DIP: Dynamic In-Context Planner For Diffusion Language Models

Anonymous ACL submission

Abstract

Diffusion language models (DLMs) have shown strong potential for general natural language tasks with in-context examples. However, due to the bidirectional attention mechanism, DLMs incur substantial computational cost as context length increases. This work addresses this issue with a key discovery: unlike the sequential generation in autoregressive language models (ARLMs), the diffusion generation paradigm in DLMs allows *efficient dynamic adjustment of the context* during generation. Building on this insight, we propose **D**ynamic **I**n-Context **P**lanner (DIP), a context-optimization method that dynamically selects and inserts in-context examples during generation, rather than providing all examples in the prompt upfront. Results show DIP maintains generation quality while achieving up to 12.9× inference speedup over standard inference and 1.17× over KV cache-enhanced inference.

1 Introduction & Related Work

Diffusion language models have emerged as a new paradigm for language modeling, demonstrating strong potential across general natural language tasks (Nie et al., 2025; Ye et al., 2025; Austin et al., 2021; DeepMind, 2025; Arriola et al., 2025; Lou et al., 2024; Sahoo et al., 2024; Lou et al., 2024). However, the DLM's decoding mechanism is fundamentally different. By utilizing bidirectional attention and iterative refinement, DLMs remove the strict left-to-right constraint inherent to AR generation. This paradigm shift offers fine-grained control over the generation process, enabling flexible editing and global planning (Zhang et al., 2025; Jin et al., 2025; Li et al., 2025; Chen et al., 2025).

Similar to ARLMs, DLMs also benefit from In-Context Learning (ICL), in which providing in-context examples improves performance on many retrieval-augmented generation tasks (Gao et al., 2023). Existing ICL approaches for DLMs largely inherit the AR convention (Radford et al., 2018; Bi et al., 2025), which keeps the prompt fixed and uses all in-context examples throughout generation (Radford et al., 2018; Dong et al., 2024). As shown in Figure 1, the static all-in-context example approach could lead to slower inference speed. Popular DLMs (Ye et al., 2025; Nie et al., 2025) utilize bidirectional attention, which scales poorly with context length. While recent optimizations, such as KV cache (Liu et al., 2025; Song et al., 2025), parallel decoding (Jiang et al., 2025; Wu et al., 2025), and block-wise decoding (Wu et al., 2025), can reduce the computation of static prompts, they do not resolve the fundamental inefficiency: the generative process still requires computing full attention over context during the generation process, which limits inference throughput (Wu et al., 2025).

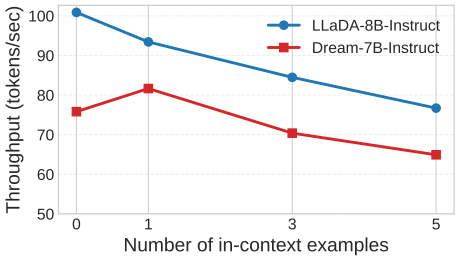

Figure 1: DLMs using Fast-dLLM (Wu et al., 2025) shows a negative correlation between throughput and the number of examples in the prompt.

This work addresses this issue with a **key discovery**: unlike the sequential generation, which requires a fixed context in ARLMs, the diffusion generation paradigm in DLMs is free from these constraints and allows *efficient dynamic adjustment of the context* during generation. Take the 3-shot GSM8k task as an example. In ARLMs, all examples must be provided at once in the prompt before generation begins. On the contrary, as illustrated in the middle column of Figure 2, the diffusion generation paradigm allows generation with 1 to 3

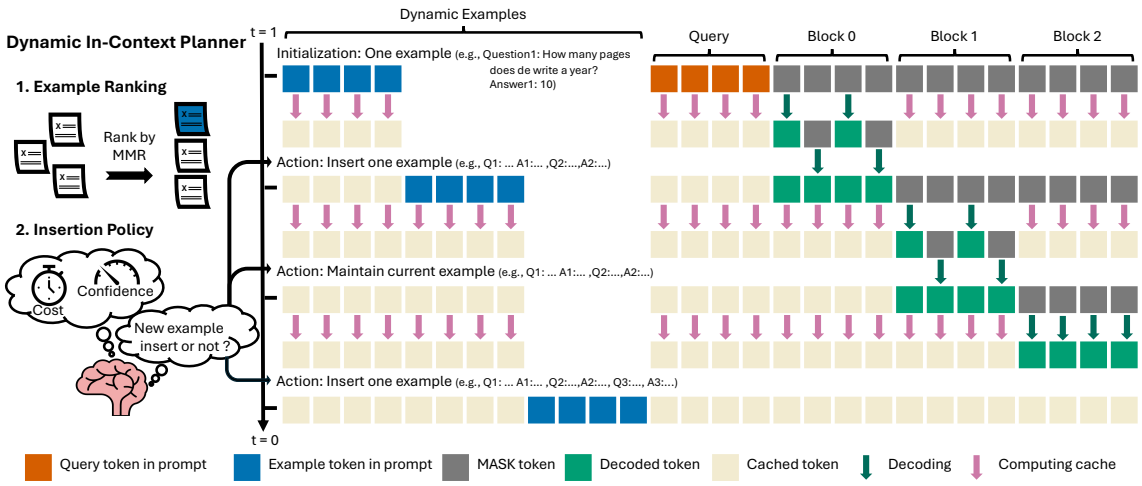

Figure 2: Dynamic In-Context Planner (DIP): (1) Example ranking stage uses MMR to rank the candidate examples by their marginal utility. (2) Insertion policy progressively inserts new examples into the context between blocks.

examples across different stages of the generation process (initialized with 1-shot at block 1, insert 1 example at block 2, ...). By dynamically selecting in-context examples based on the evolving generation state, we could increase inference throughput, ideally without compromising the generation quality.

Building on these insights, we propose **D**ynamic **I**n-Context **P**lanner (DIP), a context-optimization method that dynamically selects in-context examples during generation. DIP consists of a Maximal Marginal Relevance (MMR) based example ranking stage and a confidence- and generation-process-aware example insertion policy that performs insertion decisions during generation. DIP is training-free and can serve as a plug-in method that efficiently selects examples and updates the prompt on top of prominent DLM frameworks such as Fast-dLLM (Wu et al., 2025).

Our main contributions include: (1) We make a key discovery that the diffusion generation paradigm of DLMs allows efficient dynamic context adjustment, opening a new avenue for efficient DLMs inference based on dynamic context optimization. (2) We propose DIP, a practical context-optimization method that dynamically updates in-context examples during generation, which consistently yields inference speedup. (3) Empirical results on the 5-shot GSM8k task show that DIP can maintain generation quality while achieving up to 12.9× inference speedup over LLaDA's standard inference (Nie et al., 2025) and 1.17× over KV cache-enhanced inference like Fast-dLLM (Wu et al., 2025).

2 Masked DLMs with Dynamic Context

In this section, we briefly discuss the modeling details of DLMs that enable dynamic ICL with minimal overhead. Masked DLMs (the absorbing state discrete diffusion models) (Sahoo et al., 2024; Zhu et al., 2025; Ye et al., 2025) consist of a forward and reverse process, where an absorbing state is a mask token, denoted as [MASK], which represents missing or corrupted information (Lou et al., 2024; Nie et al., 2025).

In the *forward* process, q with an initial sequence y_0 of length N is organized into discrete *time steps*, where each time step corresponds to a single masking stage. At each time step, independent tokens are gradually replaced by [MASK] tokens with probability $t \in [0, 1]$, which is defined as:

$$q(y_t|y_0) = \prod_{i=0}^{N-1} q(y_t^i|y_0^i), \tag{1}$$

where $i \in [0, N]$ and y^i stands for the i-th token. Once a token transitions to [MASK], it remains in this state throughout the forward process. As t increases to 1, the original sequence will eventually become a fully masked sequence.

Conversely, in the *reverse* process p_θ, at each time step, masked DLMs aim to predict the probability distribution over all masked tokens simultaneously given y_t, which can be formulated as:

$$p_\theta(y_s|y_t) = \prod_{i=0}^{N-1} p_\theta(y_s^i|y_t^i). \tag{2}$$

Based on these predictions, the additional sampling method S decides which tokens to "unmask" at each step. As t decreases to 0, the masked sequence will eventually recover the original sequence.

Masked DLMs usually employ a bidirectional attention mechanism (e.g., BERT (Devlin et al., 2019)) and scale poorly with context length, as shown in Figure 1. To address this, recent KV cache inference frameworks often use block diffusion, dividing the overall generation length into b equal-length discrete blocks and performing semi-AR decoding on the block level. For instance, Fast-dLLM (Wu et al., 2025) employs a KV cache for prefix and masked-suffix tokens in non-decoding-blocks, achieving significant speedups.

3 Problem Statement

We formally define the new dynamic example optimization task: instead of using all examples as context throughout DLM's generation process, we aim to decide when and which examples to insert during generation to accelerate inference while maintaining inference quality.

Definition 1 ***(ICL Setting)*** *Given a prompt p with a pool of in-context examples $E_{pool} = \{(q_i, a_i)\}_{i=1}^{K}$, where q_i and a_i denote the query and answer of the i-th example, respectively.*

Definition 2 ***(Efficient Inference Setting)*** *Given a KV cache-based inference process (e.g., Fast-dLLM (Wu et al., 2025)) divided into N non-KV cache steps (or blocks), indexed by $b \in \{0, \cdots, N-1\}$.*

Goal: The goal is to find an optimal policy π that selects a subset of examples $E_b \subseteq \{E_{pool}, \emptyset\}$ for each block b, which maintains the generation quality while improving the overall throughput.

4 Dynamic In-Context Planner

As shown in Figure 2, the Dynamic In-Context Planner (DIP) is a two-stage framework that decouples selection complexity into: (1) example ranking stage and (2) example insertion policy. The *ranking stage* orders the candidate examples in E_{pool} by their marginal utility. Then, an *example insertion policy* is a monotonic-increasing policy that progressively inserts these examples into the context at different generation blocks b based on generation confidence and generation stage. A formal algorithm is outlined in Appendix A.

4.1 Block-wise Decoding with KV Cache

As DIP aims to optimize inference throughput, it is essential to ensure compatibility with the KV cache-enhanced inference framework. We build DIP on top of Fast-dLLM (Wu et al., 2025), a KV cache enhanced block-wise inference framework, which requires a necessary non-KV cache call at the start of discrete block decoding to update the KV cache, providing an opportunity to efficiently adjust the prompt during generation (details in Algorithm 1 of Appendix A). This update usually introduces minimal overhead to the generation process.

In principle, DIP is a plug-in method that works with any masked DLM KV cache enhanced inference framework, whereas a non-KV-cache call is required to update the KV cache.

4.2 Example Ranking

In the ranking stage, DIP aims to assign ranks to examples in E_{pool} based on their importance (Carbonell and Goldstein, 1998; Pickett et al., 2024; Nafee et al., 2025). Then we can use the top-related examples to guide our generation. To achieve this and balance the examples' relevance and diversity with the test input, DIP deploys Maximal Marginal Relevance (MMR) on the query sentence embeddings to derive a ranked list.

$$\begin{aligned}\mathrm{MMR}(q_i, \lambda) = \; & \lambda \cdot \mathrm{Sim}(q_p, q_i) \\ & - (1-\lambda) \max_{j \in S} \mathrm{Sim}(q_i, q_j)\,, \end{aligned} \tag{3}$$

where q_p denotes the representation of the actual query LLM aims to solve, $\mathcal{S}$ is the index set of already selected examples, and $\lambda \in [0, 1]$ controls the trade-off between relevance and redundancy.

4.3 Example Insertion Policy

Given ranked examples, DIP needs to decide when to insert the next-best example into the prompt and perform a prompt update. Intuitively, a good example-insertion policy should balance inference efficiency alone with generation confidence.

Inference efficiency. As illustrated in Figure 1, there is an inverse correlation between model inference throughput and the number of in-context examples. Since the cost of bidirectional attention scales poorly with sequence length, the policy should aim to minimize the context length to achieve greater inference gain. However, this is not always beneficial, as fewer examples can degrade performance in some cases. As the number of examples of our policy is non-decreasing, we will also gain greater inference speedup if we introduce new examples at later stages. Guided by those principles, we introduce a generation process penalty term G to scale the probability of inserting

an example as follows:

$$G(n, N, \epsilon) = (1 - \epsilon) + \epsilon \cdot (n/N) \ , \qquad (4)$$

where n stands for the current number of non-KV cache call, N stands for total number of non-KV cache call, and $\epsilon \in [0, 1]$ is a hyperparameter controlling the ratio of penalty.

Generation confidence guided. While our goal is to improve inference speed, we should also maintain the same generation quality. It is essential to detect changes in generation quality during the decoding process. How can we effectively measure the shift in generation quality? The answer is the confidence of the generated tokens. Confidence is a critical signal for sampling methods to decide which tokens to unmask at each step, thus a suitable measure without additional cost.

DIP's example insertion policy uses the average confidence across all generated tokens as a reference and compares it with the tokens generated in the current block or step. Then we define the probability for inserting a new example as the scaled ratio between two averages as follows:

$$P(\text{insert}) = \frac{1 - \mu}{2(1 - \bar{\mu})} \ , \qquad (5)$$

where μ stands for the average confidence of tokens from the current block and $\bar{\mu}$ stands for the running average of all generated tokens.

To incorporate the time penalty, DIP's final example insertion policy can be defined as a Bernoulli distribution as follows:

$$\begin{aligned} P(a = \text{insert}) &= P(\text{insert}) \cdot G(n, N, \epsilon) \\ P(a = \text{keep}) &= 1 - P(\text{insert}) \cdot G(n, N, \epsilon) \end{aligned} \qquad (6)$$

5 Experiment

5.1 Experiment Setup

We evaluate DIP on the representative LLaDA-Instruct model, with a generation length of 256 and block size of 32, on the 5-shot GSM8k task (Cobbe et al., 2021). For embedding used in the ranking stage, we use Qwen3-Embedding-0.6B throughout the experiment. To ensure reproducibility, our experiment uses the *lm-eval* library (Gao et al., 2024). For metrics, we compare the flexible match accuracy provided by the lm-eval library along with throughput, measured as the average number of output tokens generated per second. For baseline, we selected Fast-dLLm (Wu et al., 2025) and decoding from LLaDA (Zhu et al., 2025). We run all experiments on a single NVIDIA 5090 GPU to ensure a fair, consistent comparison.

Method	Throughput	Acc
LLaDA	6.94 (1.0×)	78.2
Fast-dLLM	75.87 (10.9×)	77.7
DIP(λ=0.1,ϵ=0.2)	86.54 (12.5×)	77.8
DIP(λ=0.1, ϵ=1)	89.47 (12.9×)	76.7

Table 1: Results on 5-shot GSM8k with LLaDA-8B-Instruct (Zhu et al., 2025). Blue: relative speedup.

Parameters	Throughput	Accuracy
$\epsilon = 0.2, \lambda = 0.0,$	87.45	76.6
$\epsilon = 0.2, \lambda = 0.25$	86.49	77.6
$\epsilon = 0.2, \lambda = 0.5$	86.44	76.3
$\epsilon = 0.2, \lambda = 0.75$	86.68	76.5
$\epsilon = 0.2, \lambda = 1.0$	86.46	77.3
$\lambda = 0.1, \epsilon = 0.0$	85.54	76.9
$\lambda = 0.1, \epsilon = 0.25$	86.27	77.8
$\lambda = 0.1, \epsilon = 0.5$	87.51	77.0
$\lambda = 0.1, \epsilon = 0.75$	88.81	76.0
$\lambda = 0.1, \epsilon = 1.0$	89.47	76.7

Table 2: Abalation study on DIP's insertion policy over choice of MMR parameter λ and generation process penalty parameter ϵ.

5.2 Results

In Table 1, we present the accuracy and throughput using DIP. Results show that DIP outperforms LLaDA's decoding and previous KV cache-only methods in terms of throughput. DIP can achieve up to 12.9× inference speedup compared to LLaDA's decoding and 1.17× over Fast-dLLM, while maintaining on-par accuracy. In Table 2, we present the results over the choice of hyperparameter. Results show that: (1) increasing the MMR parameter λ, which emphasizes embedding similarity over redundancy, has little effect on throughput but slightly affects accuracy; (2) increasing the generation process penalty parameter ϵ, which puts a larger penalty on early insertion, will increase the throughput but shows performance degradation.

6 Conclusion

In this paper, we made a key discovery: the diffusion generation paradigm of DLMs enables efficient context adjustment. Inspired by this, we propose DIP, a context optimization method that dynamically updates context examples during generation. Results show that DIP can maintain generation quality while achieving up to 12.9× inference speedup compared to standard inference and 1.17× compared to KV-cache enhanced inference.

Limitations

There are a few limitations to our work. First, due to computational constraints and time limitations, our experiment was conducted on the GSM8k benchmarks using LLaDA-8B-Instruct models. We encourage future work to explore DIP's performance across more tasks and models. Second, DIP is built and tested on the Fast-dLLM inference framework. While theoretically DIP is compatible with standard KV cache methods, we encourage future work to explore different inference frameworks with DIP. Lastly, we adopted the inference setup from Fast-dLLMs. We encourage future work to explore different inference settings with dynamic context.

References

Marianne Arriola, Subham Sekhar Sahoo, Aaron Gokaslan, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Justin T Chiu, and Volodymyr Kuleshov. 2025. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. 2021. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993.

Zhenyu Bi, Meng Lu, Yang Li, Swastik Roy, Weijie Guan, Morteza Ziyadi, and Xuan Wang. 2025. Optagent: Optimizing multi-agent llm interactions through verbal reinforcement learning for enhanced reasoning. *arXiv preprint arXiv:2510.18032*.

Jaime Carbonell and Jade Goldstein. 1998. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336.

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. 2025. dparallel: Learnable parallel decoding for dllms. *arXiv preprint arXiv:2509.26488*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

DeepMind. 2025. Gemini Diffusion — deepmind.google. [Accessed 01-01-2026].

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, and 1 others. 2024. A survey on in-context learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 1107–1128.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The language model evaluation harness.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1).

Yuchu Jiang, Yue Cai, Xiangzhong Luo, Jiale Fu, Jiarui Wang, Chonghan Liu, and Xu Yang. 2025. d^2 cache: Accelerating diffusion-based llms via dual adaptive caching. *arXiv preprint arXiv:2509.23094*.

Xiangqi Jin, Yuxuan Wang, Yifeng Gao, Zichen Wen, Biqing Qi, Dongrui Liu, and Linfeng Zhang. 2025. Thinking inside the mask: In-place prompting in diffusion llms. *arXiv preprint arXiv:2508.10736*.

Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Yi Liang, Soroush Vosoughi, and Shiwei Liu. 2025. Diffusion language models know the answer before decoding. In *NeurIPS 2025 Workshop on Efficient Reasoning*.

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. 2025. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*.

Aaron Lou, Chenlin Meng, and Stefano Ermon. 2024. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning*, pages 32819–32848.

Mahmud Wasif Nafee, Maiqi Jiang, Haipeng Chen, and Yanfu Zhang. 2025. Dynamic retriever for in-context knowledge editing via policy optimization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 16755–16768.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, JUN ZHOU, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025. Large language diffusion models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Marc Pickett, Jeremy Hartman, Ayan Kumar Bhowmick, Raquib-ul Alam, and Aditya Vempaty. 2024. Better rag using relevant information gain. *arXiv preprint arXiv:2407.12101*.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, and 1 others. 2018. Improving language understanding by generative pre-training.

Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. 2024. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184.

Yuerong Song, Xiaoran Liu, Ruixiao Li, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. 2025. Sparse-dllm: Accelerating diffusion llms with dynamic cache eviction. *arXiv preprint arXiv:2508.02558*.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. 2025. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. 2025. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*.

Lingzhe Zhang, Liancheng Fang, Chiming Duan, Minghua He, Leyi Pan, Pei Xiao, Shiyu Huang, Yunpeng Zhai, Xuming Hu, Philip S Yu, and 1 others. 2025. A survey on parallel text generation: From parallel decoding to diffusion language models. *arXiv preprint arXiv:2508.08712*.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, and 1 others. 2025. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*.

A Appendix: Algorithm for Fast-dLLM and DIP

Algorithm 1 presents the decoding process of Fast-dLLM (Wu et al., 2025). Building on top of Fast-dLLM, we present the algorithm of DIP in Algorithm 2.

B Appendix: Use Of Ai Assistants

In this paper, we used Gemini solely to help with grammar and fluency. We also use Copilot for syntax assistance.

C Appendix: Code Implementation

Please refer to the supplementary materials for the code implementation.

Algorithm 1: Fast-dLLM Decoding with Threshold

Input: model p_θ, prompt p, answer length L, blocks N, block size B, step per block T, threshold τ

$x \leftarrow$ [p;[MASK],...,[MASK]]
Initialize KV Cache for x. ▷▷▷ KV Cache Init
for n =*1 to* N **do**
  $s \leftarrow [|p| + (n-1)B]$, $e = |p| + nB$
  Update KV Cache with non-KV cache call ▷▷▷ KV Cache Update
  for $t = 1$ *to* T **do**
    Use cache, call p_θ on $x^{[s,e)}$ ▷▷▷ KV Cache Reuse
    For each masked x^i, confidence $c^i = \max_x p_\theta(x^i)$
    Unmask all i in $[s, e)$ with $c^i \geq \tau$, with index max c^i
    if *all* $x^{[s,e)}$! = *[MASK]* **then**
      break, move to next block
return x;

Algorithm 2: DIP with Fast-dLLM

Input: model p_θ, answer length L, blocks N, block size B, step per block T, threshold τ, example insertion policy π, examples set E, number of examples K, query q

Rank E by MMR, E_{rank} = $[E_1, E_2, ..., E_K]$ ▷▷▷ Example Ranking Stage
$p \leftarrow [E_1; q]$, $k = 1$ ▷▷▷ Initialize Prompt With π
$x \leftarrow$ [p; [MASK],...,[MASK]]
Initialize KV Cache for x. ▷▷▷ KV Cache Init
$\bar{\mu} = 0$ ▷▷▷ Running Confidence Mean
$\mu = 0$ ▷▷▷ Confidence Mean Of Current Block
for n =*1 to* N **do**
  $s \leftarrow [|p| + (n-1)B]$, $e = |p| + nB$
  if $n! = 1$ **then**
    Insertion = $\pi(\mu, \bar{\mu})$ ▷▷▷ Get Action Following π
    if *Insertion* $== True$ **then**
      $p \leftarrow [E_1, E_2, .., E_k; q]$, $k+ = 1$ ▷▷▷ Update Prompt
      $lp =$ length of p
      $x \leftarrow$ [p; [MASK],...,[MASK]]
    Update KV Cache with non-KV cache call ▷▷▷ KV Cache Update
  for $t = 1$ *to* T **do**
    Use cache, call p_θ on $x^{[s,e)}$ ▷▷▷ KV Cache Reuse
    For each masked x^i, confidence $c^i = \max_x p_\theta(x^i)$
    Unmask all i in $[s, e)$ with $c^i \geq \tau$, with index max c^i
    if *all* $x^{[s,e)}$! = *[MASK]* **then**
      μ = mean(c^i of $x^{[s,e)}$) ▷▷▷ Save Confidence Mean Of Current Block
      $\bar{\mu}$ = mean(c^i of $x^{[lp,e)}$) ▷▷▷ Save Confidence Mean Of Generated Tokens
      break, move to next block
    μ = mean(c^i of $x^{[s,e)}$) ▷▷▷ Save Confidence Mean Of Current Block
    $\bar{\mu}$ = mean(c^i of $x^{[lp,e)}$) ▷▷▷ Save Confidence Mean Of Generated Tokens
return x;