt(u, \mathcal{I}^n), y \leftarrow i$
20: **end for**
21: Execute line 7 and 8
22: **until** Convergence

output sequences as recommendations (in the form of ID tokens) or explanations. Among the many decoding algorithms, we choose beam search because of its effectiveness in finding good sequences. Suppose we set the number of beams to $b$, at each time step there would be $b$ candidate sequences. In the next step, any word in the vocabulary $\mathcal{V}$ can be appended to the end of the candidate sequences, which makes $b \times \mathcal{V}$ combinations, from which we can select $b$ sequences that have the maximum log-likelihood. The LLM can keep doing this until the candidate sequences reach a pre-defined maximum length. For sequential recommendation and top-N recommendation, the $b$ candidate sequences form the recommendation list. As to explanation, we pick the one with the largest log-likelihood from the candidates.

# 4 EXPERIMENTAL SETUP

In this section, we give the details of our experimental setup, including data selection, baseline methods, evaluation protocols and implementation.

## 4.1 Datasets

We conduct experiments on three widely used datasets, which are all collected from an e-commerce platform Amazon[2]. There are in total 29 different product categories in Amazon datasets[3], and we adopt Sports & Outdoors, Beauty, and Toys & Games for experimentation. Each record in the three datasets is comprised of a user ID, an item ID, a rating, a textual review and a timestamp. The statistics of the datasets are given in Table 1.

**Table 1: Statistics of the datasets.**

| Dataset | **Sports** | **Beauty** | **Toys** |
|---|---|---|---|
| #Users | 35,598 | 22,363 | 19,412 |
| #Items | 18,357 | 12,101 | 11,924 |
| #Reviews | 296,337 | 198,502 | 167,597 |
| #Sparsity (%) | 0.0453 | 0.0734 | 0.0724 |

To obtain natural language explanations for the explanation generation task, we follow the procedure in [19, 20] to extract from user reviews some sentences that contain item features (e.g., "price") with the Sentires[4] [44, 45] toolkit. For this task, we divide each dataset into training, validation and testing sets with the ratio of 8:1:1, and also hold at least one record for each user and item in the training set. For the sequential recommendation task, we first sort a user's interacted items chronologically in accordance with their timestamps to obtain an item sequence. Then, the last item in the interaction sequence is used for evaluation, the penultimate item for validation, and the rest for training. To prevent from data leakage, we also adopt this data partition strategy for the top-N recommendation task.

## 4.2 Baselines

To verify our approach's effectiveness on different recommendation tasks, we compare it with the following three groups of representative baselines.

*4.2.1 Sequential Recommendation.* Given a user and his/her interacted item sequence, recommendation models should predict the next item that the user is likely to interact with.

- **CASER**: ConvolutionAl Sequence Embedding Recommendation model [36]. This method considers a user's recently interacted items as a virtual image, so that convoluational neural network can be utilized to capture the sequential pattern.
- **HGN**: Hierarchical Gating Network [29]. It consists of two gating modules (i.e., feature-level and instance-level) for modeling users' long-term and short-term interests, respectively.
- **GRU4Rec** [11]. It is a session-based recommendation approach where GRU is employed to process item sessions. In this work, we treat a user's whole item sequence as a session.
- **BERT4Rec** [35]. This is a bidirectional Transformer model trained with the BERT-style cloze task. Specifically, some items in a user's historical item sequence are randomly masked, and then the model needs to predict them with the other context items in the sequence.

[2]https://www.amazon.com/

[3]https://jmcauley.ucsd.edu/data/amazon/

[4]https://github.com/evison/Sentires

- **FDSA**: Feature-level Deeper Self-Attention network [43]. Item features are incorporated into the model as feature sequences, which are then merged with item sequences for making the recommendation.
- **SASRec**: Self-Attention based Sequential Recommendation model [16]. With self-attention, the model attempts to combine the merits of Markov Chains in dealing with short-term semantics and Recurrent Neural Networks in handling long-term semantics.
- **$S^3$-Rec**: Self-Supervised learning for Sequential Recommendation [47]. With the help of mutual information maximization principle, four self-supervised learning objectives are devised to learn the correlation between items in the sequential data.
- **P5**: Pretrain, Personalized Prompt, and Predict Paradigm [9]. It is an LLM-based recommendation model with discrete prompt. Sequential recommendation is formulated as a sequence-to-sequence generation problem in this model. We feed the model a template like "Here is the purchase history list of user_{user_id}: {purchase_history}. Try to recommend next item to the user" to enable next-item prediction.

*4.2.2 Top-N Recommendation.* Based on a user's item history, the recommendation models need to suggest a list of N items that the user never interacted with but might match his/her interests.

- **MF**: Matrix Factorization [17]. It is a classic collaborative filtering method, whose prediction is made by the inner product between user and item latent factors. To better learn users' preferences, Bayesian Personalized Ranking (BPR) [33] is adopted as the loss function.
- **MLP**: Multi-Layer Perceptron [5]. User and item embeddings are passed through a stack of non-linear layers for making a prediction. BPR is also equipped in this model.
- **P5** [9]. Again, with a discrete prompt like "We want to make recommendation for user_{user_id}. Select the best item from these candidates: {candidate_items}", this model can perform top-N recommendation in the way of natural language generation.

*4.2.3 Explanation Generation.* A pair of user and item is given to each model for them to produce a textual sentence that can explain why the recommender system recommends this item to the user.

- **Att2Seq**: Attribute-to-Sequence [8]. It is a review generation method where the encoder (an MLP) converts the user and item IDs into a hidden state, from which the decoder (an LSTM) then decodes a word sequence. In this case, we regard the ground-truth explanation as the review.
- **NRT**: Neural Ratings and Tips generation [23]. This is a tip generation framework. Similarly, it employs MLP to encode the user and item IDs, but the decoder is a GRU. In our experiment, we treat the explanation as the tip.
- **PETER**: PErsonalized Transformer for Explainable Recommendation [21]. It is a Transformer-based model whose attention masking matrix is slightly revised so as to enable the interaction between user and item IDs.

For this task, we omit the comparison with P5, because it utilizes additional data, i.e., item titles and ratings, which would be unfair to the other models.

## 4.3 Evaluation Metrics

To evaluate recommendation accuracy, we employ the commonly used Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG) for both sequential recommendation and top-N recommendation. While HR measures the ratio of ground-truth items really appearing in the recommendation list, NDCG gives credit to those matched items that are ranked higher in the list. We report HR@1, HR@5 and HR@10, and NDCG@5 and NDCG@10. As to the evaluation of generated explanations, we adopt two well-known metrics: BLEU [30] and ROUGE [25]. The former is precision-oriented, while the latter is recall-oriented, but both metrics measure the overlapping degree of $n$-grams within the generated sentences and the ground-truth. We report BLEU-4, and F1 of ROUGE-1, ROUGE-2 and ROUGE-L. For all these metrics, a larger value means a better performance.

## 4.4 Implementation Details

For a fair comparison, both the baseline P5 [9] and our approach POD adopt T5-small [31] as the backbone. In this LLM, the encoder and decoder both have 6 layers, each of which is an 8-headed attention layer. T5 utilizes SentencePiece [18] to tokenize a sentence into a sequence of sub-words (see Sec. 3.2). There are in total 32,100 tokens in T5's vocabulary $\mathcal{V}$, and their embedding dimensionality is 512. As described in Algorithm 1, we randomly sample a segment from a user's item sequence for training the sequential recommendation task; the number of negative items $n$ for top-N recommendation is set to 99 for both training and evaluation. We train our POD on the training set with the AdamW optimizer [28], and report the results on the testing set. After hyper-parameters tuning on the validation set, we set the number of continuous prompt vectors $K$ for each task to 3, the batch size for training all the three tasks to 64, and the learning rate to 0.001 for Sports dataset and 0.0005 for both Beauty and Toys datasets. We borrow the discrete prompt templates for different tasks from [9], since constructing them is not our key focus. At the training stage, we save a checkpoint if the model's total validation loss on the three tasks is the lowest as of the current epoch. If this does not happen for 5 times, we terminate the training process and load the best checkpoint for evaluation. At the inference stage, we set the number of beams $b$ to 20 for sequential recommendation and top-N recommendation. As to explanation generation, we apply group beam search and set $b$ to 21 and the number of beam groups to 3. The inference batch size is set to 32 for all tasks.

# 5 RESULTS AND ANALYSIS

This section presents the experimental results on the three recommendation tasks, the efficiency comparison for both training and inference stages, and hyper-parameter analysis.

## 5.1 Sequential Recommendation

Table 2 shows the recommendation accuracy comparison between different sequential recommendation methods. From the table, we

**Table 2: Performance comparison on sequential recommendation.**

| Methods | Sports | | | | Beauty | | | | Toys | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | HR@5 | NDCG@5 | HR@10 | NDCG@10 | HR@5 | NDCG@5 | HR@10 | NDCG@10 | HR@5 | NDCG@5 | HR@10 | NDCG@10 |
| Caser | 0.0116 | 0.0072 | 0.0194 | 0.0097 | 0.0205 | 0.0131 | 0.0347 | 0.0176 | 0.0166 | 0.0107 | 0.0270 | 0.0141 |
| HGN | 0.0189 | 0.0120 | 0.0313 | 0.0159 | 0.0325 | 0.0206 | 0.0512 | 0.0266 | 0.0321 | 0.0221 | 0.0497 | 0.0277 |
| GRU4Rec | 0.0129 | 0.0086 | 0.0204 | 0.0110 | 0.0164 | 0.0099 | 0.0283 | 0.0137 | 0.0097 | 0.0059 | 0.0176 | 0.0084 |
| BERT4Rec | 0.0115 | 0.0075 | 0.0191 | 0.0099 | 0.0203 | 0.0124 | 0.0347 | 0.0170 | 0.0116 | 0.0071 | 0.0203 | 0.0099 |
| FDSA | 0.0182 | 0.0122 | 0.0288 | 0.0156 | 0.0267 | 0.0163 | 0.0407 | 0.0208 | 0.0228 | 0.0140 | 0.0381 | 0.0189 |
| SASRec | 0.0233 | 0.0154 | 0.0350 | 0.0192 | 0.0387 | 0.0249 | 0.0605 | 0.0318 | 0.0463 | 0.0306 | 0.0675 | 0.0374 |
| $S^3$-Rec | 0.0251 | 0.0161 | 0.0385 | 0.0204 | 0.0387 | 0.0244 | 0.0647 | 0.0327 | 0.0443 | 0.0294 | 0.0700 | 0.0376 |
| P5 | 0.0272 | 0.0169 | 0.0361 | 0.0198 | 0.0503 | 0.0370 | 0.0659 | 0.0421 | 0.0648 | 0.0567 | 0.0709 | 0.0587 |
| POD | **0.0496** | **0.0396** | **0.0576** | **0.0419** | **0.0537** | **0.0395** | **0.0688** | **0.0443** | **0.0691** | **0.0599** | **0.0742** | **0.0610** |
| Improvement (%) | 82.35 | 134.32 | 49.61 | 105.39 | 6.76 | 6.76 | 4.40 | 5.23 | 6.64 | 5.64 | 4.65 | 3.92 |

**Table 3: Performance comparison on top-N recommendation.**

| Methods | Sports | | | | | Beauty | | | | | Toys | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | HR@1 | HR@5 | NDCG@5 | HR@10 | NDCG@10 | HR@1 | HR@5 | NDCG@5 | HR@10 | NDCG@10 | HR@1 | HR@5 | NDCG@5 | HR@10 | NDCG@10 |
| MF | 0.0314 | 0.1404 | 0.0848 | 0.2563 | 0.1220 | 0.0311 | 0.1426 | 0.0857 | 0.2573 | 0.1224 | 0.0233 | 0.1066 | 0.0641 | 0.2003 | 0.0940 |
| MLP | 0.0351 | 0.1520 | 0.0927 | 0.2671 | 0.1296 | 0.0317 | 0.1392 | 0.0848 | 0.2542 | 0.1215 | 0.0252 | 0.1142 | 0.0688 | 0.2077 | 0.0988 |
| P5 | 0.0567 | 0.1514 | 0.1049 | 0.2196 | 0.1269 | 0.0571 | 0.1566 | 0.1078 | 0.2317 | 0.1318 | 0.0451 | 0.1322 | 0.0889 | 0.2023 | 0.1114 |
| POD | **0.0895** | **0.2086** | **0.1506** | **0.2873** | **0.1756** | **0.0829** | **0.1926** | **0.1391** | **0.2670** | **0.1629** | **0.0567** | **0.1433** | **0.1009** | **0.2082** | **0.1215** |
| Improvement (%) | 57.85 | 37.24 | 43.57 | 7.56 | 35.49 | 45.18 | 22.99 | 29.04 | 3.77 | 23.60 | 25.72 | 8.40 | 13.50 | 0.24 | 9.07 |

**Table 4: Performance comparison on explanation generation (%).**

| Methods | Sports | | | | Beauty | | | | Toys | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BLUE-4 | ROUGE-1 | ROUGE-2 | ROUGE-L | BLUE-4 | ROUGE-1 | ROUGE-2 | ROUGE-L | BLUE-4 | ROUGE-1 | ROUGE-2 | ROUGE-L |
| Att2Seq | 0.5305 | 12.2800 | 1.2107 | 9.1312 | 0.7889 | 12.6590 | 1.6820 | 9.7481 | 1.6238 | 13.2245 | 2.9942 | 10.7398 |
| NRT | 0.4793 | 11.0723 | 1.1304 | 7.6674 | 0.8295 | 12.7815 | 1.8543 | 9.9477 | 1.9084 | 13.5231 | 3.6708 | 11.1867 |
| PETER | 0.7112 | 12.8944 | 1.3283 | 9.8635 | **1.1541** | 14.8497 | **2.1413** | **11.4143** | 1.9861 | **14.2716** | 3.6718 | **11.7010** |
| POD | **1.0013** | **14.0168** | **2.0436** | **11.1236** | 1.0630 | **15.2517** | 1.5737 | 11.3283 | **2.3053** | 12.2889 | **3.8512** | 10.3923 |
| Improvement (%) | 40.79 | 8.70 | 53.85 | 12.78 | -7.89 | 2.71 | -26.51 | -0.75 | 16.07 | -13.89 | 4.89 | -11.18 |

**Table 5: Efficiency comparison between two training strategies on Sports dataset. In this table, "h" and "m" denote "hours" and "minutes", respectively.**

| Training Strategies | Time | Epochs | Time/Epoch |
|---|---|---|---|
| Sample-mixed | 15h59m | 13 | 1h14m |
| Task-alternated | 6h55m | 22 | 19m |
| Improvement (%) | 56.73 | - | 74.32 |

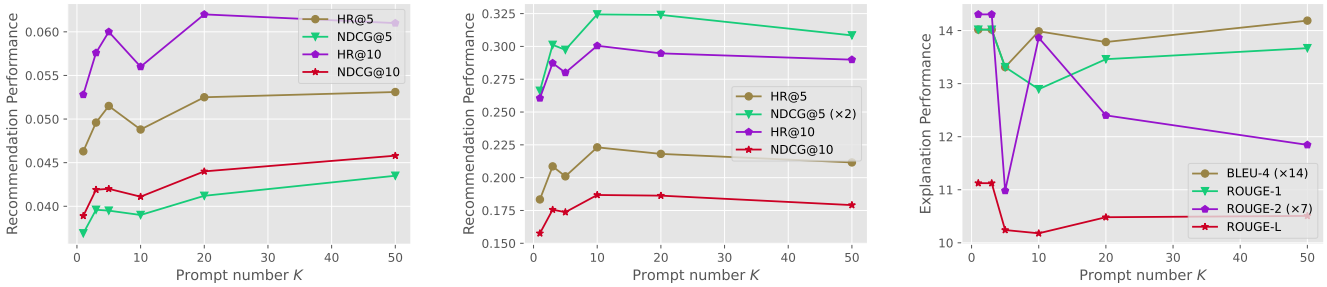
can see that our method POD beats all the baselines by a large margin, especially P5 which shares the similar architecture with ours. The only difference between our approach and P5 is that it relies heavily on the manually constructed prompt templates for making predictions. However, the word tokens and ID tokens need extensive fine-tuning to be learned into the same embedding space. Instead, our approach distills the discrete prompt into continuous prompt, which is no longer restricted to fixed words and thus could be more compatible with IDs, leading to improved recommendation performance. Among the other baselines, we also notice that $S^3$-Rec and SASRec are both very competitive, which could be attributed to the advantage of self-attention mechanism in modeling sequential data. However, they are not as effective as our POD and the baseline P5, since they are not LLM and cannot leverage the sequential pattern in the large textual corpora for pre-training.

## 5.2 Top-N Recommendation

In Table 3, we compare our method with representative baselines for top-N recommendation. MF and MLP are both classic recommendation models, and their performance is not bad. In general, MF is slightly worse than MLP, probably because the simple inner product may not be able to fully exploit the complex pattern in user-item interactions. This may explain why recent works adopt neural networks that can perform non-linear transformations on recommendation data. However, MLP is still not comparable to LLM-based recommendation models (i.e., P5 and POD). In particular, the latter's top-1 recommendation accuracy is way much better than the other baselines, because they can directly generate recommendations. This demonstrates the advantage of LLM-based recommendation models in accurately modeling users' preferences towards items. Most importantly, our method POD consistently performs better than all the baselines, as the continuous prompt is more compatible with LLM since neural networks are essentially continuous [27].

**Table 6: Inference efficiency comparison on Sports dataset. In the "Time" column, "m" and "s" stand for minutes and seconds, respectively.**

| Methods | Sequential Recommendation | | | | | Top-N Recommendation | | | | | Explanation Generation (%) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | HR@5 | NDCG@5 | HR@10 | NDCG@10 | Time | HR@5 | NDCG@5 | HR@10 | NDCG@10 | Time | BLEU-4 | ROUGE-2 | ROUGE-L | Time |
| Continuous+Discrete | 0.0509 | 0.0411 | 0.0583 | 0.0432 | 24m6s | 0.2079 | 0.1508 | 0.2882 | 0.1763 | 48m31s | 1.0012 | 2.0436 | 11.1202 | 9m30s |
| Continuous only (POD) | 0.0496 | 0.0396 | 0.0576 | 0.0419 | 22m17s | 0.2086 | 0.1506 | 0.2873 | 0.1756 | 47m13s | 1.0013 | 2.0436 | 11.1236 | 8m59s |
| Improvement (%) | -2.55 | -3.65 | -1.20 | -3.01 | 7.54 | 0.34 | -0.13 | -0.31 | -0.40 | 2.68 | 0.01 | 0.00 | 0.03 | 5.44 |




(a) Sequential recommendation (b) Top-N recommendation (c) Explanation generation

**Figure 4: The effect of continuous prompt number $K$ on different tasks on the Sports dataset. The values of some metrics are linearly scaled for better visualization.**

We also notice that when N grows large, the performance improvement of our approach against baselines becomes small. Owing to the leave-one-out evaluation protocol as explained in Sec. 4.1, there is only one ground-truth item in the testing set. Hence, the more items for testing (i.e., larger N), the higher the hit ratio of baselines. However, this would not help with our approach's accuracy, since it already can correctly predict the item at the first position, i.e., top-1 recommendation. This attribute is of great practical value, especially when the number of recommendations is limited, e.g., in conversational recommendation scenario where the system can only display a few (usually just one) recommendation results.

## 5.3 Explanation Generation

The performance comparison between different explanation generation methods is presented in Table 4. The results on the three datasets are not very consistent. Our approach obtains the best performance on Sports dataset, and comparable results on Beauty and Toys datasets. Admittedly, it does not always outperform baselines, probably because the evaluation metrics BLEU and ROUGE only compute the overlapping segments from generated explanations and the ground-truth [40]. This actually puts LLM at disadvantage, as they are capable of generating diverse and expressive content, given that they have learned the world knowledge during the pre-training stage. As for the baselines, we can see that the results of Att2Seq and NRT are generally not as competitive as the other two Transformer-based methods, which could be explained by the fact that recurrent neural networks (i.e., LSTM in Att2Seq and GRU in NRT) suffer from long-range dependency problem. However, this is not a problem to Transformer [38], whose self-attention mechanism allows future tokens to fully attend to proceeding tokens in a sequence.

## 5.4 Training Efficiency

The efficiency of training an LLM with multiple tasks is a critical issue. In this work, we propose an efficient training strategy named Task-alternated Training. To verify its effectiveness in helping improve the training efficiency, we compare it with the training method used in a previous work [9]. We name this comparative method Sample-mixed Training, as the samples from different tasks are mixed in one batch. The experiments are conducted on Sports dataset with an NVIDIA Tesla V100S GPU. When testing the two training strategies, we tune the batch size to its maximum so that the model can occupy as much memory as possible for a fair comparison. We keep the other settings the same for both methods.

The experimental results are given in Table 5. Although sample-mixed training takes less epochs to train, its overall training time is twice as long as our task-alternated training. With regard to the average training time per epoch, our approach is approximately five times more efficient than the compared approach. As we have explained before, mixing the samples of different tasks in the same batch would waste a lot of memory on padding because they are very likely to be of different length. On one GPU with fixed amount of memory, the batch size would be small, and the number of iterations as well as the training time would be increased. Our training strategy mitigates this problem by alternately feeding the LLM a batch of samples from the same task.

## 5.5 Inference Efficiency

As discussed above, we drop the discrete prompt and only use the continuous prompt during the inference stage so as to reduce computation cost. As an ablation study, we investigate whether this strategy can indeed help improve inference efficiency. There is no doubt that the comparative method should use both the continuous

prompt and the discrete prompt, which we denote as "Continuous+Discrete". Again, we test the two methods on Sports dataset with an NVIDIA Tesla V100S GPU, and keep their settings the same. The results are shown in Table 6. As we can see from the table, the inference efficiency can be improved when the discrete prompt is removed, while the recommendation and explanation performances do not change much. We also notice that the efficiency improvement on the top-N recommendation task is less significant than the other two tasks. The reason is as follows. There are in total 101 IDs (i.e., one user ID, one target item ID and 99 negative item IDs) for this task, which result in approximately 400 tokens. Compared with them, the prompt templates are much shorter, which would make the improvement room small. Nevertheless, this experiment demonstrates that our POD approach can indeed reduce the inference time.

### 5.6 Number of Continuous Prompt Vectors

At last, we would like to investigate how varying number of continuous prompt vectors $K$ would affect the performance of different recommendation tasks. We conduct the experiment on Sports dataset, and search $K$ from [1, 3, 5, 10, 20, 50]. As we can see from Fig. 4, the trend of recommendation performance and that of explanation performance are different. With the increase of $K$, the performances of both sequential recommendation and top-N recommendation generally grow, but the explanation performance could already reach its optimum when $K$ is small (i.e., 1 or 3). Considering that a large amount of prompt vectors would affect both training and inference efficiency, we prefer a small $K$. Therefore, we set $K$ to 3, where the explanation performance is the best and the recommendation performance can already beat all the baselines.

## 6 CONCLUSION

In this work, we present a simple but effective PrOmpt Distillation (POD) approach that can distill the knowledge of discrete prompt templates into continuous prompt vectors for LLM-based recommendation models. In our experiments, we demonstrate that POD is both effective on typical recommendation tasks and efficient during the inference stage. We also propose a Task-alternated Training strategy that can largely improve the efficiency of training an LLM-based recommendation model. We believe that both approaches have the potential to be extended to other fields where the tasks have different data formats. As simply removing the discrete prompt could lead to improved inference efficiency, we plan to make the LLM light-weighted so as to save computation cost. Also, we are interested in cross-task prompt transfer [1, 39, 41, 46] that might be able to generalize LLM to new recommendation tasks.

## ACKNOWLEDGMENTS

This work was supported by Hong Kong Research Grants Council (RGC) GRF project (RGC/HKBU12201620), Hong Kong Baptist University IG-FNRA project (RC-FNRA-IG/21-22/SCI/01), and partially supported by NSF IIS-1910154, 2007907, and 2046457. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

## REFERENCES

[1] Akari Asai, Mohammadreza Salehi, Matthew Peters, and Hannaneh Hajishirzi. 2022. ATTEMPT: Parameter-Efficient Multi-task Tuning via Attentional Mixtures of Soft Prompts. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 6655–6672. https://aclanthology.org/2022.emnlp-main.446

[2] Eyal Ben-David, Nadav Oved, and Roi Reichart. 2022. PADA: Example-based Prompt Learning for on-the-fly Adaptation to Unseen Domains. *Transactions of the Association for Computational Linguistics* 10 (04 2022), 414–433.

[3] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Advances in neural information processing systems*.

[4] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 535–541.

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[6] Zeyu Cui, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. 2022. M6-Rec: Generative Pretrained Language Models are Open-Ended Recommender Systems. *arXiv preprint arXiv:2205.08084* (2022).

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

[8] Li Dong, Shaohan Huang, Furu Wei, Mirella Lapata, Ming Zhou, and Ke Xu. 2017. Learning to generate product reviews from attributes. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. 623–632.

[9] 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 *Sixteenth ACM Conference on Recommender Systems*.

[10] Shijie Geng, Juntao Tan, Shuchang Liu, Zuohui Fu, and Yongfeng Zhang. 2023. VIP5: Towards Multimodal Foundation Models for Recommendation. *arXiv preprint arXiv:2305.14302* (2023).

[11] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *International Conference on Learning Representations (ICLR)*.

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

[13] Yupeng Hou, Shanlei Mu, Wayne Xin Zhao, Yaliang Li, Bolin Ding, and Ji-Rong Wen. 2022. Towards Universal Sequence Representation Learning for Recommender Systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 585–593.

[14] Wenyue Hua, Yingqiang Ge, Shuyuan Xu, Jianchao Ji, and Yongfeng Zhang. 2023. UP5: Unbiased Foundation Model for Fairness-aware Recommendation. *arXiv preprint arXiv:2305.12090* (2023).

[15] Wenyue Hua, Shuyuan Xu, Yingqiang Ge, and Yongfeng Zhang. 2023. How to Index Item IDs for Recommendation Foundation Models. *arXiv preprint arXiv:2305.06569* (2023).

[16] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*. IEEE, 197–206.

[17] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[18] Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Brussels, Belgium, 66–71. https://doi.org/10.18653/v1/D18-2012

[19] Lei Li, Yongfeng Zhang, and Li Chen. 2020. Generate neural template explanations for recommendation. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 755–764.

[20] Lei Li, Yongfeng Zhang, and Li Chen. 2021. Extra: Explanation ranking datasets for explainable recommendation. In *Proceedings of the 44th International ACM SIGIR conference on Research and Development in Information Retrieval*. 2463–2469.

[21] Lei Li, Yongfeng Zhang, and Li Chen. 2021. Personalized Transformer for Explainable Recommendation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*. 4947–4957.

[22] Lei Li, Yongfeng Zhang, and Li Chen. 2023. Personalized prompt learning for explainable recommendation. *ACM Transactions on Information Systems* 41, 4 (2023), 1–26.

[23] Piji Li, Zihao Wang, Zhaochun Ren, Lidong Bing, and Wai Lam. 2017. Neural rating regression with abstractive tips generation for recommendation. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 345–354.

[24] Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*.

[25] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[26] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.

[27] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. GPT understands, too. *arXiv preprint arXiv:2103.10385* (2021).

[28] Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

[29] Chen Ma, Peng Kang, and Xue Liu. 2019. Hierarchical gating networks for sequential recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 825–833.

[30] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[31] 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. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[32] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *International Conference on Machine Learning*. PMLR, 8821–8831.

[33] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. 452–461.

[34] 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*. ACM, 285–295.

[35] 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 *Proceedings of the 28th ACM international conference on information and knowledge management*. 1441–1450.

[36] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the eleventh ACM international conference on web search and data mining*. 565–573.

[37] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[38] 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 *Advances in neural information processing systems*.

[39] Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou, and Daniel Cer. 2022. Spot: Better frozen model adaptation through soft prompt transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*.

[40] Xiaolei Wang, Xinyu Tang, Wayne Xin Zhao, Jingyuan Wang, and Ji-Rong Wen. 2023. Rethinking the Evaluation for Conversational Recommendation in the Era of Large Language Models. *arXiv preprint arXiv:2305.13112* (2023).

[41] Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, Huan Sun, and Yoon Kim. 2023. Multitask prompt tuning enables parameter-efficient transfer learning. In *International Conference on Learning Representations (ICLR)*.

[42] Shuyuan Xu, Wenyue Hua, and Yongfeng Zhang. 2023. OpenP5: Benchmarking Foundation Models for Recommendation. *arXiv preprint arXiv:2306.11134* (2023).

[43] Tingting Zhang, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Deqing Wang, Guanfeng Liu, and Xiaofang Zhou. 2019. Feature-level Deeper Self-Attention Network for Sequential Recommendation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. 4320–4326.

[44] Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. 2014. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. 83–92.

[45] Yongfeng Zhang, Haochen Zhang, Min Zhang, Yiqun Liu, and Shaoping Ma. 2014. Do users rate or review? Boost phrase-level sentiment labeling with review-level sentiment classification. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. 1027–1030.

[46] Qihuang Zhong, Liang Ding, Juhua Liu, Bo Du, and Dacheng Tao. 2022. Panda: Prompt transfer meets knowledge distillation for efficient model adaptation. *arXiv preprint arXiv:2208.10160* (2022).

[47] 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 *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1893–1902.