COMP5331 Knowledge Discovery in Databases

Learning Language Representations for Sequential Recommendation

Group Number: 3

Group Members: CHEN Xiao, LI Tsz On, XU Congying, CHEN Songqiang, LU Weiqi, XU Mingshi

Project Information

Project Type:

Implementation-oriented (with an extra modification to the model structure)

Paper to Implement:

Jiacheng Li, Ming Wang, Jin Li, Jinmiao Fu, Xin Shen, Jingbo Shang, and Julian McAuley. 2023. Text Is All You Need: Learning Language Representations for Sequential Recommendation. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23). Association for Computing Machinery, New York, NY, USA, 1258–1267. https://doi.org/10.1145/3580305.3599519pp

Group Member Information

Student ID	Name	Email	Supervisor
20881149	CHEN, Xiao	xchenfu@connect.ust.hk	Prof. Shing-Chi CHEUNG
20783957	LI, Tsz On	toli@connect.ust.hk	Prof. Shing-Chi CHEUNG
20881462	XU, Congying	cxubl@connect.ust.hk	Prof. Shing-Chi CHEUNG
20973370	CHEN, Songqiang	i9s.chen@connect.ust.hk	Prof. Shing-Chi CHEUNG
20678308	LU, Weiqi	wluak@connect.ust.hk	Prof. Shing-Chi CHEUNG
20678140	XU, Mingshi	mxuax@connect.ust.hk	Prof. Wilfred Siu Hung NG

Content

FYP/Research Topics of Group Members

CHEN Xiao

- FYP/research topic: program analysis, software testing.
- Explanation: My FYP and current research topic is focusing on analyzing Java programs for better fault localization and program repair. So it is not related to this project.

LI Tsz On

- FYP/research topic: Nuances are the Key: Unlocking ChatGPT to Find Failure-Inducing Tests with Differential Prompting
- Explanation: My research topic is about finding failure-inducing test cases using ChatGPT. This research topic is not related to recommendation systems or the design of transformer-based models.

XU Congying

- FYP/research topic: Automatically Deriving Metamorphic Relations from Open-Source Software Projects.
- Explanation: My research topic is about mining knowledge from existing programs for testing software. The research problem is not related to the recommendation systems and the design of transformer-based models.

CHEN Songqiang

- FYP/research topic: Test and repair for the LLM-generated code snippets.
- Explanation: My research is about designing (semi-)automated methods to test and repair the code snippets generated by the large language models. The research problem are not related to the recommendation systems and the design of transformer-based models.

LU Weiqi

- FYP/research topic: Automated Bug Reproduction from Reports.
- Explanation: My research topic is about automatically reproducing bugs from user reports, possibly applied in Java and Android projects. The research problem and subjects are not related to the recommendation systems and transformers.

XU Mingshi

- FYP/research topic: Financial Time Series Data Forecasting based on Deep Learning Tools.
- Explanation: My research topic focuses on mining the complex structural information in the financial time series data using deep learning models such as GNN. The research topic is not related to recommendation models and transformers.

Contribution

CHEN Xiao

- Implementing the base model of Recformer, which is the core of the whole methodology.
- Implementing the four embedding layers to adeptly understand item patterns by combining the idea of embedding layers from language models.
- Implement the representations of item and sequences based on Longformer to integrate the advantages of self-attentive sequential recommenders.
- Writing Section 1.2 (Problem Definition), the overview of Section 2 (Methodology) and Section 2.1.1 (Model Architecture).

LI Tsz On

- Implementing scripts for finetuning Recformer (finetune.py, utils.py, optimization.py). These files contain the loading of dataset/model, training of model (with specific configuration of gradients), computation of evaluation metrics etc
- Implementing a script for conducting an ablation study for Recformer, and conducting the corresponding experiments. Specifically, I create five model variants, by enabling/disabling certain components of Recformer (implemented in finetune.py). Then I evaluate each of the model's performance.
- Writing Section 3.5 (Ablation Study). This section compares the performances of the five model variants, in order to compare the performances. This section also discusses the insight behind the discrepancies of the performances.

XU Congying

- Implementing the scripts to pre-train models and evaluate the performance of Recformer on processed datasets (i.e., lightning_pretrain.py, and litmodels.py). These files build up a pipeline of data preprocessing, model training, and performance evaluation, utilizing PyTorch Lightning for efficient training management and TensorBoard for logging.
- Training Recformer, finely tuning hyperparameters to achieve maximum learning efficiency and accuracy.
- Conducting a thorough evaluation of trained models, assessing not only their overall performance but also their zero-shot capabilities to show knowledge transferability.
- Writing Sections 3.2 - 3.4 (Experiment Setup, Overall Performance, and Zero-shot Performance) of this report.

CHEN Songqiang

- Proposing the idea of replacing the base-sized Longformer language model for a smaller language model for much lower GPU memory usage and much faster execution speed, as well as finalizing the selection of the mini-sized Longformer model as the subject.
- Analyzing and making the code adaption to fit the language model substitution, including tuning the model architecture (layer number), changing the tokenizer, as well as the according miscellaneous adaption.
- Performing a series of preliminary performance exploration experiments, including the efficiency evaluation of the GPU memory usage and execution time, and the parameter adjusting like batch size tuning and the learning rate adjustment, so as to understand the potential of the language model substitution.
- Writing Section 2.2.2 (Recformer Model for Pre-Training) and Section 2.2.3 (Beyond Reimplementation: Modify Model for Lower Resource & Time Cost) of this report.

LU Weiqi

- Reading and selecting the paper that formed the basis of our implementation.
- Feasibility analysis of the pre-training and fine-tuning processes for the Recformer model, which motivates the proposal of Longformer-mini model substitution.
- Write scripts of data processing and loading, including collection of Amazon Review Data, pre-processing to convert discrete review data into item sequences of users (pretrain_data/preprocess.py, finetune_data/preprocess.py), tokenizing item sequences for both the pre-training and fine-tuning stages (recformer/tokenization.py), and data-loaders for both pre-training and fine-tuning (dataloader.py).
- Writing Section 1 (Introduction) including Motivation and Contribution, and Section 3.1 (Datasets) mentioning data information and statistics.

XU Mingshi

- Implementing the collator module (collator.py) in the code to prepare data for pre-training, finetuning, validation and testing. This module converts all raw data into batches and masks the tokens as required for Masked Language Modeling (MLM) according to the methodology specified in the pre-training section.
- Constructing the pre-training model (part of models.py) by implementing the model structure and building the pre-training procedure according to the methodology outlined in the research paper.
- Writing Section 2.2(Learning Framework) including Section 2.2.1(Pre-training) and Section 2.2.2(Finetuning). Pre-training mainly revolves around Masked Language Modeling MLM and item-item contrastive (IIC). Finetuning mainly about the two stage finetuning mechanism.

1 Introduction

1.1 Motivation

Our project centers on developing Recformer, an innovative approach for enhancing language representation and comprehension in the context of sequential recommendation systems. Sequential recommendation is a process where the system suggests items to users based on their past interactions. For example, given that a user bought "MacBook" and "Mouse" before, he/she is likely to buy "iPhone" as illustrated in Figure 1, so the sequential recommendation system will suggest "iPhone".

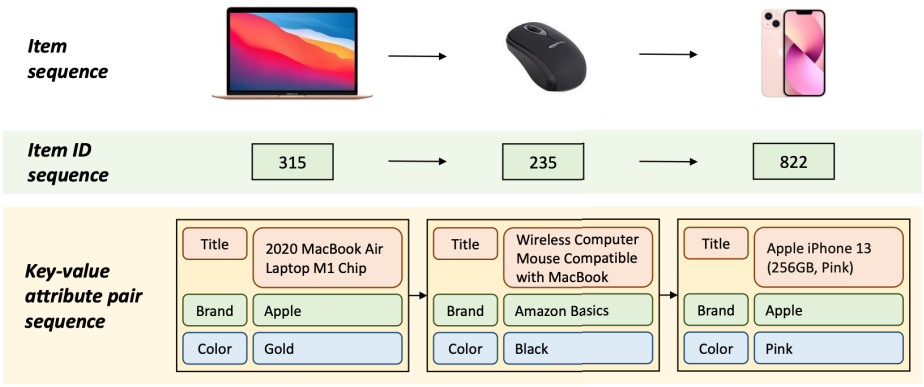

Figure 1: An Example of Item Sequence

Traditional methods in this domain utilize either learnable item embedding tables or pre-trained language models to infer potential item suggestions. For the learnable item embedding table approach, it suffers from the cold-start problem: the learned item embeddings are only applicable to the current application scenario, and cannot be generalized to other domains. For the pre-trained language model approach, even though it can be generalized to all item types for text representation, such a model is not dedicated to sequential recommendation but serves as a model of general language understanding. Therefore, the text representation given by pre-trained language models is usually sub-optimal. Moreover, the text representations produced by these models do not adequately capture the significance of text labels in the recommendation context and are generally at a too coarse-grained level, focusing on sentences rather than individual words. Another limitation is due to the independent training of pre-trained models and downstream models, there exists a semantic gap between the pre-trained models and the downstream recommendation models, limiting the overall language understanding capabilities of the system for recommendation purposes.

In this work, we implement a recommendation model called Recformer proposed by Li et al. [9] to address the aforementioned limitations. This framework encodes items by their key-value pairs of attributes, offering a more detailed text representation of items than previous methodologies. The Recformer leverages an advanced bi-directional Transformer architecture based on the Longformer structure to efficiently learn the critical item features from text labels for sequential recommendation. Moreover, it incorporates a distinctive learning framework that encompasses model pre-training and fine-tuning for learning both item and sequence representations in the same semantic space. Based on the embedding distances between a sequence and all items, the Recformer model can effectively recommend the most probable item to a user.

1.2 Problem Definition

Given an item set I and a user's chronological interaction sequence $s = \{i_1, i_2, ..., i_n\} (i \in I)$, the problem we want to solve is predicting the next item based on the sequence s.

- Each item i_k is described by a dictionary D_k consisting of attribute pairs $\{(k_1, v_1), (k_2, v_2), ..., (k_m, v_m)\}$, where the key k is an attribute name and the value v is the corresponding value of this attribute.
- Both attributes and values are described by natural languages with words, denoted as $(k, v) = \{w_1^k, ..., w_s^k, w_{s+1}^v, ..., w_t^v\}$, where w^k and w^v are words of k and v from a shared vocabulary and t is the truncated length of the attribute pair.
- We flatten the attribute pairs into a single "sentence" $T_i = \{k_1, v_1, k_2, v_2, ..., k_m, v_m\}$ as input data of the language model.

1.3 Contribution

The contribution of our work is the re-implementation of the Recformer in line with the original paper, but **with a notable modification**: replacing the standard Longformer model with a more compact variant, the Longformer-mini. This adaptation aims to achieve similar results to the larger Longformer model while significantly reducing computational costs and enhancing inference efficiency. This modification highlights our commitment to creating a more efficient yet equally effective solution in the realm of sequential recommendation systems.

2 Methodology

The general workflow of Recformer begins with formulating text labels into key-value pairs, offering a more granular text representation than earlier models. Utilizing a bidirectional

Transformer based on Longformer's architecture, Recformer is able to learn text labels' essential features (embeddings) adeptly from these text labels for sequential recommendation. Finally, Recformer uses a novel learning framework for model pre-training, finetuning, and inferring both text presentations and item representations.

In this section, we will present the principles of methodology from two aspects, including the model architecture and the learning framework.

2.1 Model Architecture

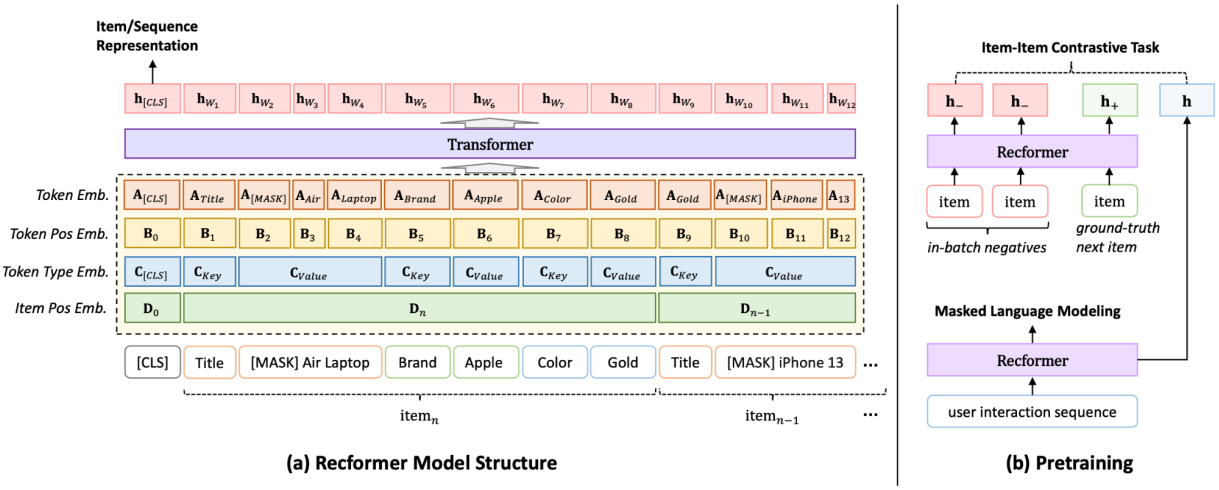

Figure 2: The Overall Framework of Recformer

2.1.1 Recformer Model

Recformer utilizes a multi-layer bidirectional Transformer equipped with an attention mechanism for linear scaling with sequence length. This idea is reminiscent of Longformer [10] considering computational efficiency while the method is open to other bidirectional Transformer structures.

Model Inputs. First, Recformer reverses item sequence after the flattening based on the intuition that latest items are more important for the prediction of next item. Then, we add a special token [CLS] at the beginning of the sequence of item sequences to form the model input $X = \{[CLS], T_n, T_{n-1}, ..., T_1\}$.

Embedding Layer. Recformer combines the embedding layers from both language models and self-attentive sequential recommenders in order to understand the input sequence from both

language perspective and sequential patterns for recommendation. The architecture consists of four embeddings.

- **Token embedding** represents tokens as $A \in \mathbb{R}^{V_w \times d}$, where V_w denotes the vocabulary word number and d is the embedding dimension. Recformer relies on token embeddings for item understanding in sequences unlike previous sequential recommenders, maintaining a consistent model size regardless of the item count.
- **Token position embedding** represents the i-th word in sequence X as $B_i \in \mathbb{R}^d$. It aims at helping the Transformer better understand the word sequential patterns.
- **Token type embedding** summarizes the attribute type information of each token, i.e., keys (C_{Key}), values (C_{Value}) or [CLS] ($C_{[CLS]}$). As different token types vary in importance for predicting the next item, models with token type embedding are able to recognize recurring words from consistent attribute keys in recommendation datasets.
- **Item position embedding** denotes a word from attributes of the k-th item in sequence X as $D_k \in \mathbb{R}^d$ and $D \in \mathbb{R}^{n \times d}$, where n is the maximum length of the user's interaction sequence s. Recformer leverages item position embedding for sequential pattern learning and aligning word tokens with items, just like earlier self-attentive sequential recommenders.

In summary, given a word w from the input sequence X, the input embedding is the sum of four embeddings with layer normalization, i.e., $E_w = LayerNorm\left(A_w + B_w + C_w + D_w\right)$, where $E_w \in \mathbb{R}^d$. So the embedding of model input can be represented as $E_{[CLS]}, E_{w_1}, \ldots, E_{w_l}$, where $E_X \in \mathbb{R}^{(l+1) \times d}$ and l is the maximum length of the user's interaction sequence tokens.

Item or Sequence Representations. We utilize Longformer's bidirectional Transformer to encode E_X in consideration of its high efficiency of handling long sequences. According to the standard settings for document comprehension in Longformer, only the [CLS] token has global attention while other tokens have local windowed attention. Therefore, the d-dimensional word representations are calculated as $\left[h_{[CLS]}, h_{w_1}, \ldots, h_{w_l}\right] = Longformer\left(\left[E_{[CLS]}, E_{w_1}, \ldots, E_{w_l}\right]\right)$, where $h_w \in \mathbb{R}^d$.

Prediction. Recformer predicts the next item based on the cosine similarity between the item i and the user's interaction sequence s, i.e., $r_{i,s} = \frac{h_i^\top h_s}{\|h_i\| \cdot \|h_s\|}$, where h_i is the item representation, h_s is the sequence representation and $r_{i,s} \in \mathbb{R}$ is the relevance of item i being the next item given s. Finally, we select the predicted item $\hat{i}_s$ with the highest relevance as the next item, i.e., $\hat{i}_s = argmax_{i \in I}(r_{i,s})$.

2.1.2 Recformer Model For Pre-Training

As introduced in Section 2.1.1, the Recformer model is an encoding model, which encodes the given item or item sequence into a vector. The pre-training phase of Recformer requires this feature to perform a pre-training task. Besides, the pre-training phase further requires a token-level classification model architecture to perform another pre-training task of the Masked Language Modelling (MLM, the details about the MLM task will be introduced in Section 2.2.1). MLM cannot be performed with the original Recformer which is an encoding model.

To support MLM, we build the RecformerForPreTraining model. RecformerPreTraining model contains the whole original Recformer model. In addition, it adds a fully-connected linear module on top of the Recformer model to perform the token-level classification based on the last layer output of Recformer. All the tokens share the same classification module. For the last layer output $\{h_{[CLS]}, h_{w_1}, h_{w_2}, ..., h_{w_l}\}$, for each token, the classification module computes:

$$m = LayerNorm(W_h h_w + b_h);\ p_w = Softmax(W_0 m + b_0)$$

where $W_h \in \mathbb{R}^{d \times d}$, $b_h \in \mathbb{R}^d$, $W_0 \in \mathbb{R}^{|V| \times d}$, $b_h \in \mathbb{R}^{|V|}$.

As a result, the RecformerForPreTraining model can output a vector of $h_{[CLS]}$ to encode the whole input sequence, as well as classification probability distribution vectors for each of the input token p_w.

2.1.3 Beyond Reimplementation: Modify Model for Lower Resource & Time Cost

In our implementation, we go further than simply reimplementing the methodology introduced in the original literature. We modify the language model used in Recformer to save the resource and time budget. Specifically, the original literature adopts the Longformer model of "base" size to build Recformer. However, this model architecture is quite large and costs lots of resources and time to run. According to our statistics, the suggested setup of using a batch size of 16 in one GPU and a maximum token length of 1024 consumes 33.6 GB and 25.6 GB for pre-training and fine-tuning, respectively. This cost cannot be supported even by the premium consumer-level

GPUs NVIDIA RTX 3090/4090, which have 24 GB memory and are also the best device we can frequently use. Meanwhile, the time cost is also huge. It takes 49.5 hours to finish a pre-training epoch, and over 0.5 hour to finish a fine-tuning epoch (there could be over 256 fine-tuning epochs needed).

Thus, in our implementation, we try to adjust the model structure to reduce the cost of resources and time. This modification is meaningful for two-fold reasons. On one hand, we can only run the model with our device by reducing the GPU memory usage of model. On the other hand, we try to cut off the time budget to process one sample so that both the training and predicting will be more efficient and low-cost. This helps us obtain more results more efficiently. More importantly, **it contributes to the practicality of the methodology proposed in the original literature**. In our work, we use two strategies to realize this goal.

Table 1: Resource and Time Cost When Using Different Model Structures / Parameters

LongFormer Size	Max Input Length	Batch Size	Pre-Training		Fine-Tuning	
			GPU Mem	Time/Epoch	GPU Mem	Time/Epoch
Base	1024	16	33,660 MB	49.5 hours	25,656 MB	1800+ sec
Base	512	16	19,958 MB	41 hours	19,966 MB	1600+ sec
Mini	512	16	5,932 MB	14.5 hours	6,060 MB	502 sec
Mini	512	80	20,248 MB	5 hours	17,552 MB	312 sec

The first strategy we adopt is to limit the maximum number of input tokens to 512. The fewer tokens input to the model, the fewer hidden state storage and calculations are needed. The authors have suggested that truncating the input to 512 tokens will not cause obvious performance decay. Specifically, given the text description sequence for an item sequence, we truncate it into 512 tokens before feeding it to the model.

As shown in the third row in Table 1, by this means, we reduce the GPU memory usage in pre-training and fine-tuning phases from 33.6GB and 25.6GB to 19.9GB and 19.9GB, respectively. Now we can use NVIDIA RTX 3090/4090 whose memory is 24GB to run the proposed model. The time cost also reduces moderately.

Since the time and resource budgets are still fairly high, we try to further reduce them by considering to reduce the model size. We realize this goal by substituting the core, i.e., the Longformer language model, in Recformer model. However, we found that the Longformer authors only release the pretrained wights for the model at size of base (i.e., the current model, denoted as Longformer-base later) and large (i.e., a more resource demanding model). Fortunately, the third-party researcher releases a mini-sized Longformer model (denoted as Longformer-mini later). It is possible to use the other model, such as BERT, as the language

model. But we do not want to lose the feature of the advanced Longformer model architecture. Thus, we choose to use the third-party Longformer-mini.

As shown in Figure 3, in comparison, the Longformer-base model differs from the Longformer-mini model in 3 main aspects. Specifically, as the content in red shows, Longformer-base includes 12 Transformer self-attention layers while Longformer-mini uses only 6 such layers. Besides, as the content in yellow shows, the hidden states of the Longformer-base model is a 768d vector while that of the Longformer-mini model is a 256d vector. In addition, as the content in skyblue shows, the Longformer-base model uses a GPT-tokenizer and saves embeddings for 50265 tokens; while Longformer-mini uses a more advanced BERT-tokenizer and only saves embeddings for 30522 tokens. We make corresponding adaptions to our code to build the Longformer-mini model. With these reductions, the parameter number of Recformer model decreases from 148M to 14M. The hidden states for a sample will decrease from roughly 12*768d vectors to 6*256d vectors, saving 6 times of space. We build a Recformer model on the basis of such a Longformer-mini model and we call our model as **Recformer-mini**.

As shown in the third row in Table 1, by this means, we reduce the GPU memory usage in the pre-training and fine-tuning phases from 33.6 GB and 25.6 GB to 19.9 GB and 19.9 GB, respectively. Now we can easily employ an NVIDIA RTX 3090/4090 with 24 GB GPU memory to run the proposed model.

```
Recformer-Base(
  (embeddings): RecformerEmbeddings( # param: 41,794,560
    (word_embeddings): Embedding(50265, 768)
    (position_embeddings): Embedding(4098, 768)
    (token_type_embeddings): Embedding(4, 768)
    (item_position_embeddings): Embedding(51, 768)
    (LayerNorm): LayerNorm((768,))
  )
  (encoder): LongformerEncoder( # LongFormer-Base
    (layer): ModuleList(
      (0): LongformerLayer( # param: 8,859,648 x 12 ls = 106,315,776
        (attention): LongformerAttention(
          (self): LongformerSelfAttention(
            (query): Linear(in_features=768, out_features=768)
            (key): Linear(in_features=768, out_features=768)
            (value): Linear(in_features=768, out_features=768)
            (query_global): Linear(in_features=768, out_features=768)
            (key_global): Linear(in_features=768, out_features=768)
            (value_global): Linear(in_features=768, out_features=768)
          )
          (output): LongformerSelfOutput(
            (dense): Linear(in_features=768, out_features=768)
            (LayerNorm): LayerNorm((768,))
          )
        )
        (intermediate): LongformerIntermediate(
          (dense): Linear(in_features=768, out_features=3072)
        )
        (output): LongformerOutput(
          (dense): Linear(in_features=3072, out_features=768)
          (LayerNorm): LayerNorm((768,))
        )
      )
      (1): LongformerLayer(...)
      ...
      (11): LongformerLayer(...)
    )
  )
  (pooler): RecformerPooler()
)
```

```
Recformer-Mini(
  (embeddings): RecformerEmbeddings( # param: 8,090,880
    (word_embeddings): Embedding(30522, 256)
    (position_embeddings): Embedding(1026, 256)
    (token_type_embeddings): Embedding(4, 256)
    (item_position_embeddings): Embedding(51, 256)
    (LayerNorm): LayerNorm((256,))
  )
  (encoder): LongformerEncoder( # LongFormer-Mini
    (layer): ModuleList(
      (0): LongformerLayer( # param: 987,136 x 6 layers = 5,922,816
        (attention): LongformerAttention(
          (self): LongformerSelfAttention(
            (query): Linear(in_features=256, out_features=256)
            (key): Linear(in_features=256, out_features=256)
            (value): Linear(in_features=256, out_features=256)
            (query_global): Linear(in_features=256, out_features=256)
            (key_global): Linear(in_features=256, out_features=256)
            (value_global): Linear(in_features=256, out_features=256)
          )
          (output): LongformerSelfOutput(
            (dense): Linear(in_features=256, out_features=256)
            (LayerNorm): LayerNorm((256,))
          )
        )
        (intermediate): LongformerIntermediate(
          (dense): Linear(in_features=256, out_features=1024)
        )
        (output): LongformerOutput(
          (dense): Linear(in_features=1024, out_features=256)
          (LayerNorm): LayerNorm((256,))
        )
      )
      (1): LongformerLayer(...)
      ...
      (5): LongformerLayer(...)
    )
  )
  (pooler): RecformerPooler()
)
```

Figure 3: Model Architecture of Recformer Based on Longformer-Base and Longformer-Mini

Since only 6GB GPU memory has been used, the GPU computation power has not been fully utilized. Thus, we consider increasing the batch size to further boost the evaluation efficiency. Actually, a larger batch size is also helpful to increase the stability of training progress because it can better guide the model to converge in the correct direction. According to the GPU memory consumption, we set the final batch size for our final model Recformer-mini to be 80, i.e., **5x the original batch size of 16**. As shown in the last row in Table 1, by fully utilizing the computation power of our GPU, we can finish an epoch of pre-training and fine-tuning in 5h and 312s, respectively, which is **roughly 10x and 6x speedup** in comparison to the original setup. This gives birth to the possibility of running the model with our device, as well as the sufficient efficiency of performing comprehensive evaluation.

As a reminder, in this section we only briefly discuss the improvement in terms of resource usage. The model performance comparison will be shown in Section 4.2.

As a reminder, a possible strategy to accommodate the model with fewer GPU memory usage is to reduce the data batch size. But this idea can be less effective to reducing the resource and time budget and thus we do not consider it. Specifically, reducing batch size does not reduce the time to process each sample. That is, in the inference scenario, reducing the batch size does not reduce the computation needed to process a sample. Moreover, the batch size of 16 has already been small and a too small batch size may degrade the training performance.

2.2 Learning Framework

To extract the semantic information from the sequential text input, this paper proposes a learning framework. Including pre-training and two-stage finetuning.

2.2.1 Pre-Training

The pretraining procedure has 2 tasks: (1) Masked Language Modeling(MLM) and (2) item-item contrastive(IIC). These two tasks are specially designed to extract both semantic information and item information for the recommendation.

Masked Language Modeling (MLM): In this part, we follow the idea of a commonly used language model pretraining procedure, the data generator selects 15% of the token positions at random for prediction. For all the selected tokens 1) the token is replaced with a [MASK] token with an 80% probability; (2) the token is substituted with a random token with a 10% probability; or (3) the token remains unchanged with a 10% probability. This masking strategy encourages the model to focus on the context of the sentence rather than relying solely on the masked tokens. By incorporating random tokens and retaining some original tokens, MLM

prevents the model from becoming too reliant on the presence of [MASK] tokens during pre-training, which in turn results in a more robust understanding of language and improved performance on downstream tasks.
The loss function of MLM is shown below, |V| is the vocabulary used in the language model.

$$L_{MLM} = -\sum_{i \in |V|} y_i \, log(p_i)$$

Item-item Contrastive(IIC): This task is commonly employed in predicting the next item for recommendations. In line with some previous works, this paper uses the ground-truth next items as positive instances. However, To accelerate the pretraining process, we use in-batch next items as opposed to negative sampling. Traditional recommenders maintain an item embedding table, allowing for easy retrieval and updating of item embeddings. In this paper, item embeddings originate from Recformer, making it impractical to re-encode items per batch for training. In-batch negative instances involve using ground truth items from other instance sequences within the same batch as negative items. Although this may occasionally result in false negatives, they are less likely in a pre-training dataset of considerable size.

$$L_{IIC} = -log\{exp[sim(h_s, h_i^+)/\tau] / \sum_{i \in B} exp[sim(h_s, h_i)/\tau]\}$$

Thus the loss function of the pretraining process is shown below, λ here is the hyperparameter that controls the MLM loss:

$$l_{PT} = L_{IIC} + \lambda \cdot L_{MLM}$$

2.2.2 Fine-Tuning

During the fine-tuning phase, this study employs a similar method to that used in the pre-training stage, with all items encoded by Recformer instead of retaining the embedding table. As previously mentioned, the negative instance generation method employed in the pre-training phase may result in some false negatives, prompting the use of the two-stage fine-tuning approach proposed in this paper. The primary objective of this method is still to hasten the training process, and the key concept is to divide the fine-tuning into two stages, as illustrated in Algorithm 1.
In the first stage, both the item encoding result and model parameters are updated until the most optimal item encoding results are obtained. The item representation matrix is denoted as 'I,' as shown in Algorithm 1. In the second stage, the item encoding remains unchanged, and the model parameters are iteratively updated. The fine-tuning loss function is presented below, where I_i represents the item feature of the ith item.

$$L_{FT} = -log\{exp[sim(h_s, I_i^+)/\tau] / \sum_{i \in I} exp[sim(h_s, I_i)/\tau]\}$$

Algorithm 1: Two-Stage Finetuning

Input: D_{train}, D_{valid}, $\mathcal{I}$, M
Hyper-parameters: n_{epoch}
Output: M', $\mathbf{I}'$

$M \leftarrow$ initialized with pre-trained parameters
$p \leftarrow$ metrics are initialized with 0
Stage 1
for n in n_{epoch} **do**
  $\mathbf{I} \leftarrow \text{Encode}(M, \mathcal{I})$
  $M \leftarrow \text{Train}(M, \mathbf{I}, D_{\text{train}})$
  $p' \leftarrow \text{Evaluate}(M, \mathbf{I}, D_{\text{valid}})$
  if $p' > p$ **then**
    $M', \mathbf{I}' \leftarrow M, \mathbf{I}$
    $p \leftarrow p'$
  end if
end for
Stage 2
$M \leftarrow M'$
for n in n_{epoch} **do**
  $M \leftarrow \text{Train}(M, \mathbf{I}', D_{\text{train}})$
  $p' \leftarrow \text{Evaluate}(M, \mathbf{I}', D_{\text{valid}})$
  if $p' > p$ **then**
    $M' \leftarrow M$
    $p \leftarrow p'$
  end if
end for
return M', $\mathbf{I}'$

3. Evaluation

3.1 Datasets

For training and evaluation of our model, we sample data from The Amazon Review Data (2018)[1], an expansive dataset featuring 233.1 million product reviews from May 1996 to October 2018. The dataset includes detailed metadata, such as product descriptions, brands, categories, and image features, enhancing its utility for sequential recommendation systems. This dataset's breadth and depth, encompassing a wide range of products and extensive user reviews, make it ideal for analyzing consumer behaviors and trends, crucial for developing accurate and personalized recommendation models. Its comprehensive nature allows for a nuanced understanding of user preferences over time.

[1] https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/

For the pre-training dataset, we sample from seven categories for training: “Automotive”, “Cell Phones and Accessories”, “Clothing Shoes and Jewelry”, “Electronics”, “Grocery and Gourmet Food”, “Home and Kitchen”, “Movies and TV”, and an extra category “CDs and Vinyl” for validation. For fine-tuning, we choose six categories for training and testing: “Arts, Crafts and Sewing”, “Industrial and Scientific”, “Musical Instruments”, “Office Products”, “Pet Supplies”, and “Video Games”. As indicated in the figure below, most item sequences exhibit a length of at least five, ensuring sufficient historical interaction data for meaningful and reliable predictions in both the training and testing phases.

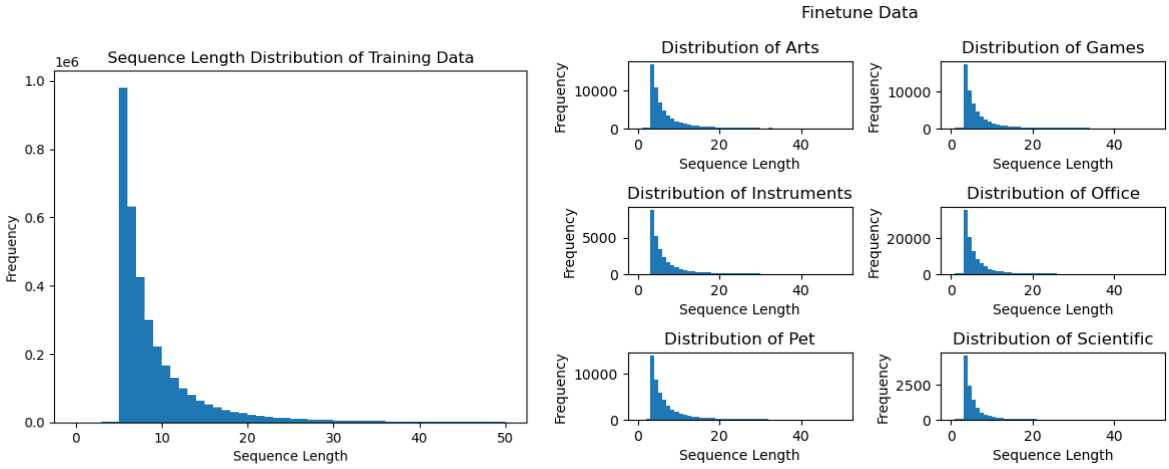

Figure 4: Sequence Length Distributions of Pre-training Dataset (Left) and Six Fine-tuning Datasets (Right)

The following is an example of item information containing the detailed description (title), brand, and fine-grained category:

```
7138258879: {'title': 'Elite Mailers 9&amp;quot;x2&amp;quot; i-VTEC SOHC Vinyl Decal Sticker - White - 2 pieces', 'brand': 'Elite Mailers', 'category': 'Automotive Exterior Accessories Bumper Stickers, Decals & Magnets'}
```

The item information is flatten into one sentence containing attribute keys and values as mentioned in Problem Definition. Since Longformer-mini is based on BERT, we use BertTokenizerFast to tokenize the flattened sentence of an item, and truncate the attribute values if their lengths exceed the user-defined threshold. Apply this process to all items in an interaction sequence, then we obtain the token embeddings of a sequence as input data.

3.2 Experimental Setup

3.2.1 Metrics

To evaluate the effectiveness of Recformer-mini in sequential recommendation, we employ three widely used metrics NDCG@N, Recall@N and MRR (N: the considered top N items for evaluation).

- **NDCG** (Normalized Discounted Cumulative Gain) measures the ranking quality of a list of items by considering both the relevance and the position of the items. A higher NDCG score indicates a better ranking of the items.
- **Recall** measures the proportion of relevant items that are successfully retrieved out of all the relevant items available. A higher Recall score indicates a higher proportion of relevant items being retrieved.
- **MRR** (Mean Reciprocal Rank) measures the average of the reciprocal ranks of the first relevant item in a ranked list. In other words, MRR measures how quickly the first relevant item is found. A higher MRR score indicates a system that is able to rank relevant items higher on average.

3.2.2 Implementation Details

We build Recformer-mini based on Longformer-mini[2] implemented by user kiddothe2b at the Huggingface platform. Following the setting of the original paper, we set the size of the local attention windows in Longformer-mini to 64. The maximum number of tokens is 32 for each attribute and 512 for each interaction sequence. The maximum number of items in a user sequence is 50. The temperature parameter τ is 0.05 and the weight of MLM loss λ is 0.1. The batch size is 80. We optimize Recformer with Adam optimizer with a learning rate of 1e-5 for pretraining and 1e-4 for fine-tuning and adopt an early stop with the patience of 5 epochs to prevent overfitting.

3.3 Overall Performance

We conduct a comparative analysis between Recformer-mini, Recformer, and other selected baselines, which include ID-only methods: “GRU4Rec” [1], “SASRec” [2], “BERT4Rec” [3], “RecGRU” [4], ID-Text methods: “FDSA” [5], “S3-Rec” [6], and Text-only methods “ZESRec” [7] and “UniSRec” [8]. This comparative evaluation allows us to assess the performance of Recformer-mini in relation to these baselines and gain insights into its strengths and weaknesses.

[2] https://huggingface.co/kiddothe2b/longformer-mini-1024

As shown in Table 2, our implemented Recformer-mini demonstrates superior performance when compared to the majority of the baselines. In fact, it achieves comparable results to Recformer on six datasets, which highlights its effectiveness in sequential recommendation tasks.

Table 2: Performance Comparison of Recformer-mini, Recformer and Baselines
(outperforming all baselines is in light green, further outperforming Recformer is in dark green)

Dataset	Metric	GRU4Rec	SASRec	BERT4Rec	RecGRU	FDSA	S3-Rec	ZESRec	UniSRec	Recformer	Recformer -mini
Scientific	NDCG@10	0.0826	0.0797	0.0790	0.0575	0.0716	0.0451	0.0843	0.0862	0.1027	0.1040
	Recall@10	0.1055	0.1305	0.1061	0.0781	0.0967	0.0804	0.1260	0.1255	0.1448	0.1451
	MRR	0.0702	0.0696	0.0759	0.0566	0.0692	0.0392	0.0745	0.0786	0.0951	0.0967
Instruments	NDCG@10	0.0633	0.0634	0.0707	0.0468	0.0731	0.0797	0.0694	0.0785	0.0830	0.0805
	Recall@10	0.0969	0.0995	0.0972	0.0617	0.1006	0.1110	0.1078	0.1119	0.1052	0.1034
	MRR	0.0707	0.0577	0.0677	0.0460	0.0748	0.0755	0.0633	0.0740	0.0807	0.0780
Arts	NDCG@10	0.1075	0.0848	0.0942	0.0525	0.0994	0.1026	0.0970	0.0894	0.1252	0.1179
	Recall@10	0.1317	0.1342	0.1236	0.0742	0.1209	0.1399	0.1349	0.1333	0.1614	0.1539
	MRR	0.1041	0.0742	0.0899	0.0488	0.0941	0.1057	0.0870	0.0798	0.1189	0.1113
Office	NDCG@10	0.0761	0.0832	0.0972	0.0500	0.0922	0.0911	0.0865	0.0919	0.1141	0.1114
	Recall@10	0.1053	0.1196	0.1205	0.0647	0.1285	0.1186	0.1199	0.1262	0.1403	0.1405
	MRR	0.0731	0.0751	0.0932	0.0483	0.0972	0.0957	0.0797	0.0848	0.1089	0.1055
Games	NDCG@10	0.0586	0.0547	0.0628	0.0386	0.0600	0.0532	0.0530	0.0580	0.0684	0.0637
	Recall@10	0.0988	0.0953	0.1029	0.0479	0.0931	0.0879	0.0844	0.0923	0.1039	0.0989
	MRR	0.0539	0.0505	0.0585	0.0396	0.0546	0.0500	0.0505	0.0552	0.0650	0.0601
Pet	NDCG@10	0.0648	0.0569	0.0602	0.0366	0.0673	0.0742	0.0754	0.0702	0.0972	0.0958
	Recall@10	0.0781	0.0881	0.0765	0.0415	0.0949	0.1039	0.1018	0.0933	0.1162	0.1161
	MRR	0.0632	0.0507	0.0585	0.0371	0.0650	0.0710	0.0706	0.0650	0.0940	0.0922

On “Scientific” and “Office” datasets, Recformer-mini even slightly outperforms Recformer in terms of Recall@10 and MRR. This indicates that Recformer-mini not only achieves similar performance but also exhibits improved results in certain scenarios, specifically in terms of capturing relevant items and ranking them effectively.

These results show that, compared with Recformer, Recformer-mini is capable of delivering comparable performance in sequential recommendation tasks while requiring fewer computational resources and memory.

3.4 Zero-Shot Performance

To demonstrate the zero-shot recommendation performance of Recformer-mini, we conduct an evaluation that focuses on the effectiveness of pre-training. Specifically, we compare the performance of the Recformer-mini with and without fine-tuning.

Table 3: Performance Contribution of Pre-Training in Recformer-mini

(Significant contribution is in green)

Dataset	Metric	pretrain only	pretrain & fine-tune	Con.	Dataset	Metric	pretrain only	pretrain & fine-tune	Con.
Scientific	NDCG@10	0.0823	0.1040	79%	Office	NDCG@10	0.0476	0.1114	43%
	Recall@10	0.1259	0.1451	87%		Recall@10	0.0767	0.1405	55%
	MRR	0.0734	0.0967	76%		MRR	0.0417	0.1055	40%
Instruments	NDCG@10	0.0436	0.0805	54%	Games	NDCG@10	0.0426	0.0637	67%
	Recall@10	0.0700	0.1034	68%		Recall@10	0.0685	0.0989	69%
	MRR	0.0395	0.0780	51%		MRR	0.0386	0.0601	64%
Arts	NDCG@10	0.0692	0.1179	59%	Pet	NDCG@10	0.0523	0.0958	55%
	Recall@10	0.1153	0.1539	75%		Recall@10	0.0771	0.1161	66%
	MRR	0.0591	0.1113	53%		MRR	0.0468	0.0922	51%

As shown in Table 3, pre-training plays a significant role in the performance of Recformer-mini, contributing approximately 40% to 70% across various evaluation metrics. On the "Scientific" dataset, pre-training demonstrates an impressive contribution of up to 87%.

These findings highlight the knowledge transferability of Recformer-mini in different recommendation scenarios. The ability of the model to leverage pre-training and apply learned knowledge to new domains or tasks is a crucial aspect of its effectiveness. The contribution of pre-training shows the model's capacity to generalize and adapt its recommendations to diverse contexts.

3.5 Ablation Study

To evaluate the effectiveness of Recformers' components (e.g., item embeddings, fine-tuning strategy). We conduct an ablation study with 4 extra model setups (variants):

1. Recformer with **fixed item embedding** during training, and **trainable item embedding** during fine-tuning (Original Recformer)
2. Recformer with **fixed item embedding** during training, and **fixed item embedding** during fine-tuning
3. Recformer with **trainable item embedding** during training, and **fixed item embedding** during fine-tuning
4. Recformer **with training** and **without fine-tuning**
5. Recformer **without training** and **with fine-tuning**

In the implementation level, to make item embedding untrainable, we set requires_grad for item embedding to be False, vice versa. To skip the training stage, we save the model using torch.save() and torch.load() to separate these two processes. For (5), we initialize a model from scratch and directly perform fine-tuning.

We evaluate these five setups on six evaluation subjects: “Industrial and Scientific”, “Musical Instruments”, “Arts, Crafts and Sewing”, “Office Products”, “Video Games”, and “Pet”. These six subjects are domain-specific datasets from Amazon review datasets. Note that we evaluate four more subjects compared to the ablation study reported by Recformer’s paper (which evaluates only two subjects: “Industrial and Scientific” and “Musical Instruments”). The additional evaluation results can provide further insight into the effectiveness of Recformer.

We conduct this ablation study on 4 NVIDIA GeForce RTX 3090 GPU cards. Since time is limited, we only train the model for 5 epochs, and fine-tune the model for 20 epochs.

Table 4: Ablation Study with Different Variants of Recformer

(Best performance is in green)

Dataset	Metric	Variant (1)	Variant (2)	Variant (3)	Variant (4)	Variant (5)
Scientific	NDCG@10	0.0986	0.0984	0.0986	0.0867	0.0294
	Recall@10	0.1400	0.1394	0.1407	0.1321	0.0413
	MRR	0.0906	0.0903	0.0903	0.0767	0.0392
Instruments	NDCG@10	0.0660	0.0625	0.0666	0.0470	0.0162
	Recall@10	0.0902	0.0870	0.0911	0.0787	0.0314
	MRR	0.0624	0.0588	0.0627	0.0410	0.0143
Arts	NDCG@10	0.1003	0.0789	0.0966	0.0783	0.0536
	Recall@10	0.1462	0.1238	0.1464	0.1230	0.0880
	MRR	0.0900	0.0684	0.0855	0.0689	0.0470
Office	NDCG@10	0.0892	0.0530	0.0892	0.0538	0.0887
	Recall@10	0.1260	0.0831	0.1248	0.0843	0.1275
	MRR	0.0806	0.0473	0.0810	0.0467	0.0797
Games	NDCG@10	0.0556	0.0831	0.0562	0.0538	0.0195
	Recall@10	0.0863	0.0633	0.0865	0.0843	0.0375
	MRR	0.0524	0.0397	0.0533	0.0400	0.0191
Pet	NDCG@10	0.0886	0.0447	0.0878	0.0443	0.0363
	Recall@10	0.1109	0.0734	0.1097	0.0727	0.0463
	MRR	0.0841	0.0575	0.0834	0.0572	0.0352

Finding 1: Our results show that the implications brought by evaluation results on *Industrial and Scientific* and *Musical Instruments* are consistent with those on the rest of the subjects. For instance, *Industrial and Scientific* and *Musical Instruments* show that (1) or (3) always outperform the rest of the variants. Evaluation on the rest of the subjects also show similar results. This finding can be useful for speculating the generalizability of results reported in Recformer’s paper (which conducts an ablation study on only *Industrial and Scientific* and *Musical Instruments*).

Finding 2: Our results show that Recformer's original workflow (1) always outperforms the truncated workflows i.e., training-only (4) or fine-tuning-only (5). The results are intuitive and consistent with the results reported in Recformer's paper.

Finding 3: Our result shows that (3) sometimes outperforms the original Recformer (1), which is not completely consistent with the result reported in Recformer's paper. Specifically, in Recformer's paper, (1) always outperforms all other variants. In addition, in Recformer's paper, (2) sometimes outperforms (3), yet this situation is not observed in our results. One possible reason for these discrepancies is that we use a different training/fine-tuning parameters (e.g., number of epochs) compared to those in Recformer's paper. Another possible reason is due to randomness.

Finding 4: Our results show that even for the same subject (e.g., *Industrial and Scientific*), different metrics (e.g., Recall@10 and MRR) make different variants of Recformer have different rankings. For instance, (3) outperforms (1) with Recall@10, but they have an opposite rank with MRR. This finding is consistent with that reported by Recformer's paper. Nevertheless, even those variants have different ranks, the discrepancies between the differently-ranked variants are small (same as the results reported in Recformer's paper). Hence, the differences in rank do not necessarily indicate a significant difference in performance.

Reference

[1] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based Recommendations with Recurrent Neural Networks. CoRR abs/1511.06939 (2015).

[2] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. 2018 IEEE International Conference on Data Mining (ICDM) (2018), 197–206.

[3] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin,WenwuOu,andPengJiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. Proceedings of the 28th ACM International Conference on Information and Knowledge Management (2019).

[4] Chenglin Li, Mingjun Zhao, Huanming Zhang, Chenyun Yu, Lei Cheng, Guoqiang Shu, Beibei Kong, and Di Niu. 2021. RecGURU: Adversarial Learning of Generalized User Representations for Cross-Domain Recommendation. Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (2021).

[5] 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 International Joint Conference on Artificial Intelligence.

[6] Kun Zhou, Haibo Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Jirong Wen. 2020. S3-Rec: Self-Supervised Learning for Sequential Recommendation with Mutual Information Maximization. Proceedings of the 29th ACM International Conference on Information & Knowledge Management (2020).

[7] Hao Ding, Yifei Ma, Anoop Deoras, Bernie Wang, and Hao Wang. 2021. Zero-Shot Recommender Systems. ArXiv abs/2105.08318 (2021).

[8] Yupeng Hou, Shanlei Mu, Wayne Xin Zhao, Yaliang Li, BolinDing, and Jirong Wen. 2022. Towards Universal Sequence Representation Learning for Recommender Systems. Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2022).

[9] Jiacheng Li, Ming Wang, Jin Li, Jinmiao Fu, Xin Shen, Jingbo Shang, and Julian McAuley. 2023. Text Is All You Need: Learning Language Representations for Sequential Recommendation. Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23).

[10] Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The Long- Document Transformer. ArXiv abs/2004.05150 (2020).