Efficient Item ID Generation for Large-Scale LLM-based Recommendation

Anushya Subbiah
Google Research
Mountain View, USA
anushyas@google.com

Vikram Aggarwal
Google Research
Mountain View, USA
viki@google.com

James Pine
Google Research
Mountain View, USA
rubicon@google.com

Steffen Rendle
Google Research
Mountain View, USA
srendle@google.com

Krishna Sayana
Google Research
Mountain View, USA
ksayana@google.com

Kun Su
Google Research
Mountain View, USA
sukun@google.com

ABSTRACT

Integrating product catalogs and user behavior into LLMs can enhance recommendations with broad world knowledge, but the scale of real-world item catalogs, often containing millions of discrete item identifiers (Item IDs), poses a significant challenge. This contrasts with the smaller, tokenized text vocabularies typically used in LLMs. The predominant view within the LLM-based recommendation literature is that it is infeasible to treat item ids as a first class citizen in the LLM and instead some sort of tokenization of an item into multiple tokens is required. However, this creates a key practical bottleneck in serving these models for real-time low-latency applications.

Our paper challenges this predominant practice and integrates item ids as first class citizens into the LLM. We provide simple, yet highly effective, novel training and inference modifications that enable *single-token* representations of items and *single-step* decoding. Our method shows improvements in recommendation quality (Recall and NDCG) over existing techniques on the Amazon shopping datasets while significantly improving inference efficiency by 5x-14x. Our work offers an efficiency perspective distinct from that of other popular approaches within LLM-based recommendation, potentially inspiring further research and opening up a new direction for integrating IDs into LLMs. Our code is available here https://drive.google.com/file/d/1cUMj37rV0Z1bCWMdhQ6i4q4eTRQLURtC

CCS CONCEPTS

• **Information systems → Recommender systems**; **Language models**.

KEYWORDS

Recommendation, LLM, Inference, topk, item catalog

1 INTRODUCTION

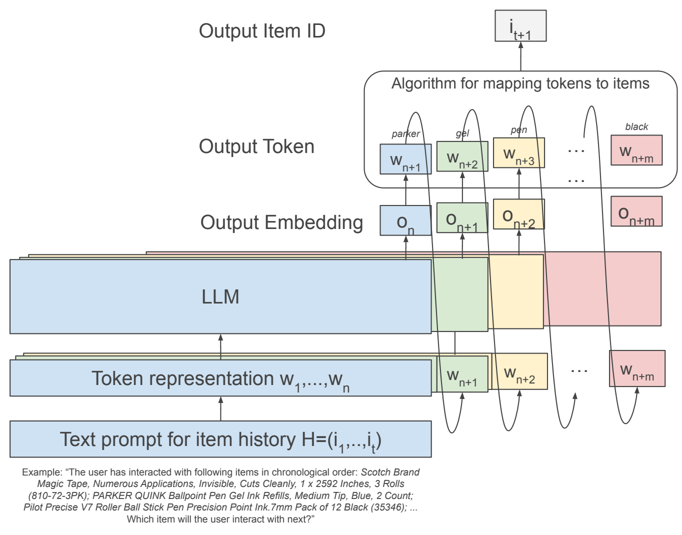
Recommender systems (RS) are crucial for online platforms, helping users navigate vast item catalogs [3]. Traditional RS often rely on collaborative filtering techniques like matrix factorization [17] and two-tower models [33] to capture user-item interactions. These methods learn latent representations for users and items, with the item embedding matrix scaling linearly with the catalog size ($|\mathcal{I}|$). While effective, these approaches may not fully utilize the rich semantic information in item descriptions and user preferences. Integrating LLMs into RS promises to combine collaborative filtering with semantic understanding, potentially leading to more accurate and context-aware recommendations.

Symbol	Description
$\mathcal{V}$	vocabulary of text tokens (from the LLM)
$\mathcal{I}$	catalog of all items
C	set of clusters
$x \in \mathcal{V}$	text token
$i \in \mathcal{I}$	item
$w \in (\mathcal{V} \cup \mathcal{I})$	token
d	dimension of token embedding
$E^V \in \mathbb{R}^{\|\mathcal{V}\| \times d}$	embeddings of all text tokens (from the LLM)
$E^I \in \mathbb{R}^{\|\mathcal{I}\| \times d}$	embeddings of all item IDs
$E^C \in \mathbb{R}^{\|C\| \times d}$	cluster centroid embeddings of all clusters

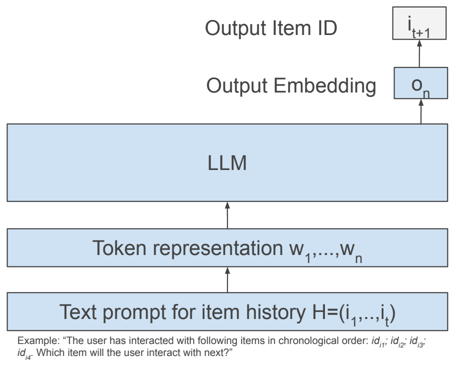
Table 1: Symbols

The challenge of inference latency in LLM-based recommendation: Most existing approaches [6, 8, 19, 21, 37] rely on multiple tokens (multi-token) to represent an item by forcing items through the standard text tokenization process of LLMs. However, a critical challenge hinders the practical deployment of these LLM-based recommenders: the inference latency associated with generating recommendations. They generate textual descriptions or tokenized representations of items, requiring a longer prefill input and multiple sequential calls to the LLM to decode each recommendation. This design fundamentally limits real-time performance, dramatically increasing inference latency and posing a severe bottleneck in real-time scenarios such as e-commerce, dynamic ad placement, and potentially even slate decoding(set of items to present to a user, considering the collective impact of those items).

Why is prefill and decode a bottleneck? LLMs generate text auto-regressively, meaning each new token depends on the previously generated ones. This process has two phases: a parallel *prefill* phase that processes the input prompt, and a sequential *decode* phase where each token is generated one by one, each requiring a full forward pass through the model [1, 25, 28]. Current LLM-based recommenders often use multi-step decoding and longer prefill input, representing each item as a sequence of tokens (see Figure 1a). In real-world scenarios inputs are dominated by potentially hundreds of user history item IDs and the number of items is often O(millions). Further, prefill and decode latency increase with LLM


Output Item ID
i_{t+1}
Algorithm for mapping tokens to items
Output Token
parker
gel
pen
black
w_{n+1}
w_{n+2}
w_{n+3}
w_{n+m}
Output Embedding
o_n
o_{n+1}
o_{n+2}
o_{n+m}
LLM
Token representation $w_1,...,w_n$
Text prompt for item history $H=(i_1,..,i_t)$
Example: "The user has interacted with following items in chronological order: *Scotch Brand Magic Tape, Numerous Applications, Invisible, Cuts Cleanly, 1 x 2592 Inches, 3 Rolls (810-72-3PK); PARKER QUINK Ballpoint Pen Gel Ink Refills, Medium Tip, Blue, 2 Count; Pilot Precise V7 Roller Ball Stick Pen Precision Point Ink.7mm Pack of 12 Black (35346); ...* Which item will the user interact with next?"

(a) Multi-token representation (Baselines): Total latency is $l_{\text{prefill}}(m\cdot|H|+\text{const})+m\cdot l_{\text{decode}}$, where m is the number of tokens to represent a single item.


Output Item ID
i_{t+1}
Output Embedding
o_n
LLM
Token representation $w_1,...,w_n$
Text prompt for item history $H=(i_1,..,i_t)$
Example: "The user has interacted with following items in chronological order: id_{i1}; id_{i2}; id_{i3}; id_{i4}. Which item will the user interact with next?"

(b) Single-token represenation (Ours): Total latency is $l_{\text{prefill}}(|H|+\text{const})+l_{\text{decode}}$.

Figure 1: This figure highlights the key distinction between baselines and our technique's *single-token* representation of items in LLM-based inference. Multi-step methods require multiple iterations of the LLM, each with a latency of l_{decode}, while our single-step approach completes the process with a single l_{decode}. Both methods incur an initial latency of $l_{\text{prefill}}(m\cdot|H|+\text{const})$ for parallel input processing and this latency is also lower as $m=1$ in our case.

size and as we will see later in our experiments, recommendation quality increases with larger LLMs, emphasizing the importance of our efficient method.

Vocabulary disparity and computational cost LLMs and traditional recommender models differ substantially in their output spaces. LLMs utilize tokenizers like WordPiece [5], resulting in a vocabulary ($|\mathcal{V}|$) of only thousands of text tokens. Recommender systems, however, must handle item catalogs ($|\mathcal{I}|$) with potentially millions of discrete item IDs. This disparity creates a computational bottleneck in the LLM's output layer. The standard softmax requires a matrix multiplication with time and space complexity $O((|\mathcal{V}|+|\mathcal{I}|)\cdot d)$ per example, where d is the embedding dimension. While a significant portion of LLM parameters and research effort are dedicated to the attention mechanism (with its quadratic time complexity in input sequence length, as addressed by methods like Reformer [16]), our work tackles the distinct, and for large-scale recommendation settings often dominant bottleneck of the softmax output layer and item input length.

Existing approaches and their limitations Many techniques have been proposed to incorporate LLMs into recommendation systems. One common method is to represent items in the model input and output using their textual metadata, such as titles, descriptions, or categories [6, 8, 19, 21]. Another strategy involves tokenizing item IDs into multiple sub-item tokens, either directly [12] or using hash tables derived from separately trained models [26, 37]. These methods are aimed to reduce the softmax vocabulary size but still rely on *multi-step* decoding, where each individual step within that multi-step process requires a full forward pass of the LLM. Also, the user history is the majority of input prefill and hence prefill length scales proportional to the number of sub-item tokens. This makes the generation of items computationally expensive in latency-sensitive scenarios. While this leverages the natural language processing and generation capabilities of LLMs, it introduces challenges during inference, as accurately mapping the generated tokens back to specific item IDs can add additional complication. To mitigate this, most existing techniques also use specialized approximate beam search (e.g., in CALRec [21]). These limitations highlight the need for a more efficient approach to practically applying LLMs in recommendation systems where low-latency is crucial.

This paper tackles the scalability challenges of using LLMs for recommendation with large item catalogs and long user interaction history. **Our contributions**:

- Identifying the significant prefill-decode bottleneck caused by predominant techniques in LLM-based recommendation literature, and **fundamentally challenging** the prevailing assumption that items must be mapped into the text token space. We identify and demonstrate that representing items via single, direct embeddings offers a superior strategy for LLM-based recommenders.
- Training and inference optimizations, including a Two-Level softmax based on precomputed clusters, that enable efficient training and, crucially, **efficient single-step decoding** for low-latency recommendations using two methods: Inference using Hierarchical Structure and Inference using Approximate Nearest Neighbor (ANN) search.

- A **novel and crucial training paradigm** that leverages rich textual item metadata during training while enabling **highly efficient, ID-only inference**. Our simpler, end-to-end single-phase training achieves higher quality than multi-stage methods predominant in our baselines and in LLM-based recommendation domain.

Figure 1 provides a visual overview of our approach.

2 RELATED WORK

Our work builds upon two main research areas: sequential item prediction and the application of LLMs to RS.

2.1 Sequential Item Prediction

Sequential item prediction aims to forecast a user's next item of interest based on their past interactions. Early approaches, such as GRU4Rec [9], employed Recurrent Neural Networks (RNNs), specifically Gated Recurrent Units (GRUs), to model sequential dependencies in user behavior. Subsequent research explored the use of self-attention mechanisms. For instance, AttRec [34] models user intent within a session using self-attention and incorporates personalization through metric learning for user-item affinity. SASRec [15], utilizes self-attention to capture long-range dependencies in user sequences.

More recently, inspired by the success of masked language modeling in natural language processing, BERT4Rec [29] and S3-Rec [36] adapted Transformer models with masking strategies for sequential recommendation. While effective, these methods primarily rely on behavioral data and may not fully leverage the semantic information present in item descriptions and metadata.

More recent works have started exploring generative retrieval methods [4]. These methods leverage the capabilities of generative models to directly predict the next item, which is the area our work falls under.

2.2 LLMs for Recommendation

The field of recommender systems has witnessed a surge of interest in leveraging the language understanding and generation capabilities of LLMs. Several approaches have been proposed to incorporate user behavior and textual item information into LLMs for recommendation.

One category uses variable-length textual representations for items, leveraging the LLM's natural language processing abilities. Methods like CALRec [21], Recformer [19] and GPT4Rec [20] frame recommendation as a natural language task, using item titles or other metadata. This is beneficial for domains with rich textual item information or limited behavioral data. Another category [12, 26, 30, 32, 37] employs fixed-length tokenized representations learned from behavioral or semantic data. Crucially, these approaches typically rely on multi-token representation of items in input user history and decoding of multiple tokens to generate an item. However, accurately mapping generated tokens back to specific item IDs can be challenging, often requiring adapted beam search (e.g., CALRec [21]). The exact number of steps and the cost of LLM varies, but each step is sequential and incurs an LLM forward pass. In contrast, our work focuses on novel training and inference optimizations to enable efficient prefill-decode inference latency and adapt the LLM's output mechanism to efficiently handle large item catalogs, as detailed in Section 4.

Several existing works tackle training efficiency techniques such as frozen item parameters, LoRA[11]. These efficient training techniques are orthogonal to our approach as our goal is efficient inference.

3 PROBLEM SETTING

Refer to Table 1 for definitions of the symbols used throughout the paper. Let $\mathcal{I}$ denote the set of all items that can be recommended, referred to as the *item catalog*. Each item $i \in \mathcal{I}$ is associated with an item ID and potentially with additional textual metadata (e.g., title, brand).

We consider implicit feedback in the form of past positive interactions between users and items, such as clicks, purchases, or views. For a given user, we represent their interaction history as a chronologically ordered sequence $H = (i_1, i_2, ..., i_t)$, where each $i_j \in \mathcal{I}$ is the item interacted with at time step j.

Our goal is to build a generative recommender system that can predict the next item a user will interact with, given their interaction history. Formally, we aim to model the conditional probability distribution $P(i_{t+1}|H)$, where $i_{t+1} \in \mathcal{I}$ is the next item to be recommended at time step $t + 1$.

3.1 Item Recommendation with an LLM

We frame the sequential item prediction task as an autoregressive language modeling problem. We construct the input sequence for a user history H by interleaving text and item tokens. An example of the LLM prompt is shown below:

> The user has interacted with following items in chronological order id: id$_1$ title: title$_1$, brand: brand$_1$, price: price$_1$, category: category$_1$, id: id$_2$ .., ..., id: id$_t$ .. category: category$_t$. Which item will the user interact with next? id: id$_{t+1}$

The model is trained to efficiently generate the item id$_{t+1}$ following the prompt "Which item will the user interact with next? id:". We leave other specifics of the training task to Section 3.4 and Section 6. Our proposed method is general and can be adapted to other recommendation tasks that can be formulated as text generation problems. For example, the input could be modified to include user review information or the output can be modified to predict a multi-modal representation of item.

3.2 Input Representation of the LLM

We shortly describe how the prompt of a history H with t items is mapped to a tokenized input representation with n tokens $(w_1, \ldots, w_n)$ where each token w_j can be a text token from $\mathcal{V}$ or an item token from $\mathcal{I}$ (see figure 1).

For the textual components (e.g., item metadata, user instructions), we follow the standard LLM approach: We use the pre-trained LLM's tokenizer to convert the text into a sequence of tokens from the vocabulary $\mathcal{V}$, which are then mapped to their corresponding embeddings from the LLM's embedding matrix E^V.

For the item ID components, we store a separate item embedding table of size $|\mathcal{I}| \times k$. To align the item embeddings with the LLM's embedding space, we use a small feedforward neural network (projection head) to project the k-dimensional item embeddings into the d-dimensional LLM embedding space. We refer to the embeddings after the projection as $E^{I} \in \mathbb{R}^{|\mathcal{I}| \times d}$. We interleave the item and text embeddings to create a sequence of input embeddings to the LLM. The item embeddings are trained along with the pre-trained LLM on the final task.

Note that our use of the term "single-token " is an analogy: 1 item → 1 direct embedding vector, contrasting with "multi-token " 1 item → multiple text token embeddings.

3.3 Decoding Items from an LLM

The output of the last layer of the LLM at the very last position n is a d dimensional embedding that we denote with o_n. Based on this output embedding, we want to generate an item id. Section 4 describes our single-step decoding approach.

In contrast, multi-step decoding (see figure 1a) would predict a token w_{n+1} from o_n that is appended to the existing input and the LLM would be invoked on the input $(w_1, \ldots, w_n, w_{n+1})$ to create an embedding o_{n+1}. Multi-step decoding would repeat this process until the tokens $w_{n+1}, w_{n+2}, \ldots$ can be mapped to one of the item IDs in $\mathcal{I}$. Single-step decoding does not need this repeated process and generates the item ID directly from o_n.

Crucially: We map item IDs directly to item embeddings (from a dedicated embedding matrix), completely bypassing the LLM's text tokenizer. Instead of the inbuilt softmax, we use an efficient softmax to predict items directly from their own dedicated embedding vectors (see section 4). The core LLM machinery (transformer blocks) remains unchanged, operating on the embeddings provided.

3.4 Training paradigm for ID-only inference

We introduce a novel training paradigm designed to reconcile the LLM's ability to leverage rich textual item metadata (title, brand, price, category) with the necessity of efficient, ID-only inference in practical recommendation systems where inputs are dominated by potentially hundreds of user history item IDs.

During training, we randomly sample a subset of the item metadata (item ID, title, brand, price, category) for each item in the input sequence and similarly for the label. A crucial 25% of training examples deliberately use only the item ID for both input and output. This specifically trains the model to depend solely on item IDs during inference for evaluation. During inference, each item in the history is represented in the input using a single item embedding (ours) instead of sequences of text token embeddings (baselines), hence our prefill is efficient.

Other LLM recommenders use domain specific pre-training stages or pre-trained item embeddings. With the training paradigm that we outlined above these multi-stage training methods could be removed and provided no improvement over just training the item embeddings from scratch. This is a further simplification over current state-of-the-art LLM recommender methods.

4 EFFICIENT SINGLE-STEP ITEM ID GENERATION

This section introduces approaches for single-step ID generation. First, we describe full softmax which we prescribe if the item catalog is not very large. However, in real-world systems the item catalog is often O(millions) and in this case we propose a Two-Level softmax that can deal with larger catalogs. Due to the hierarchical structure of Two-Level softmax, the runtime is sublinear in the number of items and the model can be trained efficiently. We recommend this approach as opposed to using multi-token representation of items. Our novelty for LLM-based recommendation is the finding that single-token item representation is more efficient for inference and yields better quality than the dominant multi-token approach.

4.1 Full Softmax

In a standard LLM, o_n is used to compute a probability distribution over text tokens x from the vocabulary $\mathcal{V}$ via a softmax function. This approach can be carried over to predicting item IDs as well. The probability of the next token (item ID or text token) reads:

$$P(w|H) = \frac{\exp(\langle o_n, e_w \rangle)}{\sum_{w' \in (\mathcal{I} \cup \mathcal{V})} \exp(\langle o_n, e_{w'} \rangle)}, \quad \text{where } w \in (\mathcal{V} \cup \mathcal{I}) \tag{1}$$

However, directly computing a softmax over the combined vocabulary $\mathcal{V} \cup \mathcal{I}$ can become computationally expensive due to the large size of $\mathcal{I}$. This is because the partition function (denominator in eq. 1) requires a matrix multiplication between o_t and the embedding matrix of the combined vocabulary, which has dimensions $(|\mathcal{V}| + |\mathcal{I}|) \times d$. Therefore, training can become impractical for large item catalogs. It should be mentioned that inference for large item catalogs is less of an issue because ANN algorithms can find efficiently the top scoring items even in very large item catalogs.

To summarize, as long as the item catalog is small enough for practical training, full softmax can be directly applied for single-step decoding. In our evaluation in Section 6, we show that full softmax can achieve high quality item recommendations.

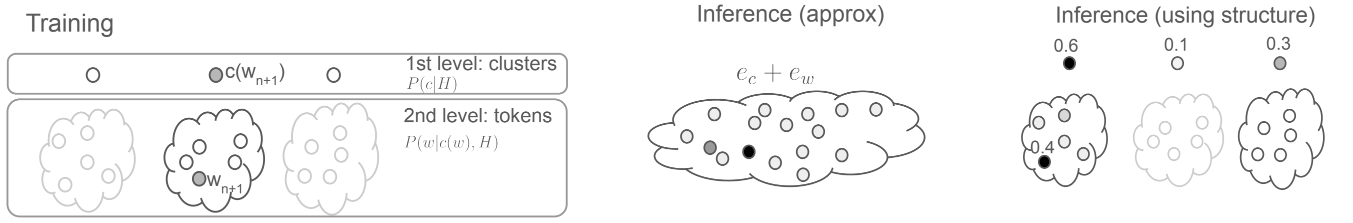
4.2 Two-Level Softmax

To overcome the computational bottleneck of full softmax for large catalogs, we take inspiration from existing research on reducing softmax complexity, such as hierarchical softmax [23]. As shown in Figure 2, instead of directly computing a softmax over $\mathcal{V} \cup \mathcal{I}$, we introduce a novel Two-Level softmax mechanism specifically designed for our problem.

Our Two-Level softmax efficiently approximates the probability distribution over $\mathcal{V} \cup \mathcal{I}$ by introducing a hierarchical structure. We modify the standard LLM softmax to accomplish three key objectives:

- **Incorporate Items into the Probability Distribution:** Extend the LLM's output distribution to encompass both the original text vocabulary $\mathcal{V}$ and the item catalog $\mathcal{I}$.
- **Reduce Computational Cost:** Avoid computing the softmax over the entire item catalog $\mathcal{I}$, which is significantly larger than the text vocabulary $\mathcal{V}$.
- **Enable Dynamic Decoding:** Allow the model to dynamically decide whether to generate a text token or an item ID without requiring external signals or hints.

To achieve this, we define a function $c : \mathcal{I} \cup \mathcal{V} \to C$ that maps each item $i \in \mathcal{I}$ and each text token $v \in \mathcal{V}$ to a cluster $c(i)$ or $c(v)$ respectively, where C is the set of all clusters. We represent each



(a) Training for a token w_{n+1} **(b) Inference using ANN** **(c) Inference using Structure**

Figure 2: This figure illustrates the efficiency gains of our Two-Level softmax during training and inference. For training, only the cluster $c(w_{n+1})$ of the target item w_{n+1} is visited, all other clusters are ignored. At inference time, we leverage either ANN search or the hierarchical structure for efficient retrieval. Here, we can eliminate the second cluster with $P(c_2|H) = 0.1 < 0.6 \cdot 0.4 = 0.24$.

Property	Our Method	Existing Approaches (e.g., multi-token)
Decoding Latency (per item)	$l_{\text{prefill}}(\lvert H\rvert + \text{const}) + l_{\text{decode}}$	$l_{\text{prefill}}(m \cdot \lvert H\rvert + \text{const}) + m \cdot l_{\text{decode}}$
Item Representation Dimension	Flexible (e.g., $k = 512$)	Constrained by LLM's d
Storage Overhead	Moderate (e.g., 0.5 billion parameters for 1M items)	Lower (no separate item embeddings)

Table 2: Comparison of key properties between our method and existing approaches. Here m represents the number of tokens that are used for representing an item (for single-token methods (ours) $m = 1$), and const represents other constant prefill costs, e.g. for the prompt.

cluster c by a learnable centroid embedding $e_c \in \mathbb{R}^d$, where E^C is the matrix of all cluster centroid embeddings.

Our Two-Level softmax then operates as follows:

First Level (Cluster Selection): The first level models a probability distribution over all clusters C. This is achieved by computing the dot product between the decoder output o_n and each cluster centroid embedding e_c, followed by a softmax:

$$P(c|H) = \frac{\exp(\langle o_n, e_c\rangle)}{\sum_{c' \in C} \exp(\langle o_n, e_{c'}\rangle)} \tag{2}$$

Second Level (Item Selection): The second level models the probabilities of individual tokens $w \in \mathcal{I} \cup \mathcal{V}$ conditioned on the token's cluster $c(w)$.

$$P(w|c(w), H) = \frac{\exp(\langle o_n, e_w\rangle)}{\sum_{w': c(w') = c(w)} \exp(\langle o_n, e_{w'}\rangle)} \tag{3}$$

where e_w is the embedding of the target token w.

The overall probability of generating a token w (either an item or a text token) is then given by:

$$P(w|H) = P(c(w)|H) \cdot P(w|c(w), H) \tag{4}$$

In practice, we suggest to partition the item catalog $\mathcal{I}$ into approximately $\sqrt{|\mathcal{I}|}$ clusters using a clustering algorithm (e.g., k-means) on pretrained item embeddings. Each text token $v \in \mathcal{V}$ can be treated as its own cluster (i.e., $c(v) = v$). In this case, for text token clusters, the centroid embedding is simply the token's embedding from E^V.

4.3 Efficient Training and Inference

Next, we will discuss, how our proposed Two-Level softmax enables efficient training and inference.

4.3.1 Training. During training, the target token w_{n+1} and its corresponding cluster $c(w_{n+1})$ are known (see Figure 2a). That means the cost for computing the target's probability (eq. 4) consists of computing the cluster probability which has complexity $O(|C| \cdot d) = O((|\mathcal{V}| + \sqrt{|\mathcal{I}|}) \cdot d)$ and the token probability with cost $O(\sqrt{|\mathcal{I}|} \cdot d)$ in the worst case with our proposed partitioning. The overall training cost of the proposed two level softmax is $O((|\mathcal{V}| + \sqrt{|\mathcal{I}|}) \cdot d)$, which is considerably smaller than the standard softmax complexity of $O((|\mathcal{V}| + |\mathcal{I}|) \cdot d)$.

4.3.2 Inference. For inference, we are interested in returning the most probable tokens for a history H, i.e., we are interested in

$$\operatorname*{argmax}_{w \in \mathcal{I} \cup \mathcal{V}} P(w|H) \tag{5}$$

A trivial computation of $P(w|H)$ for all tokens w from two level softmax has a complexity of $O((|\mathcal{V}| + |\mathcal{I}|) \cdot d)$ because we would visit all clusters.

In the next paragraphs, we will discuss two approaches for fast inference.

Approximate Nearest Neighbor Search. The first approach leverages existing efficient ANN search libraries like ScaNN [7] or FAISS [13]. These libraries are specifically designed for fast similarity search in high-dimensional spaces and are widely used in real-time, large-scale applications, such as search engines and production recommendation systems. They can handle millions of items with extremely low latency (often sub-millisecond), making them suitable for latency-sensitive recommendation scenarios.

Generic ANN algorithms are designed to find items with maximum dot products for a set of embeddings Z and a query vector y by solving $\operatorname{argmax}_{z \in Z} \langle z, y\rangle$. We can recast our two level softmax into such a structure to enable existing ANN algorithms:

$$\begin{aligned}&\operatorname*{argmax}_{w \in \mathcal{I} \cup \mathcal{V}} P(w|H)\\ &= \operatorname*{argmax}_{w \in \mathcal{I} \cup \mathcal{V}} \ln \frac{\exp(\langle o_n, e_{c(w)}\rangle)}{\sum_{c' \in C} \exp(\langle o_n, e_{c'}\rangle)} \frac{\exp(\langle o_n, e_w\rangle)}{\sum_{w' : c(w') = c(w)} \exp(\langle o_n, e_{w'}\rangle)}\end{aligned}$$

The first partition function $\sum_{c' \in C} \exp(\langle o_t, e_{c'}\rangle)$ is independent of any token w and can be ignored when computing the maximum. Thus

$$\begin{aligned}&\operatorname*{argmax}_{w \in \mathcal{I} \cup \mathcal{V}} P(w|H)\\ &= \operatorname*{argmax}_{w \in \mathcal{I} \cup \mathcal{V}} \left[\langle o_n, e_{c(w)} + e_w\rangle - \ln \sum_{w' : c(w') = c(w)} \exp(\langle o_n, e_{w'}\rangle) \right]\\ &\approx \operatorname*{argmax}_{w \in \mathcal{I} \cup \mathcal{V}} \langle o_n, e_{c(w)} + e_w\rangle\end{aligned}$$

Note that the last line is an approximation which is only accurate if $\ln \sum_{w' : c(w') = c} \exp(\langle o_n, e_{w'}\rangle)$ does not differ considerably between clusters. In our experiments(Table 5), this approximation worked well for different clustering algorithms.

To summarize, we can do inference for Two-Level softmax by applying well known ANN algorithms by indexing $e_{c(w)} + e_w, \quad \forall w \in \mathcal{V} \cup \mathcal{I}$ (see Figure 2b).

Inference Using Hierarchical Structure. Another approach for computing the most likely token is to make use of the hierarchical structure [14]. We know that the cluster probability of a token w, $P(c(w)|H)$ is an upper bound for the full probability of the token, i.e.:

$$P(w|H) = P(c(w)|H) \cdot P(w|c(w), H) \leq P(c(w)|H). \quad (6)$$

This fact can be used for pruning whole clusters of items from the maximum search. We can incrementally expand the cluster, c^*, with the highest probability $P(c^*|H)$ among the unvisited clusters and find the token w^* with maximum probability in this cluster. Based on the bound in equation 6, we know that none of the other clusters c' with $P(c'|H) < P(w^*|H)$ can contain a token with higher probability than w^* and thus all items in these clusters can be removed from the search space. See Figure 2c for an illustration.

5 PROPERTIES OF THE MODEL

Our proposed method, using the Two-Level softmax for item generation, offers several advantages in terms of efficiency and flexibility compared to existing approaches. Table 2 summarizes the key properties of our model compared to existing approaches. Next, we will provide more details on inference latency, item embeddings and storage.

5.1 Inference Latency

Inference with an LLM consists of two parts: a prefill phase that loads the prompt and a decode phase that sequentially creates the tokens of the target item. In the decode phase, each generated token requires a full forward pass through the LLM. Let l_{decode} be the latency for decoding a single token and let $l_{\text{prefill}}(n)$ be the prefill latency where n is the number of prefill tokens. The total latency for inference depends on the token representation of the items. If an item is represented by m tokens, there are m decode steps and the prompt for an item history H has a length of $m \cdot |H| + \text{const}$, where const represents the other tokens of the prompt. Thus the total latency for generating an item is $m \cdot l_{\text{decode}} + l_{\text{prefill}}(m \cdot |H| + \text{const})$. m and $|H|$ depend on the dataset and the tokenization. Figure 3 (left) presents a comparison of the number of decoding steps required by our single-step technique ($m = 1$) and multi-step baselines (which decode items as titles or categories) on the (test) datasets from our experiments in Section 6. And figure 3 (middle) presents a comparison of the length of prefill on the (test) datasets from our experiments.

The actual inference latency depends on the specific LLM architecture, hardware and backbone LLM implementation, in particular, on the deployment characteristics l_{decode} and $l_{\text{prefill}}(n)$. For example, with the characteristics of Mistral-7B [1], the overall prefill+decode latency for our method is 35ms + 20ms (55ms) and for our baselines using "title" multi-token encoding is 75ms + 400ms (475ms), improving inference efficiency by 8x. For PALM [25], our method is 48ms + 20ms (68ms) and our baselines using "title" multi-token encoding is 96ms + 580ms (676ms). Figure 3 (right) shows the speedup of our single-step technique over multi-step techniques for various datasets with the Mistral-7B [1] and PaLM [25] characteristics. Finally, we provide an analysis for *any* possible LLM deployment for which $l_{\text{prefill}}(n) = n \cdot l_{\text{prefill}}$, i.e., any LLM that has linear prefill costs. In this case, the speedup is $\frac{m \cdot l_{\text{decode}} + (m \cdot |H| + \text{const}) \cdot l_{\text{prefill}}}{l_{\text{decode}} + (|H| + \text{const}) \cdot l_{\text{prefill}}}$. For any latency characteristics (l_{decode} and l_{prefill}), this function is upper-bounded by m and lower-bounded by $\frac{m \cdot |H| + \text{const}}{|H| + \text{const}}$. Figure 3 (right) shows these upper and lower bounds for the speedup on our datasets.

5.2 Flexible Item Embedding Dimension

Our approach uses a separate embedding table E^I for item representations, which is distinct from the LLM's text token embedding table E^V. This separation offers flexibility in choosing the dimensionality of the item embeddings. While the LLM's text embeddings typically have a fixed dimension (e.g., the d in the PaLM 2 model), we can choose a different dimension k for the item embeddings. In our experiments, we use an item embedding dimension of $k = 512$. This flexibility allows us to optimize the item embeddings for the recommendation task without being constrained by the pre-trained LLM's architecture.

5.3 Storage Considerations

Our method requires storing the item embedding table E^I in addition to the LLM parameters. This does introduce additional storage requirements compared to methods that solely rely on the LLM's internal vocabulary. For example, a model with 1 million items and an item embedding dimension of $k = 500$ would need to store an additional 0.5 billion item parameters. However, the storage overhead is manageable in practice and comparable to that of traditional recommendation models that also store item embeddings. If needed techniques [22, 31] exist to optimize this.

6 EXPERIMENTS

Our experiments investigate the prediction quality of our proposed single-step decoding methods and compare them to other

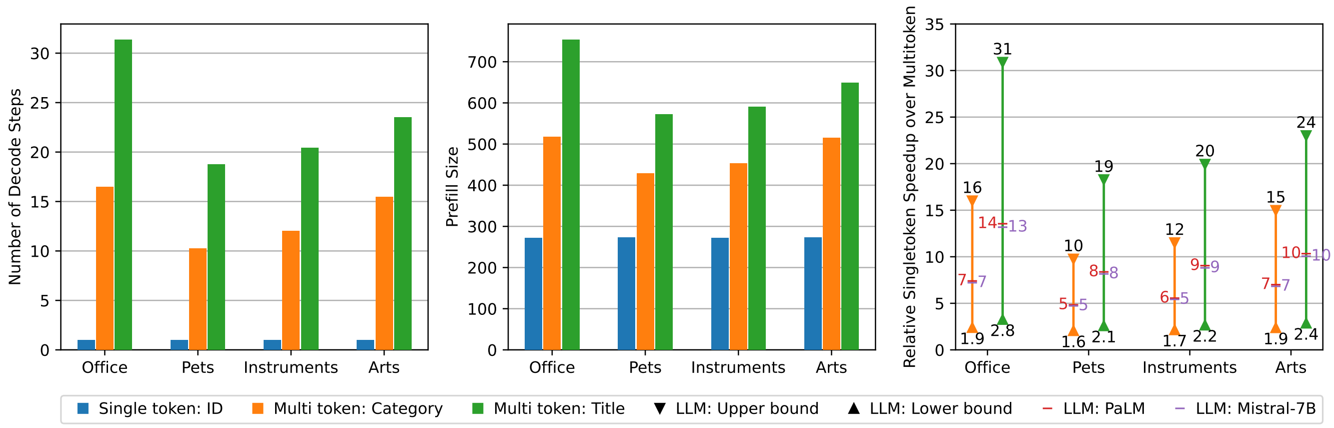


Figure 3: Latency comparison: Left and middle plots show the number of decode steps / number of prefill tokens for single-token (ours) and two multi-token encoders. The actual runtime depends on the prefill and decode characteristics of the LLM inference deployment. The right plot shows the speedup of single-token (ours) over two multi-token encoders (category and title) for four different LLM deployments: PaLM from [25], Mistral-7B from [1], as well as upper and lower bounds. On the real world deployments [1, 25], our speedup is between 5x and 14x over multi-token approaches.

Dataset	# of Users	# of Items	Items/ User	Purchases/ Item
Office	44,736	27,482	7.87	12.81
Pets	43,135	37,712	8.82	10.09
Instruments	25,577	10,599	8.39	20.24
Arts	47,197	22,828	8.72	18.02

Table 3: Statistics of the Amazon Review Datasets

approaches: LLM based recommenders with multi-step decoding and non-LLM recommenders. We also investigate the impact of individual components of our single-step algorithms, specifically the inference algorithm, clustering method and the underlying LLM.

6.1 Setup

6.1.1 Dataset and Preprocessing. We use the widely-used Amazon Review dataset (2018 version) [24], specifically 4-core subsets in Table 3, following the preprocessing steps outlined in CALRec [21]. User interactions are sorted chronologically by timestamp. We adopt the standard leave-one-out evaluation protocol: for each user, the last interaction is used for testing, the second-to-last for validation, and the remaining interactions for training. Like the baselines, our method also utilizes item features: title, category, brand, and price.

The Amazon Review dataset is popular and widely[19, 21, 26] used in the field of LLM-based recommendations due to the rich textual metadata about items present in this dataset.

6.1.2 Model Configuration. We use a batch size of 512, packing multiple examples into each batch and using masking to prevent attention across example boundaries. We use a learning rate of 5e-3, a cosine learning rate decay, and a weight decay of 1e-5. We train for a maximum of 50,000 steps. When using the larger backbone LLM model, PaLM 2 XS, we update weight decay to 1e-2. Labels are replaced with a placeholder in the input to the model and causal mask is used. We use a randomly initialized item embedding of dimension $k = 512$. This embedding is trained along with the LLM for the final task. For our Two-Level softmax, the clusters are pre-computed as a pre-processing step. Recall that 3.4 enabled us to use only item ID representation for test set.

6.1.3 Evaluation Metrics. Following standard practice in sequential recommendation, we evaluate performance using NDCG@K (Normalized Discounted Cumulative Gain), Recall@K and MRR (Mean Reciprocal Rank). We compute these metrics by scoring the full item catalog for each user in the test set and averaging the results.

6.1.4 Baselines. We compare our proposed single-step decoding method against the following baselines:

- **SASRec** [15]: Unidirectional Transformer for sequential recommendation, using item IDs.
- **BERT4Rec** [29]: Bidirectional Transformer with masked language modeling, using item IDs.
- **FDSA** [35]: Two self-attention blocks to address ID sequence and attribute feature sequence, respectively, and fuses their final representations for next-item prediction.
- **S^3-Rec** [36]: Contrastive learning on item IDs and attributes, with pretraining and fine-tuning of encoder-only transformer network.
- **UniSRec** [10]: Encodes item text with BERT (plus adapter) and uses a sequence model, where an item embedding is the sum of its ID and text representations.
- **Recformer** [19]: Longformer with token type and position embeddings, pretrained with MLM and contrastive loss. They fine-tune a pretrained checkpoint and represent items as text sentences.

		Non-LLM Methods					LLM Based Methods				
							Multi-Step Decoding		Single-Step Decoding (Ours)		
Dataset	Metric	BERT4Rec	SASRec	FDSA	S3-Rec	UniSRec	RecFormer	CALRec	Full Softmax	Structure	ANN
Office	Recall@1	0.0403	0.0134	0.0547	0.0291	0.0250	0.0328	0.0761	**0.0839**	0.0814	0.0761
	Recall@10	0.0709	0.1033	0.0982	0.0941	0.1220	0.1063	0.1213	**0.1330**	0.1234	0.1213
	NDCG@10	0.0545	0.0602	0.0749	0.0607	0.0713	0.0687	0.0976	**0.1057**	0.1004	0.0970
	MRR	0.0520	0.0499	0.0710	0.0535	0.0597	0.0603	0.0925	**0.0973**	0.0932	0.0887
Pets	Recall@1	0.0276	0.0093	0.0394	0.0162	0.0202	0.0377	0.0570	**0.0643**	0.0634	0.0625
	Recall@10	0.0499	0.0773	0.0710	0.0647	0.0970	0.0826	0.0937	**0.1104**	0.1094	0.1032
	NDCG@10	0.0376	0.0448	0.0537	0.0392	0.0574	0.0590	0.0736	**0.0839**	0.0819	0.0798
	MRR	0.0365	0.0382	0.0522	0.0349	0.0503	0.0549	0.0696	**0.0760**	0.0741	0.0730
Instruments	Recall@1	0.0494	0.0158	0.0574	0.0202	0.0237	0.0266	**0.0718**	**0.0718**	0.0688	0.0688
	Recall@10	0.0915	0.1130	0.1092	0.1048	0.1255	0.0940	0.1158	**0.1297**	0.1289	0.1225
	NDCG@10	0.0680	0.0623	0.0796	0.0606	0.0709	0.0596	0.0909	**0.0958**	0.0919	0.0913
	MRR	0.0658	0.0528	0.0765	0.0526	0.0613	0.0535	0.0864	**0.0868**	0.0814	0.0818
Arts	Recall@1	0.0347	0.0147	0.0460	0.0216	0.0213	0.0279	**0.0636**	0.0626	0.0600	0.0590
	Recall@10	0.0799	0.1118	0.1061	0.1052	**0.1320**	0.1095	0.1140	0.1229	0.1147	0.1125
	NDCG@10	0.0547	0.0619	0.0726	0.0602	0.0729	0.0662	0.0864	**0.0884**	0.0864	0.0851
	MRR	0.0513	0.0519	0.0680	0.0520	0.0615	0.0582	**0.0815**	0.0780	0.0758	0.0753

Table 4: Comparison of our efficient LLM based recommender to existing non-LLM recommenders and LLM methods. The best results are highlighted in bold, the second best are underlined. Results for baselines are taken from CALRec [21]

		Structure			ANN		
Dataset	Metric	Kmeans	Frequency	Random	Kmeans	Frequency	Random
Office	Recall@1	0.0781	**0.0814**	0.0791	**0.0761**	0.0738	0.0719
	Recall@10	**0.1244**	0.1234	0.1207	0.1213	**0.1250**	0.1201
	NDCG@10	0.0986	**0.1004**	0.0978	**0.0970**	0.0964	0.0939
	MRR	0.0907	**0.0932**	0.0908	**0.0887**	0.0884	0.0858
Pets	Recall@1	**0.0634**	0.0610	0.0615	**0.0625**	0.0566	0.0598
	Recall@10	0.1094	0.1043	**0.1095**	0.1032	0.1017	**0.1074**
	NDCG@10	**0.0819**	0.0804	0.0814	0.0798	0.0764	**0.0800**
	MRR	**0.0741**	0.0731	0.0735	**0.0730**	0.0687	0.0717
Instruments	Recall@1	0.0680	0.0679	**0.0688**	0.0665	0.0667	**0.0688**
	Recall@10	0.1207	0.1204	**0.1289**	0.1170	**0.1231**	0.1225
	NDCG@10	0.0905	0.0903	**0.0919**	0.0880	0.0884	**0.0913**
	MRR	0.0813	0.0811	**0.0814**	0.0792	0.0808	**0.0818**
Arts	Recall@1	**0.0600**	0.0568	0.0542	**0.0590**	0.0565	0.0543
	Recall@10	**0.1147**	0.1024	0.1002	**0.1125**	0.1011	0.1097
	NDCG@10	**0.0864**	0.0842	0.0805	**0.0851**	0.0841	0.0800
	MRR	**0.0758**	0.0745	0.0715	**0.0753**	0.0736	0.0710

Table 5: Ablations on clustering methods (Kmeans, Frequency and Random) for Two-Level softmax when using ANN and Structure inference.

- **CALRec** [21]: Uses multi-step decoding with item metadata, specialized beam search, and contrastive losses. CALRec and our method, use PaLM 2-XXS [2] as the backbone pre-trained LLM for a fair comparison, focusing on the impact of the decoding strategy.

6.2 Results

Table 4 presents the main results of our experiments, comparing our methods Full Softmax (Section 4.1), Two-Level ANN (Section 4.3.2), and Two-Level Structure (Section 4.3.2) against the baselines on the Amazon dataset.

6.2.1 Single-step decoding methods. In Table 4, we compare our single-step single-token techniques with LLM-based multi-step multi-token methods CALRec and Recformer and demonstrate that Full Softmax outperforms them with highest quality. This outperformance suggests that the more direct optimization of item decoding, leads to better performance in this setting. Our proposed method (Two-Level Softmax with clustering) can be used with our

Dataset	Metric	PaLM 2 XXS	PaLM 2 XS
Office	Recall@1	0.0761	**0.0780**
	Recall@10	0.1213	**0.1290**
	NDCG@10	0.0970	**0.1011**
	MRR	0.0887	**0.0903**

Table 6: Ablations on LLM size: PALM 2 XXS 1B parameters and XS 8B parameters with Two-Level softmax ANN inference.

two proposed efficient inference algorithms (Section 4.3.2): Structure and ANN. Structure inference utilizes the hierarchical structure, while ANN inference uses Approximate Nearest Neighbor search to approximate this hierarchical structure. Both inference techniques achieve results comparable to the strong LLM-based baselines, demonstrating that these approaches do not sacrifice recommendation quality for efficiency. Overall, all three approaches significantly outperform the traditional non-LLM recommendation methods (SASRec, BERT4Rec, FDSA, S3-Rec, UniSRec) on the sequential recommendation task. For the datasets used, we observe approximately a 10% improvement in training speed when using Two-Level softmax as opposed to Full softmax. In scenarios where training with Full Softmax is manageable, we recommend Full Softmax for optimal performance. However, when Full Softmax becomes impractical due to computational constraints, we recommend ANN or Structure inference based on the desired level of efficiency.

6.2.2 Ablation of inference algorithm. We ablate the performance of our two proposed inference methods (Section 4.3.2): Structure and ANN in Table 4. ANN represents our simplest setup, approximating the term, $\ln \sum_{w':c(w')=c} \exp(\langle o_n, e_{w'} \rangle)$. While Structure inference generally performs better than ANN inference, the quality of ANN remains competitive to existing multi-step baselines.

6.2.3 Ablation of clustering method. In Table 5, we ablate the choice of clustering method used in our Two-Level softmax. We compare three different clustering techniques: Kmeans, Frequency, and Random. All three techniques are performed as a pre-processing step before the LLM is trained. For Kmeans, we apply K-means clustering on item embeddings generated using an open-source matrix factorization model from [27] and use the resulting clusters to partition the items into approximately $\sqrt{|\mathcal{I}|}$ clusters. For Frequency, we count the frequency of item occurrences in the training set and group items with similar occurrence counts into the same cluster. For Random, we randomly assign items to clusters. For ANN inference, the choice of clustering method appears to have a less significant impact, with all three methods performing relatively similarly. Interestingly, Random clustering performs well in this setting. We hypothesize that this might be because random clustering makes the approximation term in ANN, $\ln \sum_{w':c(w')=c} \exp(\langle o_n, e_{w'} \rangle)$, more uniform across different clusters. Performance of Kmeans clustering is notably better with Structure inference than with ANN based inference across all metrics. For the Office dataset, Structure inference with Frequency-based clustering achieves the best performance.

6.2.4 Ablation of underlying LLM. Although we present most of our results and comparisons with baselines using PaLM2 XXS for a fair comparison to the baseline method CALRec, we also investigate increasing the LLM size from PaLM 2-XXS(1B) to PaLM 2-XS(8B) in Table 6 and this further improves the recommendation quality, suggesting that our approach can benefit from more powerful language models.

6.2.5 Ablation of training paradigm. CALRec uses multiple training stages to first align the LLM with item domain. However we find that with our training paradigm in Section 3.4 we can eliminate these stages and achieve quality. We also experimented with item pretraining stages using [27] to learn collaborative item ID embeddings and [18] to learn semantic embeddings. With Section 3.4 we were able to achieve neutral quality with and without these pretraining stages and hence we do not include these results in the paper.

6.2.6 Summary. The key findings of our study are:

- Improved quality: Our efficient single-stage training approach with Two-Level Softmax, achieves recommendation quality comparable to, and in some cases exceeding, multi-step multi-token LLM-based baselines, and significantly outperforming traditional non-LLM methods.
- Clustering Method Impact: While the clustering method can have some impact, especially with Structure inference, even simple methods like Random clustering can achieve competitive performance, particularly with ANN inference.
- Impact of LLM Size: Increasing the LLM size improves recommendation quality, highlighting the importance of our efficient method.
- Inference latency: The reduction in input prompt length improves prefill latency, and the reduction in decode steps enhances decode latency (Figure 3 for Mistral and PALM).

7 CONCLUSION

This paper addresses the critical latency bottleneck in LLM-based recommender systems which is a crucial limitation for applying LLMs in practical real-time recommendation. Contrary to prevailing methods, we demonstrate the feasibility and superiority of treating items (or any other ids) as first-class citizens of the LLM rather than forcing them through the default LLM text tokenization framework as multi-token. We identify and optimize the critical softmax bottleneck over large item vocab arising from this approach and introduce a novel single-step item decoding enabled by two-level softmax. Combined with our unique, efficient end-to-end training paradigm that eliminates the need for separate pre-training stages, our approach achieves higher quality compared to existing multi-stage and multi-step approaches on the Amazon product recommendation benchmarks while significantly reducing inference latency. Our key contributions are (1) a novel perspective on prefill-decode latency of existing approaches, (2) presenting a highly effective single-token approach, and (3) offering a simplified, single-stage training algorithm that achieves state-of-the-art quality. This work demonstrates the potential for efficiently bridging the gap between powerful LLM capabilities and the practical constraints of large-scale, real-time recommendation, potentially inspiring a new direction for future research in LLM-based recommendation. Future research directions include exploring the integration of external embeddings, use of semantic understanding for clustering and use of multi-modal item representations in our training paradigm.

8 GENAI USAGE DISCLOSURE

GenAI was not used to create or edit this manuscript. All sections and code have been created and verified by authors.

REFERENCES

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 117–134. https://dl.acm.org/doi/10.5555/3691938.3691945

[2] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403* (2023). arXiv:2305.10403 [cs.CL] http://arxiv.org/abs/2305.10403

[3] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. Association for Computing Machinery, New York, NY, USA, 191–198. doi:10.1145/2959100.2959190

[4] Yashar Deldjoo, Zhankui He, Julian McAuley, Anton Korikov, Scott Sanner, Arnau Ramisa, René Vidal, Maheswaran Sathiamoorthy, Atoosa Kasirzadeh, and Silvia Milano. 2024. A Review of Modern Recommender Systems Using Generative Models (Gen-RecSys). In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24)*. Association for Computing Machinery, New York, NY, USA, 6448–6458. doi:10.1145/3637528.3671474

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. doi:10.48550/ARXIV.1810.04805

[6] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. 2022. Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt &amp; Predict Paradigm (P5). doi:10.48550/ARXIV.2203.13366

[7] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. doi:10.48550/arXiv.1908.10396 arXiv:1908.10396.

[8] Jesse Harte, Wouter Zorgdrager, Panos Louridas, Asterios Katsifodimos, Dietmar Jannach, and Marios Fragkoulis. 2023. Leveraging Large Language Models for Sequential Recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems (RecSys '23)*. Association for Computing Machinery, New York, NY, USA, 1096–1102. doi:10.1145/3604915.3610639

[9] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015*. arXiv:1511.06939 [cs.IR] https://arxiv.org/abs/1511.06939

[10] 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 (KDD '22)*. Association for Computing Machinery, New York, NY, USA, 585–593. doi:10.1145/3534678.3539381

[11] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. doi:10.48550/arXiv.2106.09685 arXiv:2106.09685.

[12] Wenyue Hua, Shuyuan Xu, Yingqiang Ge, and Yongfeng Zhang. 2023. How to Index Item IDs for Recommendation Foundation Models. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*. ACM, Beijing China, 195–204. doi:10.1145/3624918.3625339

[13] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. doi:10.48550/arXiv.1702.08734 arXiv:1702.08734.

[14] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, Mirella Lapata, Phil Blunsom, and Alexander Koller (Eds.). Association for Computational Linguistics, Valencia, Spain, 427–431. https://aclanthology.org/E17-2068/

[15] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. 197–206. doi:10.1109/ICDM.2018.00035 ISSN: 2374-8486.

[16] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The Efficient Transformer. doi:10.48550/ARXIV.2001.04451

[17] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (Aug. 2009), 30–37. doi:10.1109/MC.2009.263

[18] Jinhyuk Lee, Zhuyun Dai, Xiaoqi Ren, Blair Chen, Daniel Cer, Jeremy R. Cole, Kai Hui, Michael Boratko, Rajvi Kapadia, Wen Ding, Yi Luan, Sai Meher Karthik Duddu, Gustavo Hernandez Abrego, Weiqiang Shi, Nithi Gupta, Aditya Kusupati, Prateek Jain, Siddhartha Reddy Jonnalagadda, Ming-Wei Chang, and Iftekhar Naim. 2024. Gecko: Versatile Text Embeddings Distilled from Large Language Models. doi:10.48550/arXiv.2403.20327 arXiv:2403.20327.

[19] 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. doi:10.1145/3580305.3599519

[20] Jinming Li, Wentao Zhang, Tiantian Wang, Guanglei Xiong, Alan Lu, and Gérard G. Medioni. 2023. GPT4Rec: A Generative Framework for Personalized Recommendation and User Interests Interpretation. *ArXiv* abs/2304.03879 (2023). https://api.semanticscholar.org/CorpusID:258048480

[21] Yaoyiran Li, Xiang Zhai, Moustafa Alzantot, Keyi Yu, Ivan Vulić, Anna Korhonen, and Mohamed Hammad. 2024. CALRec: Contrastive Alignment of Generative LLMs for Sequential Recommendation. In *18th ACM Conference on Recommender Systems*. ACM, Bari Italy, 422–432. doi:10.1145/3640457.3688121

[22] Xiao Luo, Haixin Wang, Daqing Wu, Chong Chen, Minghua Deng, Jianqiang Huang, and Xian-Sheng Hua. 2023. A Survey on Deep Hashing Methods. *ACM Transactions on Knowledge Discovery from Data* 17, 1 (Feb. 2023), 1–50. doi:10.1145/3532624

[23] Frederic Morin and Yoshua Bengio. 2005. Hierarchical Probabilistic Neural Network Language Model. In *International Workshop on Artificial Intelligence and Statistics*. PMLR, 246–252. https://proceedings.mlr.press/r5/morin05a.html

[24] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 188–197. doi:10.18653/v1/D19-1018

[25] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2022. Efficiently Scaling Transformer Inference. (2022). doi:10.48550/ARXIV.2211.05102

[26] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Keshavan, Trung Vu, Lukasz Heidt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. 2023. Recommender systems with generative retrieval. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, 10299–10315.

[27] Steffen Rendle, Walid Krichene, Li Zhang, and Yehuda Koren. 2022. Revisiting the Performance of iALS on Item Recommendation Benchmarks. In *Proceedings of the 16th ACM Conference on Recommender Systems (RecSys '22)*. Association for Computing Machinery, New York, NY, USA, 427–435. doi:10.1145/3523227.3548486

[28] Noam Shazeer. 2019. Fast Transformer Decoding: One Write-Head is All You Need. doi:10.48550/ARXIV.1911.02150

[29] 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 (CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 1441–1450. doi:10.1145/3357384.3357895

[30] Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. 2022. Transformer memory as a differentiable search index. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, 21831–21843.

[31] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, Montreal Quebec Canada, 1113–1120. doi:10.1145/1553374.1553516

[32] Shuyuan Xu, Wenyue Hua, and Yongfeng Zhang. 2024. OpenP5: An Open-Source Platform for Developing, Training, and Evaluating LLM-based Recommender Systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 386–394. doi:10.1145/3626772.3657883

[33] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. 2019. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys '19)*. Association for Computing Machinery, New York, NY, USA, 269–277. doi:10.1145/3298689.3346996

[34] Shusheng Zhang, Lina Yao, Xiang Xu, Chengan Wang, and Xu Wang. 2019. Next item recommendation with self-attentive metric learning. In *Proceedings of The 30th International Joint Conference on Artificial Intelligence*. 4316–4322. doi:10.24963/ijcai.2019/599

[35] 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 28th International Joint Conference on Artificial Intelligence (IJCAI'19)*. AAAI Press, Macao, China, 4320–4326.

[36] 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 (CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 1893–1902. doi:10.1145/3340531.3411954

[37] Jing Zhu, Mingxuan Ju, Yozen Liu, Danai Koutra, Neil Shah, and Tong Zhao. 2025. Beyond Unimodal Boundaries: Generative Recommendation with Multimodal Semantics. doi:10.48550/ARXIV.2503.23333