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 decoding items in the latent space, eliminating the time-intensive autoregressive process to reduce costs. Moreover, given that the hidden states of input sequences in the latent space have already encapsulated user preference information, latent-space decoding also has the potential to preserve performance. Towards this, we introduce *Light Latent-space Decoding (L2D)*, an effective and efficient latent-space decoding method. L2D uses the hidden states of test sequences to represent user-preferred items, and it derives 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. Empirical results demonstrate that L2D is more than 10x faster than language-space decoding while maintaining or enhancing performance.

1 Introduction

Inspired by the powerful capabilities of Large Language Models (LLMs), great efforts (Wu et al., 2024a; Li et al., 2024) have been made to adapt them to recommendations, giving rise to the LLM-based recommendation paradigm. Unlike traditional recommendations, this paradigm reimagines recommendation by shifting it to the "language space", where the task and related entities (i.e., users and items) are described in natural language (Zhang et al., 2023a; Bao et al., 2023b). The popular approaches are typically performed in a generative manner — fine-tuning LLMs to directly generate (or say, decode) items in the language space (Bao et al., 2023a; Ji et al., 2024). This manner aligns well with the generative nature of LLMs, effectively harnessing the LLMs' power to achieve promising results (Bao et al., 2024).

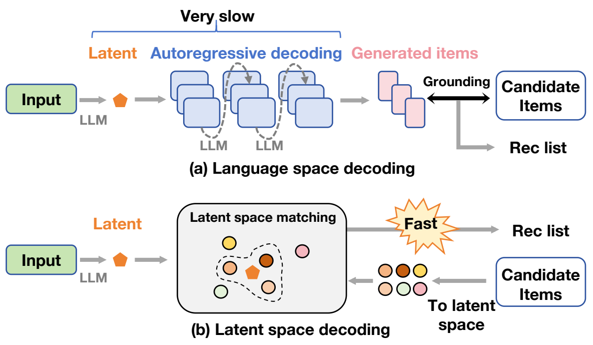


Figure 1: Illustration of language-space decoding and latent-space decoding: Latent-space decoding eliminates the autoregressive decoding process in language-space decoding and directly performs item matching in the latent space for efficient item decoding.

Decoding recommended items in the language space poses a significant challenge to inference efficiency. When decoding an item, the LLM must generate its representation (e.g., title) autoregressively, with each token depending on the previous one (Xia et al., 2024), incurring substantial time costs. Worse, each recommendation request typically requires generating a list of items (Lin et al., 2024a), causing the inference cost to scale linearly with the list size. While grounding techniques (Bao et al., 2023a) can reduce 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.,* Table 1).

Given these limitations, we explore the possibility of bypassing language-space decoding by analyzing the decoding process. Examining the detailed process shown in Figure 1 (a), the LLM first encodes the input into hidden states, which are then mapped to the output layer to produce the first token. Subsequently, this token is appended to the input, and the process repeats iteratively to generate other tokens. The initial hidden states capture the essential information that determines the generated item. This suggests that the hidden states can serve as latent representations of user-preferred items without losing key information. Then, as shown in Figure 1 (b), representing candidate items in the same latent space, we can directly decode recommended items in the latent space through matching, bypassing the time-intensive autoregressive process. As such, we can reduce the time costs while maintaining the performance.

This work investigates directly decoding items in the latent space to achieve efficient inference. The key is to represent candidate items within the same latent space as the hidden states, enabling straightforward item decoding through matching. While learning item representations seems an intuitive solution, it introduces additional costs and struggles to maintain quality for sparse items in the high-dimensional latent space. Instead, we note that the training set already provides matching pairs (hidden state, ground-truth item). Each paired hidden state reflects a feature aspect of the corresponding ground-truth item, allowing us to aggregate these paired hidden states into an effective representation of the item in the latent space without incurring extra training costs.

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 directly decoding in latent space rather than language space for LLM-based recommenders, to better balance performance and inference cost.
- We introduce $L2D$, an effective and efficient latent-space decoding method that recommends items by comparing their representations, derived from hidden states of training sequences, to the test sample's hidden state using L2 distance.
- 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 $\{(prompt(s_j), v_j)\}_j$ can be utilized to fine-tune the LLM, learning the recommendation tasks by generating the next item's textual representation.

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.

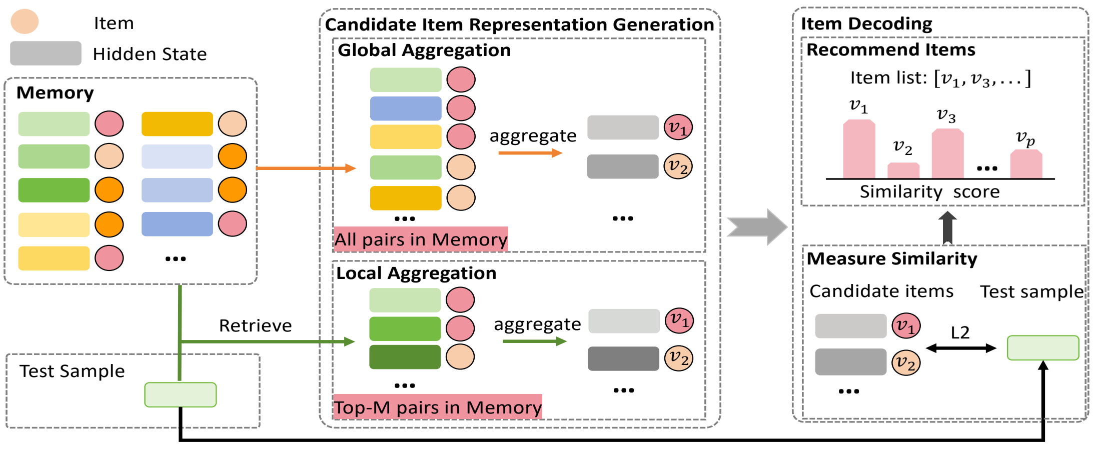


Figure 2: The overview framework of our proposed *L2D*. The left part illustrates the memory set that stores (hidden state, ground-truth item) pairs. The middle part illustrates how *L2D* generates candidate item representations via global aggregation (averaging all associated hidden states) or local 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 language-space decoding by directly operating in the latent space, thereby eliminating the time-intensive autoregressive process to reduce costs fundamentally. To achieve this, we propose *L2D*, which utilizes the hidden states from the LLM. It constructs candidate item representations in the latent space and matches them with the hidden states of the test sample to decode items. 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. Such pairs are derived from real user-item interaction data, meaning the hidden state reflects specific aspects of the corresponding item and can be further utilized for item representation generation. 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,

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

where N denotes the total number of training samples. The memory set is then utilized to generate item representations.

3.3 Candidate Item Representation Generation

After constructing the memory, *L2D* leverages the stored (hidden state, ground-truth item) pairs to generate item representations in the latent space. For each 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 item, and 2) local aggregation, which uses only the top-M most similar samples in the memory based on the test user's hidden state. The global strategy provides a comprehensive representation, while the local strategy focuses on aspects most relevant to the test sample. 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 item representations, 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**. An 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 aspects 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. Despite this, the computational cost remains relatively low, as only a small subset of the memory is considered for each representation creation and no LLM processing is involved.

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.**

Discussion. Our method only requires a single forward propagation for LLM inference, while other operations rely on vector-level operations. For both L2D-G and L2D-L, the total vector computation cost could be kept far lower than LLM inference, thus reducing overall latency. For more details, refer to Experiments and Appendix A.4. On the other hand, Storing vectors, especially for L2D-L, which retains intermediate vectors from training samples, can increase space cost. However, space efficiency is generally less critical in recommendation, and strategies like sampling can help mitigate the cost. More details are in Appendix A.3.

4 Experiments

In this section, we conduct experiments on two 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 do the performance and overhead results of our $L2D$ compare to the baselines? **RQ2**: What is the impact of each component of $L2D$? **RQ3**: What are suitable scenarios for the global and local aggregation, respectively? **RQ4**: How does $L2D$ influence the quality of recommendation list?

4.1 Experimental Settings

Datasets. We conduct experiments using two representative datasets from Amazon Product Reviews[1]: Amazon CDs (CDs) and Amazon Games (Games). These datasets consist of user review data collected from Amazon between 1996 and 2018. We regard the review as an interaction. Following (Bao et al., 2024), we truncate the datasets based on timestamps to maintain a manageable size, filter out users and items with fewer than five interactions, and set the maximum length of user interaction sequences to 10. We chronologically order the interactions and split them into training, validation, and test sets in an 8:1:1 ratio. Detailed statistics of the datasets are provided in Appendix A.7.

Compared Methods. In this work, to demonstrate the superiority of our proposed method from the perspective of balancing performance and inference overhead compared to LLM-based generative recommendation systems, we primarily selected some of the most commonly used LLM-based models in the current literature. (1)**AlphaRec** (Sheng et al., 2024): This method uses LLM embeddings for recommendations by applying a collaborative filtering model to utilize language representations. (2)**BIGRec** (Bao et al., 2023a): This is a generative recommender system based on LLMs. It predicts the next item a user might interact with by using their past interactions, mapping the generated items to the existing dataset by comparing L2 distances to semantic embeddings. (3) **GPT4Rec** (Zhang et al., 2024a): Similar to BIGRec, it uses the BM25 method for mapping items generated by LLMs to those already in the dataset. (4) D^3 (Bao et al., 2024): Building on BIGRec, this method fixes a potential bias in BIGRec's process by removing length normalization during decoding. It also includes another collaborative model, which was omitted in our implementation for fair comparison.

For all generative-based methods, we use beam search to generate multiple items and then match them to real items. The details on how beam search decodes into a recommendation list can be found in the appendix A.6.

Evaluation metrics. To assess the top-K recommendation performance of the model, we employ two widely recognized metrics: Recall@K and NDCG@K (Bao et al., 2024; Zheng et al., 2024). In our study, all evaluations are conducted following a full-ranking protocol (Bao et al., 2023a), with K generally set to 20, 50, and 100. In the following, if space is limited, we will abbreviate Recall@K and NDCG@K as R@K and N@K, respectively.

Implementation details. For our LLM-based recommendation models, we employ Llama3.2-1B (Dubey et al., 2024) as the foundational architecture. During the instruction tuning phase, we use the AdamW optimizer along with a cosine learning rate scheduler, set a batch size of 64, and modify the learning rate within the range of [1e-3, 1e-4, 5e-5]. Other configurations generally adhere to those outlined in the D^3 paper. All experiments are executed on an NVIDIA A100 GPU.

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.

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

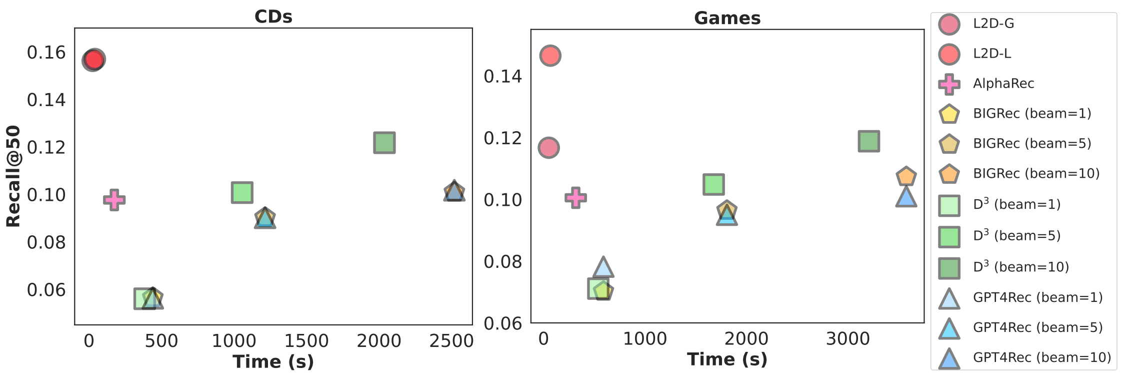


Figure 3: The Recall@50 performance and the overhead of LLM-based recommender system on two datasets.

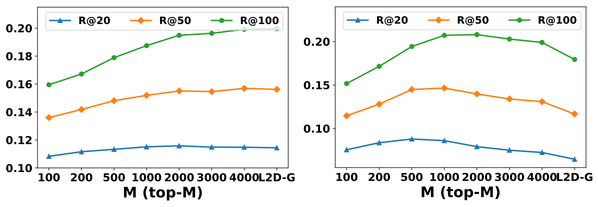
Table 1: Overall performance comparison on the CDs and Games. Results with beam size 1 are reported for methods using beam search for fair comparison, with results for other beam sizes in Figure 3. The best results are in bold.

	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
AlphaRec	0.0651	0.0976	0.1353	0.030	0.0364	0.0425	0.0619	0.1005	0.1392	0.0295	0.0371	0.0434
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
$L2D$-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**

- 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, as shown in Table 1, $L2D$ outperforms all baselines across all metrics. We attribute this improvement to the method's ability to effectively decode multiple historical interests in the latent space of LLM, which significantly increases the likelihood of meeting users' current interests and demonstrates the robustness and scalability of our method.

Notably, we include non-Amazon datasets to evaluate our method. The results show that it achieves the lowest inference latency while maintaining strong performance across most metrics. See Appendix A.1 for details.



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

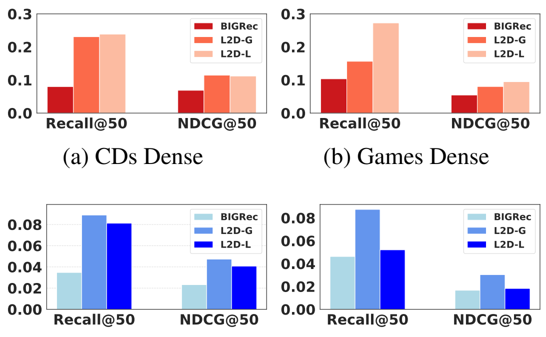
Figure 4: The impact of M on the Recall metric for $L2D$-L, where M denotes the hyperparameter that determines the number of hidden states in local aggregation. Note that $L2D$-L becomes equivalent to $L2D$-G when M reaches its maximum value.

4.2.1 Ablation study (RQ2)

To validate the effectiveness of each component of $L2D$, we conducted the following experiments:

4.3 Analysis

In this section, we conduct a thorough analysis of $L2D$. We begin with an ablation study on each component, followed by a discussion on the application scenarios for our two decoding strategies in both sparse and dense recommendation contexts. Finally, we examine how our design influences the diversity and popularity of recommendation lists.



(a) CDs Dense (b) Games Dense

(c) CDs Sparse (d) Games Sparse

Figure 5: The performance of BIGRec, $L2D$-G, and $L2D$-L on sparse and dense scenarios.

The impact of hyper-parameter M on $L2D$-L. We illustrate the impact of M in Figure 4, where only the results for Recall are reported. The results for NDCG can be found in Appendix A.8. 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.

The impact of aggregation on $L2D$-L. To gain a comprehensive understanding of how $L2D$-L operates, we conducted an experiment to validate the effectiveness of its aggregation mechanism. Specifically, we designed a variant called Onlytop, which retains only the highest-ranked hidden state for each item, as opposed to averaging all relevant hidden states to form the item representation. The experimental results are summarized in Table 2. We observed that the performance of Onlytop is significantly worse than that of $L2D$-L. This may be due to the fact that each hidden state of an item in the local memory represents an aspect of the item's characteristics that closely relate to the user's personalized preferences. By retaining only the highest-ranked hidden state, some important item characteristics may be discarded.

The impact of instruction tuning. During the instruction-tuning process, the hidden states of the LLM have captured the necessary historical user preferences. To highlight the importance of this tuning process, we designed a variant for comparison where $L2D$-L is applied to the original, non-fine-tuned LLM, called No-tune. Our observations show that No-tune performs significantly worse than $L2D$-L. This is because the hidden states of the original LLM are not aligned with recommendation tasks and cannot effectively capture user interests, making it challenging for $L2D$-L to extract useful information for recommendations.

Table 2: The performance of various versions of our proposed $L2D$ method is evaluated in the ablation study.

CDs	R@20	R@50	R@100	N@20	N@50	N@100
$L2D$-L	0.1158	0.1569	0.1996	0.0710	0.0792	0.0862
Only-top	0.1026	0.1346	0.1653	0.0570	0.0634	0.0683
No-tune	0.0688	0.1022	0.1321	0.0334	0.0400	0.0448
Games	R@20	R@50	R@100	N@20	N@50	N@100
$L2D$-L	0.0879	0.1465	0.2072	0.0399	0.0511	0.0596
Only-top	0.0654	0.1117	0.1679	0.0307	0.0399	0.0489
No-tune	0.0559	0.1021	0.1501	0.0222	0.0312	0.0390

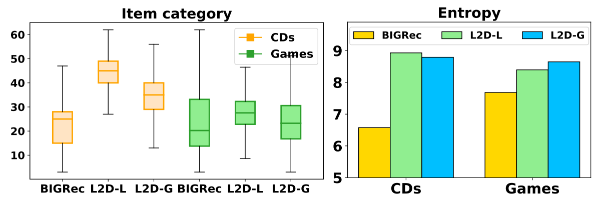
4.3.1 Sparse and Dense Scenario (RQ3)

$L2D$-L has demonstrated stable performance compared to $L2D$-G. However, since $L2D$-L was designed to create a user-specific local memory using the test sample's hidden state, it might be unsuitable for scenarios where user-item interactions are sparse. To analyze this, we divided the test set into sparse and dense categories based on item frequency in the training set. Figure 5 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 irrelevant information. (2) **Sparse scenarios:** the interactions are limited, which means that even the top similar hidden states 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.

In summary, both $L2D$-G and $L2D$-L have strengths and weaknesses, but our decoding strategy allows for flexible switching between the two methods, making it adaptable to different scenarios.

4.3.2 Recommendation Quality (RQ4)

In the previous section, we noted that auto-regressive decoding limits the ability to extract user interests from instructions. To demonstrate $L2D$'s ability to decode rich historical interests, we analyzed the diversity and popularity distribution of



(a) (b)

Figure 6: The recommendation quality comparison between $L2D$ and BIGRec.

recommended items. As shown in Figure 6a, $L2D$ offers a significantly greater variety of item categories compared to BIGRec, highlighting its ability to decode a wider range of user interests. Furthermore, we calculated the entropy of the probability of each item being recommended. A higher entropy indicates that the model recommends items more uniformly, regardless of their popularity. Figure 6b illustrates that $L2D$'s recommendations are able to cover a wider range of items, showcasing its capacity to personalize suggestions and recommend niche items to suitable users. Notably, $L2D$-L, which relies only on partial training samples of candidate items, still performs effectively in recommending niche items. We provide an explanation for this in Appendix A.5.

5 Related Work

• **LLM-based recommendation.** We discuss three paradigms of LLM-based recommenders (Wu et al., 2024b). (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., 2023b; 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., 2023a, 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., 2024), 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, the processes for obtaining historical sequence and candidate representations differ: our representations are determined by the generative state, whereas others are not. A detailed discussion is provided in Appendix A.2.

• **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 the importance of developing LLM-based recommenders to balance performance and inference overhead. 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 $L2D$ significantly reduces inference costs while achieving excellent performance. 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 struggles with handling new items, i.e., cold-start items, as these items do not have samples in the memory to model their implicit representations. 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. 2023a. 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.

Keqin Bao, Jizhi Zhang, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. 2023b. Tallrec: An effective and efficient tuning framework to align large language model with recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 1007–1014.

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*.

Jianchao Ji, Zelong Li, Shuyuan Xu, Wenyue Hua, Yingqiang Ge, Juntao Tan, and Yongfeng Zhang. 2024. Genrec: Large language model for generative recommendation. In *European Conference on Information Retrieval*, pages 494–502. Springer.

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.

Lei Li, Yongfeng Zhang, Dugang Liu, and Li Chen. 2024. Large language models for generative recommendation: A survey and visionary discussions. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 10146–10159.

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.

Jianghao Lin, Xinyi Dai, Yunjia Xi, Weiwen Liu, Bo Chen, Xiangyang Li, Chenxu Zhu, Huifeng Guo, Yong Yu, Ruiming Tang, and Weinan Zhang. 2023. How can recommender systems benefit from large language models: A survey. *CoRR*, abs/2306.05817.

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. 2024. Language models encode collaborative signals in recommendation.

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

Likang Wu, Zhi Zheng, Zhaopeng Qiu, Hao Wang, Hongchao Gu, Tingjia Shen, Chuan Qin, Chen Zhu, Hengshu Zhu, Qi Liu, et al. 2024b. 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.

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

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7655–7671, Bangkok, Thailand. Association for Computational Linguistics.

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.

Junjie Zhang, Ruobing Xie, Yupeng Hou, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. 2023a. Recommendation as instruction following: A large language model empowered recommendation approach. *arXiv preprint arXiv:2305.07001*.

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. 2023b. Collm: Integrating collaborative embeddings into large language models for recommendation. *CoRR*, abs/2310.19488.

Zihuai Zhao, Wenqi Fan, Jiatong Li, Yunqing Liu, Xiaowei Mei, Yiqi Wang, Zhen Wen, Fei Wang, Xiangyu Zhao, Jiliang Tang, and Qing Li. 2024. Recommender systems in the era of large language models (llms). *IEEE Trans. Knowl. Data Eng.*, 36(11):6889–6907.

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 Evaluation on Steam dataset.

In order to strengthen the generalizability claims of the method, we evaluated proposed $L2D$ on a widely-used non-Amazon dataset, Steam.

The results, summarized in the table 3, show that our method achieves the best performance on top-50 and top-100 metrics and performs comparably to the best baselines on top-20 metrics. Additionally, our method maintains the lowest inference time cost.

It is worth noting that the performance of $L2D$-G on Steam is nearly an order of magnitude worse than $L2D$-L. This phenomenon can be attributed to $L2D$-G being less effective in dense scenarios compared to $L2D$-L, while Steam is a denser dataset. This is supported by the results in Section 4.3.1, Sparse and Dense Scenario, where we analyzed the strengths and weaknesses of $L2D$-L and $L2D$-G across different sparsity levels. The experimental results indicate that $L2D$-G performs significantly better in sparse scenarios, whereas $L2D$-L excels in dense scenarios. Since Steam is denser than the Games and CDs datasets, this results in a substantial performance gap between $L2D$-G and $L2D$-L on the Steam dataset.

A.2 Contribution Positioning

Some existing (large) language model (LM)-based approaches (Sheng et al., 2024), 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.

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.3 Space Complexity Analysis

Let D be the hidden state size, L the number of layers, T_q and T_a the average lengths of the user query and generated item title, and N the number of beams. Since beam search replicates the inference process N times, the space complexity of language-space decoding is $O(NL(T_q+T_a)D)$, considering only hidden state storage. For our method $L2D$, space cost consists of two parts: LLM inference and storage of training data hidden states.

- LLM inference: We only need the last hidden state of the query, leading to $O(LT_qD)$.
- Training data storage:
  - L2D-G: Stores final representations for all items, with space complexity $O(DN_i)$, where N_i is the number of items.
  - L2D-L: Stores hidden states for all training samples, requiring $O(DN_t)$, where N_t is the total number of training samples.

Thus, the total space complexity is:

- L2D-G: $O(LT_qD) + O(DN_i)$
- L2D-L: $O(LT_qD) + O(DN_t)$

While our method incurs higher space costs compared to the baseline, inference latency poses a greater challenge in real-world recommendation applications. Sacrificing some space for lower latency is often acceptable. (Lin et al., 2024a; Xi et al., 2024c; Zhao et al., 2024; Lin et al., 2023).

A.4 Clarification of $L2D$-L Efficiency

In this section, we aim to clarify the efficiency issue of $L2D$-L. For $L2D$-L, the local aggregation

Table 3: The comparison of overall performance and inference time on the Steam dataset.

Model	R@20	R@50	R@100	N@20	N@50	N@100	Inference time
AlphaRec	0.1273	0.1813	0.2262	0.0506	0.0614	0.0686	362 s
GPT4Rec (beam = 1)	0.0293	0.0328	0.0389	0.0211	0.0217	0.0227	463 s
GPT4Rec (beam = 5)	0.0506	0.0604	0.0678	0.0270	0.0290	0.0302	1130 s
GPT4Rec (beam = 10)	0.0713	0.0840	0.0934	0.0319	0.0344	0.0359	1951 s
BIGRec (beam = 1)	0.0626	0.0994	0.1376	0.0317	0.0391	0.0453	466 s
BIGRec (beam = 5)	0.0701	0.1182	0.1767	0.0326	0.0421	0.0516	1133 s
BIGRec (beam = 10)	0.0851	0.1292	0.1962	0.0357	0.0443	0.0552	1957 s
D^3 (beam = 1)	0.0626	0.0994	0.1376	0.0317	0.0391	0.0453	457 s
D^3 (beam = 5)	0.0864	0.1379	0.1993	0.0461	0.0562	0.0661	1037 s
D^3 (beam = 10)	0.1139	0.1594	0.2221	0.0566	0.0655	0.0756	1593 s
$L2D$-G	0.0160	0.0390	0.0772	0.0059	0.0104	0.0165	26 s
$L2D$-L	0.1236	0.2237	0.3282	0.0494	0.0683	0.0839	39 s

process requires additional computation to determine similarity scores between the test sample's hidden state and all training samples in memory to identify the most relevant M samples. This raises concerns about whether the number of similarity score computations impacts the claimed computational efficiency.

In our proposed $L2D$-L, the number of similarity score computations would increase by $|M|$. However, the computation cost compared to the language-space decoding is very small. When increasing the value of M by 1000, the total inference cost for our method only increases by 2~3 seconds. Moreover, when setting M to a relatively smaller value, we can achieve better results than the baselines. In our experiments, setting M to 4000 for the CDs dataset could lead to significant performance improvements compared to the baseline, while the inference cost is only about 1/10 of that compared to BIGRec. Additionally, we could further leverage techniques like the approximate Nearest Neighbors search method to speed up the similarity computation process.

A.5 Effectiveness of $L2D$-L on Niche Items

The local aggregation mechanism in $L2D$-L relies on selecting the most similar Top-M training samples based on similarity scores. However, this raises the question: can $L2D$-L effectively recommend niche items with insufficient training samples? In this section, we will clearly demonstrate the advantages of our proposed method regarding this issue.

First of all, as shown in Figure 4.3.2, both $L2D$-G and $L2D$-L recommend a more diverse set of items across different categories and popularity levels compared to the baselines.

Additionally, the top-M selection process naturally increases attention to niche items as the selection is based on the test sample (query), retrieving the top-M most similar training samples. If the test sample is associated with less popular items, the retrieved training samples are also more likely to be related to less popular items.

A.6 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.

A.7 Dataset Statistics

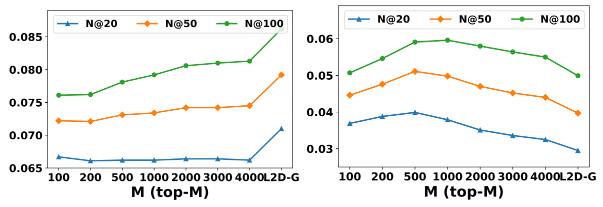
In this subsection, we supplement the statistical information of the datasets used in our experiments. The paper mainly presents the results of the CDs and Games datasets, while the experimental results of the Steam dataset can be found in the Appendix A.1.

A.8 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

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
Steam	54,206	8,268	177,046	22,130	22,130



(a) CDs NDCG (b) Games NDCG

Figure 7: 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.

method. As shown in Figure 7, the phenomenon observed in the NDCG metric is consistent with the Recall metric in the paper, further strengthening our argument.