POLESTAR-CACHE: RECONCILING PARALLEL DECODING AND ACCURACY IN DIFFUSION LLMS VIA TOKEN DRIFT-AWARE KV CACHE RECALIBRATION

Mingyu Lee[1*], **Akshat Ramachandran**[1*], **Souvik Kundu**[2], **and Tushar Krishna**[1]
[1]Georgia Institute of Technology, Atlanta, USA
[2]Intel, Santa Clara, USA
[1]mlee864@gatech.edu akshat.r@gatech.edu, tushar@ece.gatech.edu
[2]souvikk.kundu@intel.com
*Equal Contribution

ABSTRACT

Diffusion language models (dLLMs) offer a promising alternative to autoregressive generation, but their inference efficiency remains limited by high compute cost and instability under approximate key–value (KV) cache reuse. We show that existing KV-cache–enabled dLLM inference schemes suffer from token drift, where cached representations become misaligned with evolving context, leading to degraded prediction confidence, reduced parallelism, and accuracy loss. To address this, we propose Polestar-Cache, a training-free, drift-aware KV caching framework that reconciles parallel decoding with accuracy. Polestar-Cache detects layer-wise representation drift using a lightweight KL-divergence proxy computed from cached hidden states, and selectively refreshes KV cache entries. To reduce overhead, Polestar-Cache clusters hidden states and keeps only centroids on the GPU. The non-centroid tokens are offloaded to CPU memory and fetched according to drift dynamics. Extensive experiments on multiple dLLM benchmarks, including GSM8K, MBPP, and ParallelBench, demonstrate that Polestar-Cache achieves up to 11% accuracy improvement and $1.7\times$ performance improvement over prior KV-cache–enabled baselines.

1 INTRODUCTION

Autoregressive large language models (LLMs) Grattafiori et al. (2024); Jaech et al. (2024) have thus far been the dominant paradigm for sequence modeling Zhao et al. (2023), and have achieved remarkable success in natural language generation tasks, including complex, multi-step reasoning Yuan et al. (2025) and code generation Jiang et al. (2024). However, the causal sequential nature of autoregressive decoding, fundamentally limits their inference speed, particularly for long generation lengths Kang et al. (2025).

Recently, masked diffusion language models (dLLMs) Nie et al. (2025); Ye et al. (2025); Song et al. (2025) have emerged as a promising alternative to autoregressive models. Drawing inspiration from diffusion-based generative modeling, dLLMs reformulate sequence generation as an iterative denoising process Austin et al. (2021a), enabling varying degrees of parallel decoding across tokens Arriola et al. (2025).

Despite this parallelism, dLLMs are often more compute-intensive than their autoregressive counterparts, as generation requires multiple denoising iterations over the full sequence. Moreover, unlike autoregressive LLMs, dLLMs do not naturally admit key–value (KV) caching mechanisms, since token representations are repeatedly revised across denoising steps, causing cached states to become stale or invalid Wu et al. (2025).

In an effort to reduce the compute intensity of dLLMs and improve inference efficiency—thereby capitalizing on their inherent parallelism—recent methods Liu et al. (2025); Wu et al. (2025); Ma et al. (2025); Shen et al. (2025) have explored approximate KV caching techniques to reduce *per-step inference latency*. For example, Fast-dLLM Wu et al. (2025) adopts a block-wise decoding

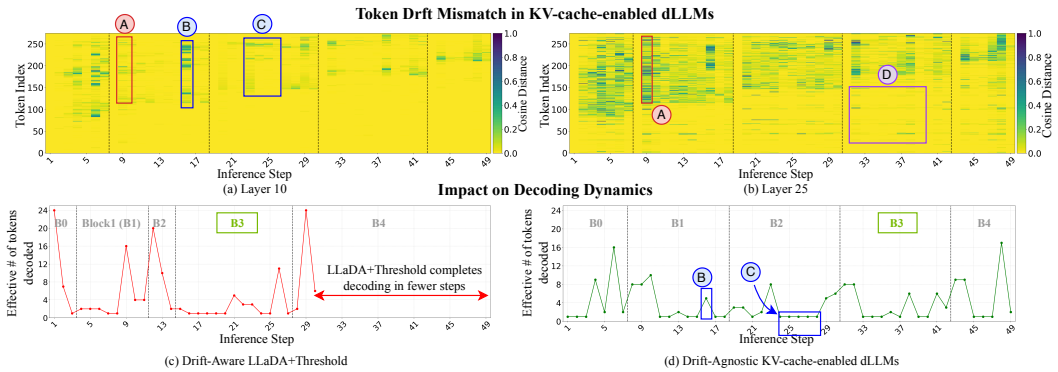

Figure 1: Token drift and its impact on parallel decoding efficiency for LLaDA-8B on GSM8K. Token-wise cosine distance heatmaps between cached and freshly recomputed KV representations at (a) layer 9 and (b) layer 25. (c) Decoding trajectory of vanilla LLaDA with confidence-based parallel decoding and (d) Decoding trajectory of LLaDA with KV-cache-enabled.

schedule and stores a KV cache for all tokens outside the current decoding block. Upon decoding completion of the block, it then refreshes the KV cache with the newly generated tokens. Such KV caching techniques are predicated on the assumption that, KV cache token representations are invariant to the decoding of tokens within the current block. However, in dLLMs, we observe that as tokens are decoded within a block, they can alter the contextual dependencies of previously cached tokens. This may cause their representations to *drift* across successive decode-steps and layers. This drift can compromise the reuse of KV cache across decoding steps of the current block, substantially degrading the model's prediction confidence and accuracy for subsequent tokens. This reduction in prediction confidence also limits achievable parallelism (tokens per forward (TPF) see Appendix A) and increases the number of decode-steps, offsetting per-step latency reductions.

Our contributions. We observe that in KV-cache-enabled dLLMs Wu et al. (2025), increasing decoding parallelism while achieving end-to-end performance gains without degradation of generation quality yields a unique and open challenge. This naturally leads to the following question: *How can a KV caching mechanism be designed to remain aligned with the evolving, drift-prone token representations in dLLMs—so as to preserve parallel decoding confidence and output quality—while still enabling efficient, high-parallelism inference?*

Towards addressing this, we introduce **Polestar-Cache**, a training-free algorithm-system co-design framework for diffusion language models that reconciles high decoding parallelism with output quality. Specifically, Polestar-Cache caches input hidden states and computes attention scores from these cached representations to detect layerwise representation drift via a lightweight KL-divergence proxy across successive decoding steps. When drift exceeds a threshold, it selectively updates the KV cache at the affected layer and propagates update signals to subsequent layers, ensuring consistent contextual alignment throughout the model.

Additionally, to preserve inference efficiency, Polestar-Cache eschews token-level drift tracking by clustering cached hidden states via spherical K-means, offloading clusters to the CPU while retaining only centroids on the GPU. Drift-triggered updates are applied sparingly and selectively fetch hidden states for high-drift clusters, minimizing overhead without compromising cache fidelity.

Extensive experiments on diverse dLLM model families and benchmarks demonstrate that Polestar-Cache achieves up to 11% **accuracy improvement** and $1.7\times$ **end-to-end inference speedup** over prior KV-cache–enabled dLLM baselines.

2 Related Work

KV cache optimizations for dLLMs. This line of research accelerates dLLM inference by reducing the computational cost of each decoding step through KV cache reuse. Fast-dLLM Wu et al. (2025) adopts a hybrid strategy that additionally introduces a block-wise approximation of the KV cache, which is refreshed at block boundaries. dLLM-Cache Liu et al. (2025) and dKV-Cache Ma et al. (2025) performs sporadic, similarity-guided or delayed updates to the KV cache of response

blocks. Elastic-Cache Nguyen-Tri et al. (2025b) adaptively performs layer-aware KV cache updates based on the most attended tokens, but incurs significant caching overhead. While these techniques reduce per-step inference latency, they assume cached token representations remain static within a block and ignore contextual drift. Such drift undermines KV cache reuse, degrades prediction confidence, and consequently reduces parallelism and increases decoding steps.

Parallel decoding with dLLMs. A growing body of work improves dLLM inference efficiency by increasing decoding parallelism using token-level uncertainty, adaptive block sizing, or hierarchical decoding strategies Lu et al. (2025); Anonymous (2025); Wu et al. (2025). While effective at scheduling parallel decoding, these methods largely focus on block formation and remain orthogonal to ensuring representation consistency under cache reuse. In contrast, our work addresses this complementary challenge, enabling parallel decoding to translate into reliable end-to-end performance gains.

3 Motivating Analysis

dLLMs employ fully bidirectional attention Zhu et al. (2025), causing intermediate activations (hidden states, keys, and values) to evolve as newly decoded context is incorporated during decoding Nguyen-Tri et al. (2025a). Consequently, even tokens outside the active decoding block remain context-dependent, making temporally invariant representations incompatible with dLLM inference. We refer the reader to Appendix A for background on diffusion language models and relevant terminology and notations.

Definition 1: Token Drift in dLLMs. The temporal (across decoding steps) and spatial (across layers) evolution of a model's internal representations—particularly keys and values—induced by the progressive incorporation of newly decoded contextual information under fully bidirectional attention.

KV-cache-enabled dLLM inference acceleration schemes reuse cached KV representations across decoding steps by assuming that tokens outside the active block remain stable while a block is currently decoding. This assumption ignores the natural token drift in dLLMs. We investigate on how the token drift agnostic nature of KV-cache-enabled inference schemes impact the decoding confidence, TPF and overall output quality.

Setup. We study the impact of token drift using a representative KV-cache-enabled inference scheme, Fast-dLLM Wu et al. (2025). Given an input prompt, we conduct inference with a KV-cache–enabled dLLM (§A.2), which iteratively decodes tokens at each step. At each decoding step along this trajectory, we perform two forward passes for every layer using the same input hidden states: (i) a KV-cache-enabled pass that reuses cached KV representations outside the current decoding block, and (ii) a reference pass that fully recomputes KV representations for all tokens. We then measure the token drift error between the cached and fully recomputed KV representations, excluding tokens in the currently decoded block. This paired evaluation allows us to isolate the effect of KV cache reuse under a uniform decoding trajectory and explicitly characterize the mismatch between cached and true representations.

Measuring Token Drift Error. We quantify token drift error by measuring the token-wise cosine distance. For each token index $i \in \mathcal{I}$, we define it as,

$$D_i^{(t,\ell)} \triangleq 1 - \frac{1}{2}\Big(\texttt{cos\_sim}\big(\mathbf{K}_{[i]}^{(t,\ell)}, \mathbf{K}_{[i],\text{ref}}^{(t,\ell)}\big) + \texttt{cos\_sim}\big(\mathbf{V}_{[i]}^{(t,\ell)}, \mathbf{V}_{[i],\text{ref}}^{(t,\ell)}\big)\Big), \tag{1}$$

where, $\mathbf{K}_{[i],\text{ref}}^{(t,\ell)}, \mathbf{V}_{[i],\text{ref}}^{(t,\ell)}$ denote the fully computed KV representation. Cosine similarity ($\texttt{cos\_sim}(\cdot)$)is computed independently for each attention head and averaged across all heads. Figure 1(a,b) show token drift errors across decoding steps at two layers for a GSM8K prompt. Figure 1(c) depicts the ideal decoding trajectory ($N^{(t)}$ vs. steps) of a LLaDA+Threshold baseline Nie et al. (2025), while Figure 1(d) reports the decoding behavior under our KV-cache–enabled setup.

Observation 1: Token drift error is relatively localized in early layers, where tokens introducing drift error can be clearly identified. However, this drift error progressively accumulates and ampli-

fies in deeper layers as representation mismatches compound through the model (annotation Ⓐ). Notably, in deeper layers, drift error emerges in prefix tokens as well (annotation Ⓓ),

Observation 2: Decoding steps with higher $N^{(t)}$ (see §A) are associated with more pronounced token drift errors (annotation Ⓑ). Consequently, subsequent decoding steps exhibit severely degraded or noisy $N^{(t)}$ (annotation Ⓒ), indicating diminished prediction confidence.

Observation 3: As exemplified in block 3 of Figure 1(c,d), KV-cache–enabled inference exhibits systematic misallocation of parallelism. In regions where the baseline shows low $N^{(t)}$, indicating hard-to-decode regions, KV-cache–enabled inference overcommits tokens, whereas in easier regions—where the baseline achieves high $N^{(t)}$—it undercommits. This behavior arises from the inability of the KV cache to adapt to token drift, leading to noisy confidence estimates and, ultimately, degradation in output quality. See appendix for block-wise generation traces for both inference schemes.

Collectively, these findings motivate the need for a drift-aware KV caching mechanism that preserves per-step decoding efficiency while remaining aligned with the evolving token representations in dLLMs.

4 POLESTAR-CACHE METHODOLOGY

4.1 CLUSTERING AND OFFLOADING

To enable accurate KV cache updates, we cache per-layer hidden states of all token positions in $\mathcal{I}$ at the initial decoding step of each block. Let $\mathcal{D} \subseteq \mathcal{I}$ denote the subset of tokens that drift. To circumvent the prohibitive memory and computational overhead of token-level drift tracking we cluster cached hidden states using spherical K-means Dhillon & Modha (2001). The cluster centroids act as representatives for approximating drift dynamics of it's members. Let $\mathcal{C}$ denote the number of total number of clusters. We separately cluster tokens in the prefix and suffix span of the current decoding block, allocating the $\mathcal{C}$ clusters proportional to their respective token lengths.

The cluster centroids are retained in GPU memory, while the non-centroid tokens in the cluster are offloaded to CPU memory. To reduce memory bandwidth overhead during token transfers non-centroid hidden states are stored as NVFP4 Abecassis et al. (2025) residuals relative to their corresponding cluster centroids (delta quantization).

4.2 DRIFT-AWARE KV CACHE UPDATE

As discussed in §3, token drift can be accurately quantified by the cosine distance between cached and fully recomputed KV representations; however, computing this metric is infeasible in practice, as it requires two full model forward passes per decoding step. To enable efficient drift detection during inference, we propose a proxy metric based on the KL divergence between attention distributions (see §5). Specifically, for each layer (ℓ) at every decoding step, Polestar-Cache calculates a proxy attention between the cluster centroids and the KV cache. KL-divergence is measured between the proxy attention of the current decoding step and the previous decoding step.

Based on the KL divergence scores, Polestar-Cache identifies the top-M drifting clusters. The $\mathcal{D}$ non-centroid tokens corresponding to the top-M selected clusters are fetched from CPU memory and this operation is efficiently parallelized with the layer's current block attention computation. The loaded non-centroid tokens are dequantized and these hidden state tokens are passed through the feed-forward transformations and self-attention of the layer to obtain the drifted output representations. The set $\mathcal{D}$ and the drifted output representations are forward to the next layer ($\ell + 1$), which it uses to update it's KV cache and cached hidden states. Upon updating hidden states and KV cache entries, we update the corresponding cluster centroids.

4.3 SELECTIVE KV-CACHE UPDATE

Motivated by Observation 2 in §3, which shows that multi-token decoding can induce abrupt and amplified representation drift, Polestar-Cache employs a drift-aware selective update strategy. We monitor the number of tokens decoded between successive unmasking steps: if more than $\mathcal{T}$ tokens

are decoded in a single step, an immediate update is triggered (§4.2) to recalibrate internal representations. Otherwise, updates are triggered once the cumulative number of decoded tokens across steps exceeds $\mathcal{T}$. This adaptive triggering allows Polestar-Cache to respond promptly to bursty decoding while avoiding unnecessary updates during stable regimes.

5 Experimental Evaluations

5.1 Experimental Setup

Models and Datasets. We evaluate Polestar-Cache on LLaDa-Instruct-8B Nie et al. (2025) across ParallelBench Kang et al. (2025), GSM8K Cobbe et al. (2021) and MBPP Austin et al. (2021b) datasets.

Hyperparameters. We set number of clusters $\mathcal{C} = 8$, $M = 2$ to select top-2 clusters and selective update threshold of $\mathcal{T} = 4$. All hyperparameters have been empirically determined.

Table 1: Polestar-Cache accuracy comparison with baselines on GSM8K and MBPP datasets.

Method	GSM8K (%)	MBPP (%)
Vanilla LLaDA	78.79	45.8
LLaDA + Conf.	78.79	45.0
Fast-dLLM	76.52	42.0
Polestar-Cache	**79.55**	**44.0**

Baselines. We compare against vanilla LLaDA Nie et al. (2025), LLaDA+confidence threshold (parallel decoding) and Fast-dLLM Wu et al. (2025).

Implementation. Polestar-Cache is fully implemented in PyTorch, with all system optimizations realized through efficient Triton kernels. We employ a generation length of 256 and a block size of 32 following Wu et al. (2025).

5.2 Main Results

Accuracy Evaluation. As shown in Table 1, across both math (GSM8K) and coding (MBPP) benchmarks, Polestar-Cache consistently improves accuracy over existing baselines. In particular, Polestar-Cache achieves up to a 3% accuracy improvement over Fast-dLLM, a SoTA KV-cache–enabled dLLM inference approach. On ParallelBench—a harder benchmark designed to stress-test dLLMs under parallel decoding—Polestar-Cache substantially outperforms Fast-dLLM, delivering $\sim$ 11% absolute accuracy improvement (Figure 2).

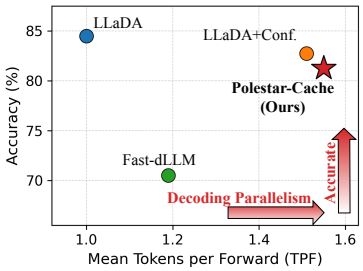

Figure 2: Accuracy-parallelism comparison on ParallelBench.

Parallelism Improvement. As shown in Figure 2, Fast-dLLM achieves low TPF due to drift-agnostic KV cache reuse, which degrades prediction confidence and limits parallel decoding. By contrast, Polestar-Cache's drift-aware updates enables substantially higher TPF. Consequently, Polestar-Cache surpasses Fast-dLLM in decoding parallelism and even outperforms LLaDA with confidence-based parallel decoding in TPF, while closely matching its accuracy, despite enabling KV-cache.

End-to-End Performance Comparison. As shown in Table 2, Polestar-Cache delivers substantial end-to-end speedups over existing approaches. Compared to vanilla LLaDA, Polestar-Cache achieves a 17.65× normalized speedup, significantly outperforming Fast-dLLM (10.39×).

Table 2: End-to-end runtime comparison.

Method	Normalized Speedup
Vanilla LLaDA	1×
Fast-dLLM	10.39×
Polestar-Cache	**17.65×**

6 Conclusion

We presented Polestar-Cache, a training-free, drift-aware KV caching framework that reconciles parallel decoding and accuracy in dLLMs by selectively updating stale representations. By explicitly accounting for token drift, Polestar-Cache improves both decoding parallelism and output quality, translating per-step efficiency gains into consistent end-to-end performance improvements. We leave a comprehensive evaluation across additional model families and detailed ablation studies to future work.

ACKNOWLEDGEMENTS

This work was supported in part by CoCoSys, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

Felix Abecassis, Anjulie Agrusa, Dong Ahn, Jonah Alben, Stefania Alborghetti, Michael Andersch, Sivakumar Arayandi, Alexis Bjorlin, Aaron Blakeman, Evan Briones, et al. Pretraining large language models with nvfp4. *arXiv preprint arXiv:2509.25149*, 2025.

Anonymous. Hierarchy decoding: A training-free parallel decoding strategy for diffusion large language models. In *Submitted to The Fourteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ZsIQUjQtdW. under review.

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.

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

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021b.

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

Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, 2001. doi: 10.1023/A:1007612920971. URL https://doi.org/10.1023/A:1007612920971.

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

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjae Lee, Yuchen Zeng, Shuibai Zhang, Coleman Hooper, Yuezhou Hu, Hyung Il Koo, Nam Ik Cho, et al. Parallelbench: Understanding the tradeoffs of parallel decoding in diffusion llms. *arXiv preprint arXiv:2510.04767*, 2025.

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

Guanxi Lu, Hao Mark Chen, Yuto Karashima, Zhican Wang, Daichi Fujiki, and Hongxiang Fan. Adablock-dllm: Semantic-aware diffusion llm inference via adaptive block size. *arXiv preprint arXiv:2509.26432*, 2025.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025.

Quan Nguyen-Tri, Mukul Ranjan, and Zhiqiang Shen. Attention is all you need for kv cache in diffusion llms. *arXiv preprint arXiv:2510.14973*, 2025a.

Quan Nguyen-Tri, Mukul Ranjan, and Zhiqiang Shen. Attention is all you need for kv cache in diffusion llms, 2025b. URL https://arxiv.org/abs/2510.14973.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Jucheng Shen, Gaurav Sarkar, Yeonju Ro, Sharath Nittur Sridhar, Zhangyang Wang, Aditya Akella, and Souvik Kundu. Improving the throughput of diffusion-based large language models via a training-free confidence-aware calibration. *arXiv preprint arXiv:2512.07173*, 2025.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

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

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

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, YX Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*, 2025.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2), 2023.

Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, et al. Llada-moe: A sparse moe diffusion language model. *arXiv preprint arXiv:2509.24389*, 2025.

A BACKGROUND AND PRELIMINARIES

A.1 INFERENCE WITH DLLMS

Let $\mathcal{V}$ denote the token vocabulary that includes a special mask token `[MASK]`. Given an input prompt $\mathbf{p}$ of length L_p, where $\mathbf{p} = (p_0, \ldots, p_{L_p-1}) \in \mathcal{V}^{L_p}$. With a generation token budget of L, a dLLM generates a response over T discrete denoise–sample steps indexed by $t \in \{T, T-1, \ldots, 0\}$. The tokens can be indexed by the set $\mathcal{J} \triangleq \{0, 1, \ldots, L_p + L - 1\}$. At step t, the sequence state is $\mathbf{y}^{(t)} = (y_i^{(t)})_{i \in \mathcal{J}} \in \mathcal{V}^{L_p+L}$, with the initial state,

$$\mathbf{y}^{(T)} = (\mathbf{p}, \underbrace{\texttt{[MASK]}, \ldots, \texttt{[MASK]}}_{L \text{ tokens}}) \tag{2}$$

Denoise. At each step t, a mask predictor m_θ produces a provisional prediction $\hat{\mathbf{y}}^{(t)}$ via greedy decoding:

$$\hat{y}_i^{(t)} = \arg \max_{v \in \mathcal{V} \setminus \texttt{[MASK]}} m_\theta(v \mid \mathbf{y}^{(t)}, i), \quad \forall i \in \mathcal{J}. \tag{3}$$

Semi-autoregressive decoding. In this process, the generation length L is partitioned into B contiguous blocks of fixed length. Decoding proceeds sequentially across blocks from left to right with tokens decoded in parallel within a block. Let $\mathcal{J}_{b^*} \subseteq \mathcal{J}$ denote the index set of the tokens in the currently decoding block. At step t, the masked positions within the block are $\mathcal{M}^{(t)} \triangleq \{i \in \mathcal{J}_{b^*} \mid y_i^{(t)} = \texttt{[MASK]}\}$. The model's certainty in its provisional prediction $\hat{y}_i^{(t)}$ at position i is measured by a confidence score $c_i^{(t)}$. Based on this score the unmask position is determined by a sampler set $\mathcal{S}^{(t)} \triangleq \{i \in \mathcal{M}^{(t)} \mid c_i^{(t)} \geq \tau\}$, where τ is a predefined confidence threshold. The sequence is then updated as

$$y_i^{(t-1)} = \begin{cases} \hat{y}_i^{(t)}, & i \in \mathcal{S}^{(t)}, \\ \texttt{[MASK]}, & i \in \mathcal{M}^{(t)} \setminus \mathcal{S}^{(t)}, \\ y_i^{(t)}, & i \notin \mathcal{J}_{b^*} \end{cases} \tag{4}$$

The denoise–sample cycle is repeated until all positions in $\mathcal{J}_{b^*}$ are unmasked, at which point decoding advances to the next block. This block-wise transition enforces autoregressive dependencies across blocks while allowing parallel decoding within each block.

Definition 2: Tokens per Forward (TPF). After a single forward pass of the dLLM, at step t, a subset of token positions $\mathcal{S}^{(t)}$ is selected to be unmasked. The *effective number of tokens* decoded is,

$$N^{(t)} \triangleq \left| \left\{ i \in \mathcal{S}^{(t)} \;\middle|\; \hat{y}_i^{(t)} \notin \{\texttt{[EOS]}\} \right\} \right| \tag{5}$$

We define the TPF as the average effective number of tokens decoded per forward pass over the entire decoding process:

$$\text{TPF} \triangleq \frac{1}{T} \sum_{t=1}^{T} N^{(t)} \tag{6}$$

A.2 DLLM INFERENCE WITH KV CACHE

For a dLLM layer ℓ at decoding step t, let $\mathcal{I} \subseteq \mathcal{J}$ denote the set of token indices whose key and value cache is denoted as $\mathbf{K}_{\mathcal{I}}^{(t,\ell)} = \{\mathbf{K}_{[i]}^{(t,\ell)}\}_{i \in \mathcal{I}}$ and $\mathbf{V}_{\mathcal{I}}^{(t,\ell)} = \{\mathbf{V}_{[i]}^{(t,\ell)}\}_{i \in \mathcal{I}}$.

Under block-wise KV caching, prior to decoding a block, the KV representations for all token positions outside the current block (b^*) are computed once and cached across multiple decoding steps. The cached token positions are given by $\mathcal{I} \triangleq \mathcal{J} \setminus \mathcal{J}_{b^*}$. At each decoding step t, KV tensors for tokens in the active block $\mathcal{J}_{b^*}$ are computed from the current hidden states $\mathbf{h}_{\mathcal{J}_{b^*}}^{t,\ell}$ via linear

projections. Queries from the active block attend jointly to cached KV in $\mathcal{I}$ and the KV tensors of the active block as,

$$\mathrm{Attn}^{(t,\ell)}_{\mathcal{J}_{b^*}} = \mathrm{Attn}\Big(\mathbf{Q}^{(t,\ell)}_{\mathcal{J}_{b^*}},\ \mathbf{K}^{(t,\ell)}_{\mathcal{I}} \cup \mathbf{K}^{(t,\ell)}_{\mathcal{J}_{b^*}},\ \mathbf{V}^{(t,\ell)}_{\mathcal{I}} \cup \mathbf{V}^{(t,\ell)}_{\mathcal{J}_{b^*}}\Big) \tag{7}$$

Once all token positions in $\mathcal{J}_{b^*}$ are decoded, the KV cache is refreshed in preparation for decoding the next block. Specifically, the KV representations of the completed block are inserted into the cache, while the KV representations corresponding to the next active block are removed, yielding the updated cache index set

$$\mathcal{I} \leftarrow (\mathcal{I} \cup \mathcal{J}_{b^*}) \setminus \mathcal{J}_{b^*+1} \tag{8}$$