Decoding in Latent Spaces for Efficient Inference in LLM-based Recommendation

Anonymous ACL submission

Abstract

Fine-tuning large language models (LLMs) for recommendation in a generative manner has delivered promising results, but encounters significant inference overhead due to autoregressive decoding in the language space. This work explores bypassing language-space decoding by directly matching candidate items with the LLM's internal thought representations in the latent space, eliminating the time-consuming autoregressive process to reduce computational costs. Towards this, we introduce *Light Latent-space Decoding* (*L2D*), an effective and efficient latent-space decoding method. *L2D* represents user-preferred items by using the hidden states of test sequences reflecting the LLM's internal thought, and obtains candidate item representations from the hidden states of training sequences labeled with the corresponding candidate items. It then matches the two types of representations to decode items, achieving latent-space decoding. In this way, it enables efficient decoding without altering the LLM's generative tuning paradigm, thereby preserving performance. Extensive empirical results demonstrate that *L2D* is more than 10x faster than language-space decoding while maintaining or enhancing performance.

1 Introduction

Large language models (LLMs) have been widely fine-tuned on recommendation data in textual format to directly generate the next item of user interest based on historical interactions. This training paradigm aligns well with the generative nature of LLMs, enabling them to develop sophisticated user understanding and interest-mining capabilities. When deploying these fine-tuned LLMs for personalized recommendations (LLM4Rec), a key research challenge is how to effectively decode the items that LLM truly "thinks" or prefers internally.

Current LLM4Rec methods primarily rely on the LLM's internal decoding ability (*i.e.,* language-space decoding), which typically operates the LLM

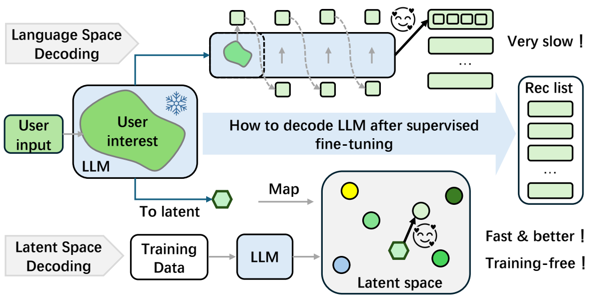


Figure 1: Illustration of language-space vs. latent-space decoding: Latent-space decoding bypasses the slow language-space decoding and instead achieve decoding via directly matching candidate item with the LLM internal 'thought' items in the latent space. It preserves the generative tuning paradigm to keep performance while enabling efficient decoding.

in an autoregressive token-by-token manner to output the item representations in the language space (e.g., title). In the autoregressive decoding, one token generation would wait for all preceding tokens' generations, incurring substantial time costs. Worse yet, each recommendation request from users typically requires generating a list of items (Lin et al., 2024a), linearly scaling the cost regarding the list size. While grounding techniques (Bao et al., 2023) can reduce these costs by mapping each generated item to multiple actual items, they may lead to performance degradation. For example, in our findings, mapping only one generated item for top-10 recommendations would result in a performance drop of fifty percent compared to generating 10 items (*c.f.,* Figure 3).

This work considers bypassing this language-space decoding to enhance decoding efficiency. Training a recommendation head to decode the next item by directly predicting its ID provides a straightforward way to avoid language-space decoding, improving decoding efficiency. However, this training objective deviates from the language model's original goal of next-token prediction, which may hinder effective utilization of its pretrained knowledge. This raises a new question: can we bypass language-space decoding while still preserving the powerful generative training characteristics of the LLM's recommendation tuning?

By re-examining LLM decoding, we can interpret the tuned LLM as having already thought a recommendation target within its internal latent space in response to a user query, with autoregressive decoding merely serving to activate it. If we can find the appropriate representation for the item in this latent space—and represent all candidate items similarly—then efficient decoding can be achieved through representation matching within the same space. Meanwhile, since this approach only extracts the LLM's internal "thoughts" learned during LLM's generative training phase, it preserves the original generative training objective during recommendation tuning.

To obtain representations of the items internally considered by the LLM, we propose using the last hidden states from the final LLM layers corresponding to a given test sample, as the state primarily drives item generation. For candidate items, since the training set already provides matching pairs of hidden states and the item serving as ground-truth items, we can aggregate all the hidden states associated with the item to form an effective representation in the latent space, without incurring additional training cost. This approach is based on the idea that each paired hidden state captures a distinct feature aspect of the corresponding ground-truth item, allowing us to combine them into a meaningful and comprehensive target representation.

To this end, we propose *Light Latent-space Decoding ($L2D$)*, a simple yet efficient method for latent-space decoding. After finishing generative training, we store the training samples' hidden states and their labels (*i.e.*, ground-truth items) in a memory module and create each item's representation by aggregating its associated hidden states in the memory. Then we decode items to recommend by finding the item whose representation is most similar to the test sample's hidden state using L2 distance. Regarding the aggregation to form item representation, $L2D$ offers two strategies: 1) global aggregation, which averages all associated hidden states for an item, and 2) local aggregation, which uses only the top-M most similar samples from the memory based on the test sample's hidden state. The global strategy provides a comprehensive representation, while the local strategy focuses on aspects most relevant to the test sample.

The main contributions of this work are summarized as follows:

- We propose bypassing language-space decoding for efficient recommendation inference while preserving the powerful generative training characteristics of LLM for recommendation.
- We introduce $L2D$, a simple yet effective method that performs latent-space decoding by leveraging the hidden states of test and training sequences, to capture the LLM's internal thought.
- Extensive experiments demonstrate that applying $L2D$ to existing LLM-based recommendation methods reduces inference latency by at least 10 times compared to language-space decoding while maintaining or enhancing performance.

2 LLM-based Generative Recommender

Let $\mathcal{D}$ represent the user-item interaction data. The j-th sample in $\mathcal{D}$ is denoted as (s_j, v_j), where s_j represents a user's interaction history, and v_j is the interacted item for the sample. Notably, both s_j and v_j are in textual form. To train an LLM-based generative recommender, we convert each sample (s_j, v_j) into instruction data, using a fixed prompt template such as "*A user has interacted with the following items: <s_j>; which item would the user like next?*", with v_j as the ground-truth model output. Then, the instruction data $\{(prompt(s_j), v_j)\}_{\mathcal{D}}$ can be utilized to fine-tune the LLM.

During inference, given a user's interaction history s to generate the next item, the LLM first encodes the prompt into hidden states, formally:

$$h = LLM_{last}(prompt(s)), \tag{1}$$

where h denotes the **last** hidden state of the input $prompt(s)$ at the final layer, and $LLM_{last}(\cdot)$ represents the function that extracts the hidden state from the last layer of the LLM. In the language-space decoding method, h is further mapped to the LLM's output layer to generate the first item token, which is then added to the input, and the process repeats to generate a full item. In contrast, we explore decoding items from the hidden state h.

3 Latent-Space Decoding

In this section, we introduce our *Light Latent-space Decoding ($L2D$)* framework, starting with presenting the overview and followed by a detailed description of its key components.

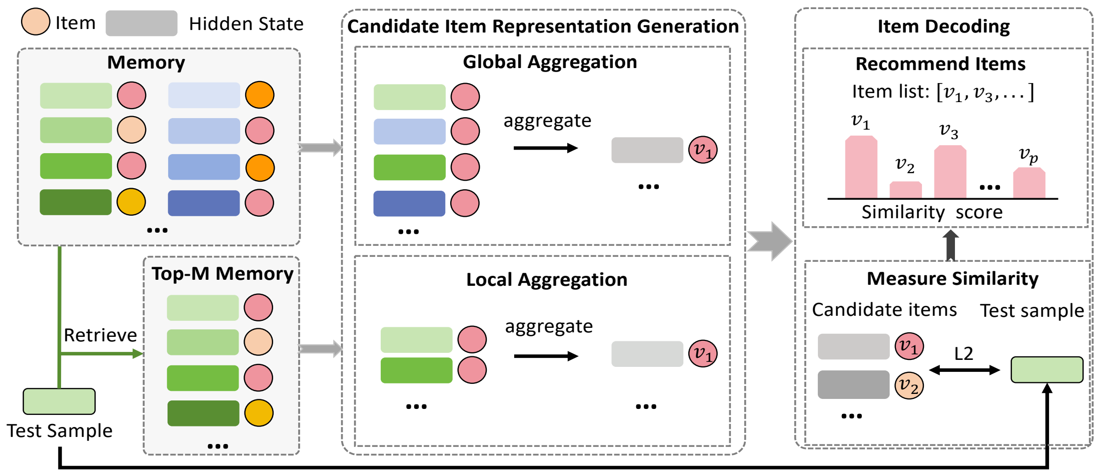


Figure 2: The overview framework of our proposed $L2D$. The left part illustrates the memory set that stores (hidden state, ground-truth item) pairs. The middle part illustrates how $L2D$ generates candidate item representations via global aggregation (averaging all associated hidden states) or local aggregation (using the top-M relevant samples to the test sample). The right part depicts the item decoding phase by measuring the similarity between the test sample's hidden state and candidate item representations.

3.1 Overview

The main idea of this work is to bypass time-intensive language-space decoding while still preserving the powerful generative training characteristics of the LLM's recommendation tuning. To achieve this, we propose $L2D$, a light latent-space decoding framework, which directly utilizes the hidden states from the LLM to construct latent space for decoding, where the LLM is trained with recommendation data in a generative manner. Figure 2 illustrates the overall $L2D$ process, which consists of three steps:

1) **Memory Construction**: Stores (hidden state, ground-truth item) pairs from training samples in a memory module, preparing for candidate item representation generation.
2) **Candidate Item Representation Generation**: Produces representations for each item by aggregating its associated hidden states stored in memory.
3) **Item Decoding**: Matches the hidden state of a test sample with the candidate item representations to determine the output.

The first step can be pre-computed, ensuring no impact on inference latency, while the last two steps operate independently of LLM, minimizing latency. We provide detailed explanations below.

3.2 Memory Construction

$L2D$ begins by constructing a memory set that stores the (hidden state, ground-truth item) pairs from the training samples. Specifically, for the j-th training sample (s_j, v_j), we compute its last hidden state at the final layer using Equation (1) as $h_j = LLM_{last}(prompt(s_j))$ and store the pair (h_j, v_j) in a memory set $\mathcal{M}$. Repeating this process for all samples in the training set, $L2D$ constructs the final memory $\mathcal{M}$, formally,

$$\mathcal{M} = \{(h_j, v_j) \mid j = 1, \ldots, N\}, \qquad (2)$$

where N denotes the total number of training samples. The hidden state in each pair reflects a specific feature aspect of the corresponding item, meaning the process effectively captures one facet of the LLM's original understanding of the item in the same latent space. The memory set is then used to generate representations of candidate items in this latent space.

3.3 Candidate Item Representation Generation

After constructing the memory, $L2D$ leverages the stored (hidden state, ground-truth item) pairs to generate representations of candidate items in the latent space. For each candidate item, it aggregates the associated hidden states—those paired with the item as the ground-truth item—to create the item's representation. In particular, $L2D$ offers two aggregating strategies: 1) global aggregation, which averages all associated hidden states for each candidate item, and 2) local aggregation, which uses only the top-M most similar samples in the memory based on the test sample's hidden state. The global strategy provides comprehensive representation of candidate items, while the local strategy makes the representation more relevant to the test samples. We will first elaborate on the two strategies, followed by a comparison.

• **Global Aggregation**. To aggregate the hidden states stored in the memory $\mathcal{M}$ for creating representation of candidate items, a straightforward approach is to directly average all hidden states associated with the same item. The global aggregation follows this strategy. Specifically, we first group hidden states in memory by items and then average the hidden states within each group to form the corresponding item's representation. Formally, for an item v, its representation $\bar{h}_v$ is computed as follows:

$$\bar{h}_v = \frac{1}{|\mathcal{M}(v)|} \sum_{h_j \in \mathcal{M}(v)} h_j, \tag{3}$$

where $\mathcal{M}(v)$ denotes the set of all hidden states associated with item v, defined as

$$\mathcal{M}(v) = \{h_j \mid (h_j, v_j) \in \mathcal{M}, v_j = v\}.$$

The size of $\mathcal{M}(v)$ is denoted by $|\mathcal{M}(v)|$.

•**Local Aggregation**. The LLM's understanding of a candidate item may encompass multiple feature aspects, and the global aggregation method combines all aspects to form a comprehensive item representation. However, during the inference stage, not all feature aspects are relevant for each test sample; only the aspects related to the test sample are important. This suggests that mixing all feature aspects in one representation may introduce interference. With this in mind, we propose local aggregation, which leverages only the top-M samples from memory that are most relevant to the test sample's hidden state for item representation generation.

Specifically, for a test sample with s_t, we first filter a subset of the memory based on the hidden state h_t of test sample, denoted as $\mathcal{M}_t$. Formally,

$$\begin{aligned} \mathcal{M}_t = \{&(h_j, v_j) \mid (h_j, v_j) \in \mathcal{M}, \\ &S(h_t, h_j) \text{ is in the top-}M \text{ largest}\}, \end{aligned} \tag{4}$$

where $S(h_t, h_j) = \frac{1}{\|h_t - h_j\|_2}$ measures the similarity between the stored hidden state h_j and the test sample's hidden state h_t. Then, a process similar to global aggregation is applied to $\mathcal{M}_t$ to obtain the candidate item representation. Given a candidate item v, the representation is formulated as follows:

$$\bar{h}_v^t = \frac{1}{|\mathcal{M}_t(v)|} \sum_{h_j \in \mathcal{M}_t(v)} h_j, \tag{5}$$

where $|\mathcal{M}_t(v)|$ denotes the size of $\mathcal{M}_t(v)$, and $\mathcal{M}_t(v)$ is the subset of $\mathcal{M}_t$ containing items with v as the ground-truth, defined as

$$\mathcal{M}_t(v) = \{h_j \mid (h_j, v_j) \in \mathcal{M}_t, \, v_j = v\}.$$

Global vs. Local Aggregation: Compared to global aggregation, local aggregation can better focus on test sample-specific aspects, potentially improving subsequent matching performance. However, it may struggle more with sparse items due to an increased lack of associated hidden states. Additionally, unlike the representation obtained through global aggregation, which is uniform for all test samples, the representation derived from local aggregation is tailored to each test sample. For quantitative comparative experiments and further analysis of their suitable scenarios for the global and local aggregation, please refer to Section 4.3.1. Meanwhile, the computational cost of both aggregation methods remains negligible, as our approach fundamentally bypasses the time-consuming autoregressive decoding of the LLM. The method requires only a single forward pass during LLM inference, with all subsequent operations being performed through efficient vector-level computations.

3.4 Item Decoding

After generating the candidate item representations, $L2D$ could efficiently decode items in the latent space during inference by measuring the similarity between the test sample's hidden state and the representations of the candidate items. Specifically, for a given test sample with hidden state h_t and a candidate item v, we denote the candidate item's representation as h_v, which is defined as:

$$h_v = \begin{cases} \bar{h}_v & \textit{in Eq. } (3) \quad \text{if global aggregation,} \\ \bar{h}_v^t & \textit{in Eq. } (5) \quad \text{if local aggregation.} \end{cases} \tag{6}$$

Then, we compute the similarity score between h_t and h_v using the L2 distance as: $S(h_t, h_v) = \frac{1}{\|h_t - h_v\|_2}$. Once the similarity scores for all candidate items are computed, the top-K items with the highest similarity scores to the test sample are selected to form the final recommendation list. **We refer to $L2D$ with global aggregation as $L2D$-G, and $L2D$ with local aggregation as $L2D$-L.**

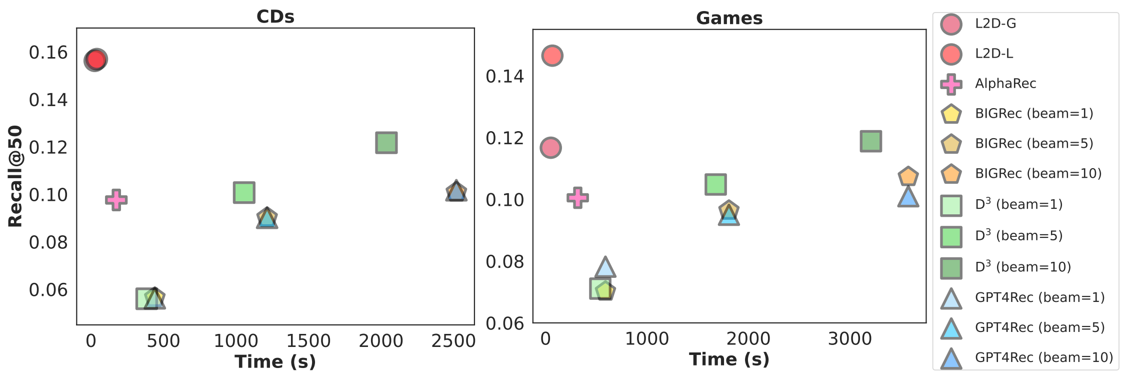


Figure 3: The Recall@50 performance and the inference overhead of LLM-based recommenders on two datasets.

Table 1: Overall performance comparison. **Results with beam size 1 are reported for methods using beam search for fair comparison, with results for other beam sizes in Figure 3**. The best results are in bold.

		CDs						Games					
	Model	R@20	R@50	R@100	N@20	N@50	N@100	R@20	R@50	R@100	N@20	N@50	N@100
Traditional	SASRec	0.1015	0.1271	0.1522	0.0602	0.0653	0.0693	0.0684	0.1117	0.1564	0.0332	0.0417	0.0490
	GRU4Rec	0.0707	0.1027	0.1347	0.0376	0.0439	0.0491	0.0664	0.1099	0.1601	0.0302	0.0387	0.0468
LLM embedding	AlphaRec	0.0651	0.0976	0.1353	0.0300	0.0364	0.0425	0.0619	0.1005	0.1392	0.0295	0.0371	0.0434
LLM Generative (beam = 1)	GPT4Rec	0.0513	0.0562	0.0652	0.0433	0.0443	0.0458	0.0508	0.0782	0.1064	0.0293	0.0347	0.0392
	BIGRec	0.0506	0.0565	0.0621	0.0435	0.0446	0.0456	0.0476	0.0702	0.1007	0.0284	0.0328	0.0378
	D^3	0.0507	0.0560	0.0623	0.0436	0.0447	0.0457	0.0478	0.0711	0.1004	0.0284	0.0330	0.0376
Ours	$L2D$-G	0.1144	0.1562	**0.1996**	**0.0710**	**0.0792**	**0.0862**	0.0646	0.1167	0.1794	0.0295	0.0397	0.0499
	$L2D$-L	**0.1158**	**0.1569**	0.1992	0.0667	0.0745	0.0813	**0.0879**	**0.1465**	**0.2072**	**0.0399**	**0.0511**	**0.0596**

4 Experiments

In this section, we conduct experiments on two widely-used real-world datasets to demonstrate the effectiveness of our $L2D$ framework in balancing performance and inference overhead. We will showcase it by following research questions: **RQ1**: How does the performance and inference overhead of our $L2D$ compare to LLM-based baselines? **RQ2**: In which scenarios are global and local aggregation most suitable, respectively? **RQ3**: How does the hyperparameter M affect $L2D$-L? **RQ4**: How does the performance of $L2D$ compare to the ID-based classifier? **RQ5**: What is the spatial efficiency of $L2D$?

4.1 Experimental Settings

Datasets. We evaluated our approach using two representative Amazon Product Review datasets [1]: Amazon CDs (CDs) and Amazon Games (Games). These datasets consist of user review data collected from Amazon between 1996 and 2018. Each review was treated as a user-item interaction. Following (Bao et al., 2024), we truncated datasets by timestamp to ensure manageable scale, filtered out users/items with fewer than five interactions, and limited user interaction sequences to a maximum length of 10. All interactions was chronologically ordered and splited into training/validation/test sets (8:1:1 ratio). Dataset statistics are detailed in Appendix A.4.

Compared Methods. In this work, to demonstrate the superiority of our proposed method from the perspective of balancing performance and inference overhead in LLM-based recommendation, we primarily selected some of the most commonly used LLM-based models in the current literature. **For LLM-based embedding**, we included **AlphaRec** (Sheng et al., 2025) as a baseline. **For LLM-based generative recommendation**, we included the following methods: **BIGRec** (Bao et al., 2023), **GPT4Rec** (Zhang et al., 2024a), **D^3** (Bao et al., 2024). Additionally, we included **non-LLM baselines** (**SASRec** (Kang and McAuley, 2018) and **GRU4Rec** (Hidasi et al., 2016)) for comprehensive comparison. For a comprehensive description of these baselines, please refer to the Appendix A.6.

For all generative-based methods, we used beam search to generate multiple items and then map them to real items in the dataset. The implementation details of beam search for recommendation list generation can be found in Appendix A.2.

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

Evaluation metrics. To evaluate the top-K recommendation performance, we employed two widely adopted metrics: Recall@K and NDCG@K (Bao et al., 2024; Zheng et al., 2024). All evaluations follow a full-ranking evaluation protocol (Bao et al., 2023), with K ∈ {20, 50, 100}. In the following, if space is limited, we will abbreviate Recall@K and NDCG@K as R@K and N@K, respectively.

Other detailed settings are in the appendix A.3

4.2 Main Results (RQ1)

To verify the effectiveness of our $L2D$, we present the performance and inference cost of our method compared to the baseline in Figure 3. Furthermore, we illustrate the performance of our method at different K values in Table 1. From the figure and the table, we can find:

- When evaluating the trade-off between performance and inference cost for all methods, we observe from Figure 3 that points closer to the top-left corner indicate better performance at lower costs. Our proposed $L2D$ method is the closest to the top-left corner on both datasets, indicating that $L2D$ achieves excellent performance while maintaining low inference cost, showcasing the effectiveness of direct decoding of items in latent space. Even when compared to the previously most efficient LLM-based method, AlphaRec, which uses LLM as embeddings, $L2D$ reduces the cost by at least a factor of five and gets a better performance, further demonstrating the remarkable potential of $L2D$ in deployment.

- When comparing the performance of baseline methods under different beam sizes, we observe that the performance of generative-based methods improves approximately linearly as the beam size and inference cost increase. Among these, D^3 shows greater scalability (with a larger growth rate). It would not be surprising if these methods could surpass $L2D$ in performance by investing more in inference (e.g., increasing the beam size to 50), but this could lead to nearly a hundredfold increase in cost, which is not feasible in most real-world scenarios. Furthermore, our experiments utilize Llama 3.2-1B as the backbone, which is a relatively small-scale language model. The deployment costs would be even higher with larger language models.

- Furthermore, Table 1 shows the comprehensive performance evaluation. To rigorously assess the real-world performance of LLM-based models during deployment, inference time costs must be carefully accounted for. Consequently, we adopt the results with the smallest beam size (beam = 1) as the fair comparison baselines for our $L2D$ method, despite the fact that its inference time cost remains substantial (cf. Figure 3). Additionally, we included traditional (i.e. non-LLM) baselines for comprehensive comparison. $L2D$ outperforms all baselines across all metrics. We attribute this improvement to the method's ability to effectively preserve the powerful generative training characteristics of the LLM's recommendation tuning, thus leveraging the LLM's capabilities acquired during the SFT phase.

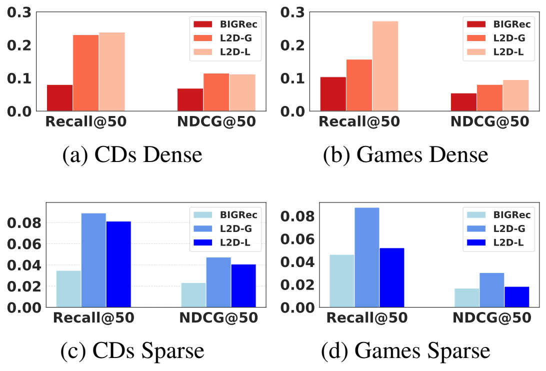


Figure 4: The performance of BIGRec, $L2D$-G, and $L2D$-L on sparse and dense scenarios.

4.3 Analysis

In this section, we present a comprehensive analysis of $L2D$. We first discuss its application scenarios in both sparse and dense recommendation settings. Next, we conduct ablation studies to evaluate: 1)The impact of the key hyperparameter M in $L2D$-L, 2)The performance comparison between $L2D$ and ID-based classifiers. Finally, we analyze the space efficiency of the $L2D$ framework.

4.3.1 Sparse and Dense Scenario (RQ2)

The discrepancy between $L2D$-G and $L2D$-L results in distinct application scenarios for these two approaches. To analyze this, we divided the test set into sparse and dense categories based on item frequency in the training set. Figure 4 shows the overall performance of the two strategies in these scenarios. We observed the following: (1) **Dense scenarios:** $L2D$-L achieves the best performance due to the availability of numerous hidden states for each item, allowing it to create a more personalized candidate item representation and eliminate

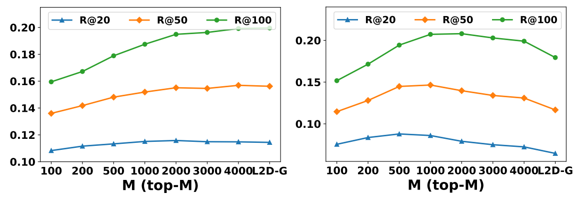
Table 2: $L2D$ vs. ID-based classfier: The overall performance (a) and performance on sparse scenarios (b).

(a)

CDs	R@20	R@50	R@100	N@20	N@50	N@100
Classfier	0.1087	0.1490	0.1886	0.0634	0.0714	0.0778
$L2D$	**0.1158**	**0.1569**	**0.1996**	**0.0710**	**0.0792**	**0.0862**
Games	R@20	R@50	R@100	N@20	N@50	N@100
Classfier	**0.0896**	**0.1557**	**0.2205**	0.0374	0.0505	**0.0610**
$L2D$	0.0879	0.1465	0.2072	**0.0399**	**0.0511**	0.0596

(b)

CDs (Sparse)	R@20	R@50	R@100	N@20	N@50	N@100
Classfier	0.0491	0.0671	0.0835	0.0271	0.0307	0.0333
$L2D$	**0.0682**	**0.0889**	**0.1125**	**0.0432**	**0.0473**	**0.0511**
Games (Sparse)	R@20	R@50	R@100	N@20	N@50	N@100
Classfier	0.0242	0.0446	0.0706	0.0095	0.0135	0.0177
$L2D$	**0.0508**	**0.0874**	**0.1305**	**0.0230**	**0.0303**	**0.0372**



(a) Recall on CDs (b) Recall on Games

Figure 5: The impact of M on the Recall metric for $L2D$-L, where M denotes the hyperparameter that determines the number of hidden states in local aggregation. Note that $L2D$-L becomes equivalent to $L2D$-G when M reaches its maximum value.

Table 3: The performance of $L2D$ when storing only 30% of the training samples. The best results are in bold.

CDs	R@20	R@50	R@100	N@20	N@50	N@100
SASRec	**0.1015**	0.1271	0.1522	0.0602	0.0653	0.0693
GRU4Rec	0.0707	0.1027	0.1347	0.0376	0.0439	0.0491
AlphaRec	0.0651	0.0976	0.1353	0.0300	0.0364	0.0425
GPT4Rec (beam=1)	0.0513	0.0562	0.0652	0.0433	0.0443	0.0458
BIGRec (beam=1)	0.0506	0.0565	0.0621	0.0435	0.0446	0.0456
D^3 (beam=1)	0.0507	0.0560	0.0623	0.0436	0.0447	0.0457
$L2D$-G (30%)	0.1012	**0.1391**	**0.1807**	**0.0621**	**0.0696**	**0.0763**
$L2D$-L (30%)	0.0991	0.1344	0.1678	0.0579	0.0649	0.0703
Games	R@20	R@50	R@100	N@20	N@50	N@100
SASRec	0.0684	0.1117	0.1564	0.0332	0.0417	0.0490
GRU4Rec	0.0664	0.1099	0.1601	0.0302	0.0387	0.0468
AlphaRec	0.0619	0.1005	0.1392	0.0295	0.0371	0.0434
GPT4Rec (beam=1)	0.0508	0.0782	0.1064	0.0293	0.0347	0.0392
BIGRec (beam=1)	0.0476	0.0702	0.1007	0.0284	0.0328	0.0378
D^3 (beam=1)	0.0478	0.0711	0.1004	0.0284	0.033	0.0376
$L2D$-G (30%)	**0.0759**	**0.1314**	**0.1973**	**0.0341**	**0.0450**	**0.0556**
$L2D$-L (30%)	0.0682	0.1214	0.1795	0.0305	0.0407	0.0500

irrelevant information. (2) **Sparse scenarios:** the interactions are limited, which means that even the top similar hidden states in Memory module may not accurately represent user preferences, potentially leading to biased results and performance drops. In contrast, $L2D$-G, which aggregates preferences globally, offers a more balanced outcome.

4.3.2 Hyper-parameter M on $L2D$-L (RQ3)

We illustrate the impact of M in Figure 5, where only the results for Recall are reported. The results for NDCG can be found in Appendix A.5.

Specifically, on the CDs dataset, the Global Aggregation method in $L2D$-G outperforms the Local Aggregation method in $L2D$-L. In contrast, on the Games dataset, we observe that performance peaks as M increases, but further increasing M leads to a decline in performance. We attribute this phenomenon to the varying demands for focusing on the test sample's feature aspects in different recommendation scenarios. The Games dataset may require a stronger emphasis on detailed feature aspects compared to the CDs dataset.

4.3.3 L2D vs. ID-based classfier (RQ4)

Training a recommendation head to decode the next item by directly predicting its ID provides a straightforward way to avoid language-space decoding, improving decoding efficiency. However, this training objective deviates from the LLM's original goal of next-token prediction, which may hinder effective utilization of its pretrained knowledge. Moreover, this approach incurs additional training overhead and may perform poorly on sparse items, as it requires learning in the LLM's high-dimensional representation space. In contrast, our method can bypass language-space decoding while still preserving the powerful generative training characteristics of the LLM's recommendation tuning, performing well on sparse items recommendation scenarios.

To verify this, we trained a classifier head using the LLM's hidden states with careful hyperparameter tuning. The results, summarized in Table 2, report both (a) overall performance and (b) performance on sparse recommendation scenario. For overall performance (a), on the CDs dataset, our method consistently outperforms the baseline, achieving an average relative improvement of 11.2%. On the Games dataset, our method performs better for smaller K in NDCG and remains comparable for larger K in both metrics. Regarding performance on sparse recommendation scenario (b), our method consistently achieves significantly better results.

4.3.4 Spatial complexity (RQ5)

In this subsection, we analyze the spatial complexity of the proposed $L2D$ framework. Although the pre-stored hidden states of training samples in $L2D$ introduce additional space requirements, these costs remain manageable since this scale of storage is feasible even for personal devices. For instance, if each sample corresponds to a 1024-dimensional hidden state (float16), storing hidden states for 10^9 training samples requires approximately 1024×2 bytes × $10^9 \approx 2$ TB.

Storage costs can be further optimized by selectively retaining only a subset of training samples, such as through reservoir sampling technique (Valkanas et al., 2024), where new data is added over time while older data is removed. Our experiments show that even when storing only 30% of the orignal training data, our method still outperforms baselines (where only the Recall@20 metric on the CDs dataset is competitive with SASRec). The results are shown in Table 3.

5 Related Work

• **LLM-based recommendation.** We discuss three paradigms of LLM-based recommenders (Wu et al., 2024). (1) **LLM-Embedding-Based Recommenders** use embeddings from LLMs in traditional systems to capture user preferences (Yuan et al., 2023; Xi et al., 2024a). While effective in language tasks, these embeddings require fine-tuning for optimal performance. (2) **LLM-Based Discriminative Recommenders** directly predict user-item interactions by optimizing the recommendation task with the LLM's loss function (Zhang et al., 2023; Li et al., 2023b; Zhang et al., 2024b). Although it dispenses with intermediate embeddings, it requires evaluating each item individually, reducing efficiency compared to traditional models. (3) **LLM-Based Generative Recommenders** generate natural language recommendations without predefined items, offering innovative potential (Bao et al., 2023, 2024; Zheng et al., 2024). However, autoregressive decoding introduces significant inference overhead. Inspired by these paradigms, we propose a novel LLM-based recommender that balances performance and overhead, addressing existing challenges to enhance quality and efficiency.

Notably, some existing (large) language model (LM)-based approaches (Sheng et al., 2025), such as RecFormer (Li et al., 2023a), can be viewed as representing candidate items in latent spaces and then matching them with the user input sequence encoded by the LM. However, they indeed modify the output layer of the LMs, with the effectiveness of their matching process tied to the training process. As a result, they fail to achieve plug-and-play integration into existing advanced LLM-based recommenders. In contrast, our method is decoupled from the training process, making it plug-and-play. Additionally, these methods' training objectives deviate from the large language model's original goal of next-token prediction, which may hinder effective utilization of its pretrained knowledge. A detailed discussion is provided in Appendix A.1.

• **Inference Acceleration for LLM-based Recommendation.** With the widespread application of LLMs, an increasing number of studies have focused on accelerating LLM inference. In particular, in the field of LLM-based recommender systems, models need to recommend products to a large number of users within a short time frame, which highlights the necessity of considering methods to accelerate LLM inference in this domain. Speculative Decoding (SD) (Leviathan et al., 2023), a significant acceleration technique in the NLP field, has been applied to recommender systems, such as DARE (Xi et al., 2024b) and AtSpeed (Lin et al., 2024b). However, these methods still rely on acceleration decoding within the language space. In contrast, our method takes a step further by exploring how to implement efficient decoding for recommendation in the latent space of LLMs, while maintaining a simple and easy-to-implement overall framework that avoids complex designs.

6 Conclusion

In this study, we emphasized that fine-tuning LLMs for recommendations in a generative manner is highly promising but encounters significant inference overhead due to the original autoregressive decoding strategy. To address this challenge, we proposed the $L2D$, which bypasses time-consuming autoregressive decoding in the language space and directly decodes items in LLM's latent space. The $L2D$ preserves the generative tuning paradigm to keep performance while enabling efficient decoding. Our results highlighted the potential of latent space decoding as a fundamental advancement in LLM-based recommender systems, and extensive results demonstrated the superiority of $L2D$.

Limitations

This paper has the following limitations: 1) Although the $L2D$ framework we introduced significantly reduces inference latency, the memory, which is pre-constructed, still incurs additional time overhead during its pre-construction process. This motivates us to explore more efficient memory construction methods in future work. 2) Our approach, while capable of processing items with at least one interaction without requiring additional training (unlike traditional methods such as SASRec/GRU4Rec and some LLM-based methods such as D3/ID-based classfier that need retraining), still shares the same fundamental limitation as conventional methods: it cannot handle fully cold-start items with zero interaction history. In the future, we plan to address this issue by using the interpolation technique or incorporating auxiliary models. 3) We have not considered the problem of memory updating. As user interaction data gradually accumulates over time, how to effectively use this new data to update the memory in $L2D$ to achieve higher decoding performance presents a promising direction. We intend to explore this issue in future research.

Ethical Considerations

In this paper, we present $L2D$, designed to balance the performance and inference overhead for generative LLMRec. Our method decode item in latent space of LLM which doesn't raise ethical concerns. Moreover, the data we use are publicly available and don't include sensitive details. However, recommendations involve user behavioral data, which might raise privacy concerns, which can be addressed through introducing the mechanism of user consent. Additionally, using LLMs may have potential negative societal biases. We argue for a thorough risk assessment and alert users to the potential risks associated with model deployment.

For the large language model use, we utilize ChatGPT to help polish the writing at the sentence level.

References

Keqin Bao, Jizhi Zhang, Wenjie Wang, Yang Zhang, Zhengyi Yang, Yancheng Luo, Chong Chen, Fuli Feng, and Qi Tian. 2023. A bi-step grounding paradigm for large language models in recommendation systems. *arXiv preprint arXiv:2308.08434*.

Keqin Bao, Jizhi Zhang, Yang Zhang, Xinyue Huo, Chong Chen, and Fuli Feng. 2024. Decoding matters: Addressing amplification bias and homogeneity issue in recommendations for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 10540–10552.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *ICLR (Poster)*.

Wang-Cheng Kang and Julian J. McAuley. 2018. Self-attentive sequential recommendation. In *ICDM*, pages 197–206. IEEE Computer Society.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Jiacheng Li, Ming Wang, Jin Li, Jinmiao Fu, Xin Shen, Jingbo Shang, and Julian J. McAuley. 2023a. Text is all you need: Learning language representations for sequential recommendation. In *KDD*, pages 1258–1267. ACM.

Xinhang Li, Chong Chen, Xiangyu Zhao, Yong Zhang, and Chunxiao Xing. 2023b. E4srec: An elegant effective efficient extensible solution of large language models for sequential recommendation. *CoRR*, abs/2312.02443.

Xinyu Lin, Chaoqun Yang, Wenjie Wang, Yongqi Li, Cunxiao Du, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. 2024a. Efficient inference for large language model-based generative recommendation. *arXiv preprint arXiv:2410.05165*.

Xinyu Lin, Chaoqun Yang, Wenjie Wang, Yongqi Li, Cunxiao Du, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. 2024b. Efficient inference for large language model-based generative recommendation. *CoRR*, abs/2410.05165.

Leheng Sheng, An Zhang, Yi Zhang, Yuxin Chen, Xiang Wang, and Tat-Seng Chua. 2025. Language representations can be what recommenders need: Findings and potentials. In *ICLR*. OpenReview.net.

Antonios Valkanas, Yuening Wang, Yingxue Zhang, and Mark Coates. 2024. Personalized negative reservoir for incremental learning in recommender systems. *CoRR*, abs/2403.03993.

Likang Wu, Zhi Zheng, Zhaopeng Qiu, Hao Wang, Hongchao Gu, Tingjia Shen, Chuan Qin, Chen Zhu, Hengshu Zhu, Qi Liu, et al. 2024. A survey on large language models for recommendation. *World Wide Web*, 27(5):60.

Yunjia Xi, Weiwen Liu, Jianghao Lin, Xiaoling Cai, Hong Zhu, Jieming Zhu, Bo Chen, Ruiming Tang, Weinan Zhang, and Yong Yu. 2024a. Towards open-world recommendation with knowledge augmentation from large language models. In *RecSys*, pages 12–22. ACM.

Yunjia Xi, Hangyu Wang, Bo Chen, Jianghao Lin, Menghui Zhu, Weiwen Liu, Ruiming Tang, Weinan Zhang, and Yong Yu. 2024b. A decoding acceleration framework for industrial deployable llm-based recommender systems. *CoRR*, abs/2408.05676.

Zheng Yuan, Fajie Yuan, Yu Song, Youhua Li, Junchen Fu, Fei Yang, Yunzhu Pan, and Yongxin Ni. 2023. Where to go next for recommender systems? ID- vs. modality-based recommender models revisited. In *SIGIR*, pages 2639–2649. ACM.

Peiyan Zhang, Yuchen Yan, Xi Zhang, Liying Kang, Chaozhuo Li, Feiran Huang, Senzhang Wang, and Sunghun Kim. 2024a. Gpt4rec: Graph prompt tuning for streaming recommendation. In *SIGIR*, pages 1774–1784. ACM.

Yang Zhang, Keqin Bao, Ming Yan, Wenjie Wang, Fuli Feng, and Xiangnan He. 2024b. Text-like encoding of collaborative information in large language models for recommendation. In *ACL (1)*, pages 9181–9191. Association for Computational Linguistics.

Yang Zhang, Fuli Feng, Jizhi Zhang, Keqin Bao, Qifan Wang, and Xiangnan He. 2023. Collm: Integrating collaborative embeddings into large language models for recommendation. *CoRR*, abs/2310.19488.

Bowen Zheng, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, Ming Chen, and Ji-Rong Wen. 2024. Adapting large language models by integrating collaborative semantics for recommendation. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 1435–1448. IEEE.

A Appendix

A.1 Contribution Positioning

Some existing (large) language model (LM)-based approaches (Sheng et al., 2025), such as RecFormer (Li et al., 2023a), can be seen as representing candidate items in latent spaces and matching them with the user input sequence encoded by the LM. This makes them somewhat similar to our approach. However, there are inherent differences between these methods and ours. First, our method does not alter the generative training process (next-token prediction); it only modifies the decoding process without requiring additional tuning. In contrast, in these existing approaches, the matching process is entangled with the training phase. As a result, they fail to achieve plug-and-play integration into existing advanced LLM-based recommenders. Also, the training objective of these methods deviates from the language model's original goal of next-token prediction, which may hinder effective utilization of its pretrained knowledge.

Secondly, even when focusing solely on the matching process, there are differences in how the sequence representations and candidate item representations are constructed, as well as in the learning processes involved. Our approach introduces the following innovations:

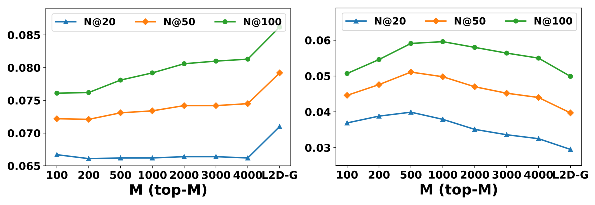
- History Representation: Our representation is derived from the hidden state embedding at the next-token prediction position, which serves as a "generative state" inherently encoding information for generating subsequent tokens. In contrast, the existing methods do not leverage such a generative state of LLMs.
- Item Representation: We construct item representations by aggregating the "generative states" of training samples where the item appears as the target. This fundamentally differs from existing works, which require an item-based forward encoding approach.
- Learning: Our history and item representations exist in the same space and do not require additional tuning. In contrast, existing methods necessitate a separate training process to align these representations for matching.

A.2 Beam-search for Recommendation

For all generative-based methods, we use beam search to generate multiple items and then match them to real items. Specifically, we first obtain the semantic representation of each generated item and compute their matching scores based on their semantic similarity with all candidate items. This results in a ranking matrix with dimensions beam_number × candidate_item_number, where each row represents the ranking list of a beam-generated item. Finally, we flatten the matrix column by column into a single vector and retain the top K unique items as the recommendation results.

Table 4: The statistics of datasets.

Dataset	#User	#Item	#Train	#Valid	#Test
CDs	21,347	14,239	148,685	18,586	18,587
Games	34,089	11,037	201,613	25,202	25,203



(a) CDs NDCG (b) Games NDCG

Figure 6: The impact of M on NDCG metric in the $L2D$-L. M is a hyperparameter that determines the number of hidden states in local aggregation. Note that $L2D$-L equals $L2D$-G when M reaches its maximum length.

A.3 Implementation details

Our LLM-based recommendation models were built on Llama3.2-1B (Dubey et al., 2024) as the backbone architecture. During the instruction tuning phase, we adopted the AdamW optimizer along with a cosine learning rate scheduler, using a batch size of 64 and exploring learning rates in {1e-3, 1e-4, 5e-5}. Other hyperparameters align with the default configurations from the D^3 paper (Bao et al., 2024). All experiments are conducted on NVIDIA A100 GPUs.

A.4 Dataset Statistics

In this subsection, we supplement the statistical information of the datasets used in our experiments. Please refer to Table 4

A.5 Impact of M on NDCG for L2D-L

In this subsection, we demonstrate the impact of parameter M on the NDCG metric in the $L2D$-L method. As shown in Figure 6, the phenomenon observed in the NDCG metric is consistent with the Recall metric in the paper, further strengthening our argument.

A.6 Compared Methods

In this work, to demonstrate the superiority of our proposed method from the perspective of balancing performance and inference overhead in LLM-based recommendation, we primarily selected some of the most commonly used LLM-based models in the current literature. **For LLM-based embedding**, we included **AlphaRec** (Sheng et al., 2025) as a baseline. This method uses LLM embeddings for recommendations by applying a collaborative filtering model to utilize language representations. **For LLM-based generative recommendation**, we included the following methods: (1)**BIGRec** (Bao et al., 2023): A generative LLM-based recommender that predicts the next item via historical interactions, mapping generated items to the dataset by L2 distance matching on semantic embeddings. (2) **GPT4Rec** (Zhang et al., 2024a): Similar to BIGRec but employs BM25 for item mapping. (3) **D^3** (Bao et al., 2024): An improved variant of BIGRec that mitigates decoding bias by eliminating length normalization. Additionally, we included **non-LLM baselines** (**SASRec** (Kang and McAuley, 2018) and **GRU4Rec** (Hidasi et al., 2016)) for comprehensive comparison.