Polestar: Drift-Aware Cache Calibration and Token Commitment for Efficient Inference of Diffusion LLMs

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

Abstract

The inference efficiency of diffusion large language models (dLLMs) is constrained by two challenges: bidirectional attention precludes efficient KV-cache reuse, while increasing decoding parallelism with static confidence thresholds can compromise generation quality. We observe that both challenges arise from a shared phenomenon: as tokens are decoded, their contextual integration through bidirectional attention causes token representations to drift (evolve) across decoding steps. This insight motivates **Polestar**, a training-free inference framework that uses *token representation drift* as a unified signal to jointly address both challenges. Polestar comprises two components: **Polestar-Cache**, which identifies stale KV-cache positions via drift and performs sparse KV-cache refreshes to enable efficient reuse, and **Polestar-Commit**, which detects sharp drift events to reliably identify commit-ready tokens. Across mathematics and coding benchmarks on several dLLM families, Polestar sets a new state of the art on the accuracy–throughput Pareto frontier, achieving up to $\mathbf{10.73}\%$ accuracy improvement, up to $\mathbf{3.7}\times$ higher throughput, and high decoding parallelism of 3.67 tokens per forward pass over existing baselines. Code: `https://github.com/gt-synergy-lab/Polestar`

1 Introduction

The inherent sequential dependency of autoregressive large language models (AR-LLMs) [11, 33, 39, 34] limits decoding parallelism and often leads to substantial inference performance degradation, particularly for long-form generation [47, 36]. Recently, masked diffusion LLMs (dLLMs) [29, 44] have emerged as an encouraging alternative to AR-LLMs. dLLMs enable parallel token decoding by reformulating generation as an iterative, multi-step denoising process [2] that progressively refines noisy tokens into clean predictions [1].

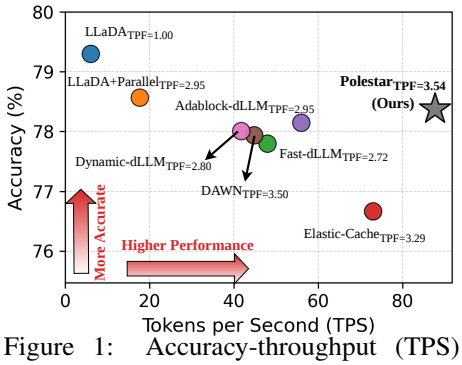


Figure 1: Accuracy-throughput (TPS) trade-off on GSM8K using LLaDA-8B-Instruct, with each method subscripted by its achieved tokens per forward pass (TPF).

However, dLLMs face a trade-off between parallelism and output quality, with the best accuracy often achieved when decoding one token per forward pass [29, 44] (see LLaDA in Figure 1). Moreover, their bidirectional attention precludes standard AR-LLM key–value (KV) caching, necessitating KV recomputation at each decode step, resulting in reduced performance compared to AR-LLMs [22].

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

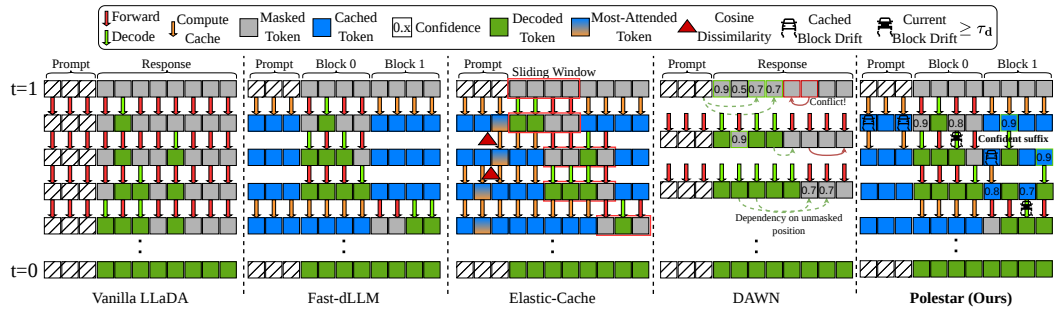


Figure 2: Illustration of dLLM inference strategies. *Vanilla LLaDA* performs full-sequence recomputation at each step. *Fast-dLLM* uses block-wise KV caching with confidence-based parallel decoding [42]. *Elastic-Cache* triggers adaptive KV refreshes by monitoring increase in cosine dissimilarity of the most-attended tokens [28]. *DAWN* uses dependency-aware decoding to improve decoding parallelism [25]. **Polestar** employs token representation drift to selectively and sporadically refresh stale KV cache positions and commit tokens earlier in both the current and suffix block.

Existing techniques explore training-free inference optimizations to address the above inherent limitations of dLLMs along two primary axes: Ⓐ reducing per-step latency, and Ⓑ increasing decoding parallelism to reduce the number of steps. Figure 2 illustrates contemporary strategies across the two axes for optimizing dLLM inference. We defer an extended discussion to §A

Reducing per-step latency: Prior works reduce the computational cost of a decoding step via KV cache reuse with selective updates to mitigate cache staleness. Fast-dLLM [42] adopts a block-wise caching that reuses KV representations outside the current decoding block with periodic refresh at block boundaries. Elastic-Cache [28] employs a sliding-window scheme with adaptive, layer-wise updates based on signals from the most-attended tokens. d^2Cache [16] selectively updates KV cache using token-level uncertainty. However, these methods either assume static representations within a block ([42]) or collect coarse proxy signals from individual tokens ([28]) or layer statistics ([43]), treating them as representative of the entire sequence. This ignores the joint evolution of token representations across steps and layers, leading to miscalculated cache validity. Thus, stale KV cache degrades prediction confidence and limits parallelism while requiring redundant computation.

Increasing decoding parallelism: Fast-dLLM [42] uses static confidence thresholds for token commitment, but fixed thresholds fail to adapt across decoding steps and inputs. Other works [23, 37] adapt block sizes based on confidence volatility, but introduces irregular execution and reduced hardware efficiency. DAWN [25] uses attention-derived dependency graphs for conflict-free updates, but incurs per-step graph construction overhead. KLASS [18] leverages output stability for commitment, but often enforces aggressive, unconstrained parallel decoding leading to accuracy degradation. Overall, these approaches are either insufficiently adaptive or incur higher per-step latencies (Figure 1) to improve decoding parallelism.

Our Contributions. Motivated by the limitations of existing techniques, we present **Polestar**, a '*training-free*' dLLM optimization method to jointly address both axes of dLLM inference optimization. We identify and leverage a principled signal: **token representational drift**. We show that drift is an inherent characteristic of dLLM inference (§3, §B) induced by bidirectional attention, rather than merely an error artifact of KV caching [28]. Specifically, Polestar uses drift to answer two key questions: 1) *when and where* should stale KV cache be updated to enable efficient cache reuse in dLLMs, and 2) *when* can tokens be reliably committed to increase decoding parallelism without compromising output quality. Polestar makes the following key contributions:

- **Polestar-Cache (§4.1):** We propose a novel KV-cache refresh scheme that uses token representation drift in each layer's input hidden states to identify stale KV cache positions and perform sparse, sporadic KV updates with minimal overhead.
- **Polestar-Commit (§4.2):** We observe that sharp token-drift spikes correlate with the onset of convergence toward a token's final output representation. We leverage this signal to commit tokens earlier and more reliably than confidence-only decoding, improving decoding parallelism.
- **Polestar System Optimization (§4.3):** We enable high-throughput Polestar execution through CPU offloading and latency hiding, effectively overlapping Polestar-specific operations with the main dLLM forward pass.

Through algorithm–system co-design, Polestar sets a new state of the art on the accuracy–throughput Pareto frontier for dLLM inference (Figure 1). Across mathematics and coding benchmarks (§5) spanning multiple dLLM families, Polestar achieves up to $\mathbf{10.73\%}$ **accuracy improvement** (3.74% on average), up to $\mathbf{3.7\times}$ **higher TPS** ($1.6\times$ on average), and decoding parallelism of $\mathbf{3.67}$ **tokens per forward (TPF)** over existing SoTA baselines.

2 Preliminaries: Inference with dLLMs

Let $\mathcal{V}$ be the token vocabulary augmented with special masking token `[MASK]`. Given a prompt $\mathbf{p} = (p_0, \ldots, p_{L_p-1}) \in \mathcal{V}^{L_p}$, the model generates L_G tokens by iterative denoising over T reverse-indexed steps $t \in \{T, T-1, \ldots, 0\}$. The initial sequence is $\mathbf{y}^{(T)} = (\mathbf{p}, \mathbf{m}_{L_G})$, where $\mathbf{m}_{L_G} \triangleq (\texttt{[MASK]})^{L_G}$.

Denoising. Let $\mathcal{J} \triangleq \{0, 1, \ldots, L_p + L_G - 1\}$ be the full token index set. At each step t, a mask predictor m_θ [29] generates a tentative reconstruction $\hat{\mathbf{y}}^{(t)}$ by greedily selecting the most likely token to be committed at every position:

$$\hat{y}_i^{(t)} = \texttt{argmax}_{v \in \mathcal{V} \setminus \texttt{[MASK]}} m_\theta(v \mid \mathbf{y}^{(t)}, i), \quad \forall i \in \mathcal{J}. \tag{1}$$

Semi-autoregressive decoding. Here, we partition L_G into $\mathcal{B}$ contiguous blocks (indexed by b) of fixed size B. Decoding proceeds sequentially across blocks with tokens decoded in parallel within a block. Let $\mathcal{J}_b \subseteq \mathcal{J}$ denote the index set of the tokens in the current decode 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 $c_i^{(t)}$ [42]. Based on this score the position to unmask 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,

$$y_i^{(t-1)} = \begin{cases} \hat{y}_i^{(t)}, & i \in \mathcal{S}^{(t)}, \\ \texttt{[MASK]}, & i \in \mathcal{M}^{(t)} \setminus \mathcal{S}^{(t)}, \\ y_i^{(t)}, & i \notin \mathcal{J}_b \end{cases} \tag{2}$$

This yields $N^{(t)} \triangleq |\{i \in \mathcal{S}^{(t)} \mid \hat{y}_i^{(t)} \neq \texttt{[EOS]}\}|$ tokens decoded at step t[1]. The denoise-decode cycle is repeated until all positions in $\mathcal{J}_b$ are unmasked, and decoding advances to the next block.

KV Caching in dLLMs. For layer ℓ at step t, let $\mathcal{I} \subseteq \mathcal{J}$ index cached keys and values, denoted $\mathbf{K}_{\mathcal{I}}^{(t,\ell)} = \{\mathbf{K}_{[i]}^{(t,\ell)}\}_{i \in \mathcal{I}}$ 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 at block entry (t_b) 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 current input hidden states $\mathbf{H}_{\mathcal{J}_b}^{(t,\ell)}$ via linear projections [28]. Queries from the active block attend jointly to cached $\mathbf{K}, \mathbf{V}$ and active-block K, V tensors[2]:

$$\mathbf{A}_{\mathcal{J}_b}^{(t,\ell)} = \mathrm{Attn}\Big(Q_{\mathcal{J}_b}^{(t,\ell)},\ \mathbf{K}_{\mathcal{I}}^{(t_b,\ell)} \cup K_{\mathcal{J}_b}^{(t,\ell)},\ \mathbf{V}_{\mathcal{I}}^{(t_b,\ell)} \cup V_{\mathcal{J}_b}^{(t,\ell)}\Big) \tag{3}$$

After $\mathcal{J}_b$ is decoded, the cache is refreshed: KV representations of the completed block are inserted, while those corresponding to the next active block are removed, giving $\mathcal{I} \leftarrow (\mathcal{I} \cup \mathcal{J}_b) \setminus \mathcal{J}_{b+1}$. Polestar adopts semi-autoregressive decoding with KV caching as its baseline inference procedure.

3 Motivational Analyses

Token Representation Drift. In dLLMs, bidirectional attention makes token representations mutually dependent: a new token unmasking influences the context available to all other positions [29] (§B). Therefore, the internal representation (hidden states) of all other tokens are updated even when those tokens are not decoded at a particular step. We term this context-conditioned evolution as *token representation drift*. Contemporary approaches [28, 43] treat drift primarily as a KV-caching error artifact between cached and recomputed KV representations, and estimate it using cosine dissimilarity over a small token subset. However, it may produce a misleading signal when the chosen token subset remains stable while the broader context evolves, leading to accuracy degradation (Figure 1).

[1]We term the average $N^{(t)}$ over all decoding steps as the tokens per forward (TPF).

[2]We refer to non-cached KV as tensors and denote them with non-bold notation.

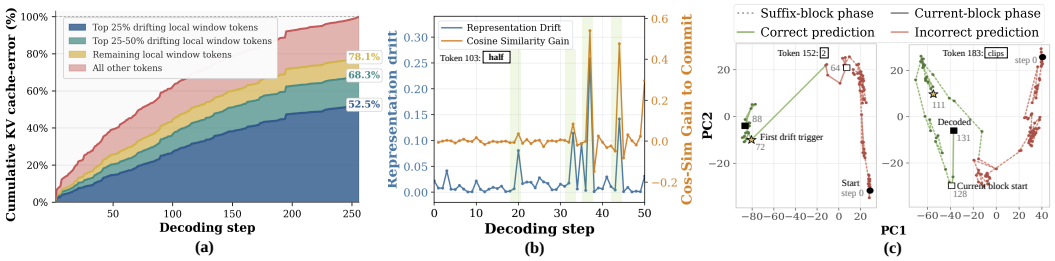


Figure 3: Motivating analyses for Polestar: (a) KV-cache error concentrates on high-drift tokens near the current decoding block; (b) token representations advance toward their final commitment states through sharp drift events; and (c) drift events align with the onset of representational convergence.

Conversely, we quantify drift using KL-divergence, which directly captures how a token's contextual distribution shifts over the full attention support (§5.3, §B). For token position i at layer ℓ and step t, we measure drift as the KL-divergence of its attention distribution between two consecutive steps [3]:

$$D_{i,\ell}^{(t)} = \texttt{KL}\Big(\mathbf{A}_{i,\ell}^{(t)} \,\|\, \mathbf{A}_{i,\ell}^{(t-1)}\Big) \tag{4}$$

3.1 Token Drift Reveals When and Where KV Cache Need Updating

Setup. We use LLaDA-8B-Instruct [29] with generation length of 256 with $B = 32$ [42], for input prompts from GSM8K [10]. The baseline performs full-sequence forward passes with a TPF of 1 at each decoding step without cache reuse.

We first demonstrate that token drift, as quantified in Equation (4), indeed captures contextual adaptation. Figure 4 presents drift as a heatmap for every token position across decoding steps for the given input prompt. Evident from Figure 4, token representation drift is prominent in both the prefix and suffix (masked positions) region of the currently decoded token, demonstrating contextual integration. Notably, we observe higher drift at semantically related positions, which serves as a validation of the KL-div-based drift measurement. For instance, when token *May* is unmasked, we observe higher drift in semantically related tokens (e.g., *April*).

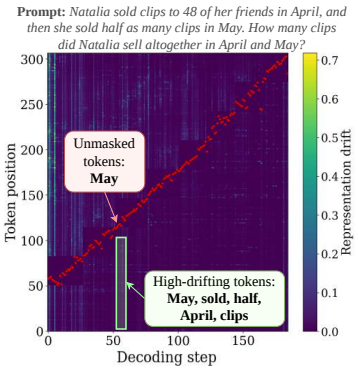


Figure 4: Heatmap depicting drift across token positions and decoding steps, with unmasked tokens at every step shown in red.

We next show how this token representation drift reveals KV-cache staleness for cache-enabled dLLM inference (§2). For the purpose of this experiment, we measure staleness as the average cosine distance (equation in §C.2) between cached KV and their corresponding recomputed KV tensors from full-sequence forward pass. Figure 3(a) presents the cumulative KV-cache error across decoding steps, normalized to the error accumulated over the full generation. For this analysis, we group tokens by both location and drift magnitude. We define a local window around the current decoding block, consisting of the immediately preceding prefix block and the following suffix block. Within this window, we separate the top-25% and top-25–50% highest-drift tokens; the remaining two groups are the least drifting tokens in the local-window and all tokens outside the local window. Figure 3(a) clearly shows that high-drift tokens near the current decoding block contribute disproportionately to the overall KV-cache error. Building on this in §D.7, we further show that local-window drift increases sharply when multiple tokens are decoded in a single forward pass, indicating that KV-cache updates are most needed after substantial contextual integration. ***Polestar-Cache*** *leverages these findings to perform sparse and sporadic KV-cache updates for high-drift tokens in the vicinity of the current decoding block.*

3.2 Token Drift Identifies Opportunities for Early Commitment

Let $z_i^{(t)}$ represents the centered log-probabilities of the output logits for each token position i at step t. $z_i^{(T_i)}$ denotes the predicted state once the token is committed. For the purpose of this analysis, we quantify commitment progress at step t as the increase in cosine similarity to the final commitment state from step $t-1$ to t as, $\Delta\mathcal{C}_i^{(t)} = \texttt{cos_sim}\Big(z_i^{(t)}, z_i^{(T_i)}\Big) - \texttt{cos_sim}\Big(z_i^{(t-1)}, z_i^{(T_i)}\Big)$.

[3] Unless otherwise specified, all subsequent references to drift refer to KL-divergence-based token representation drift.

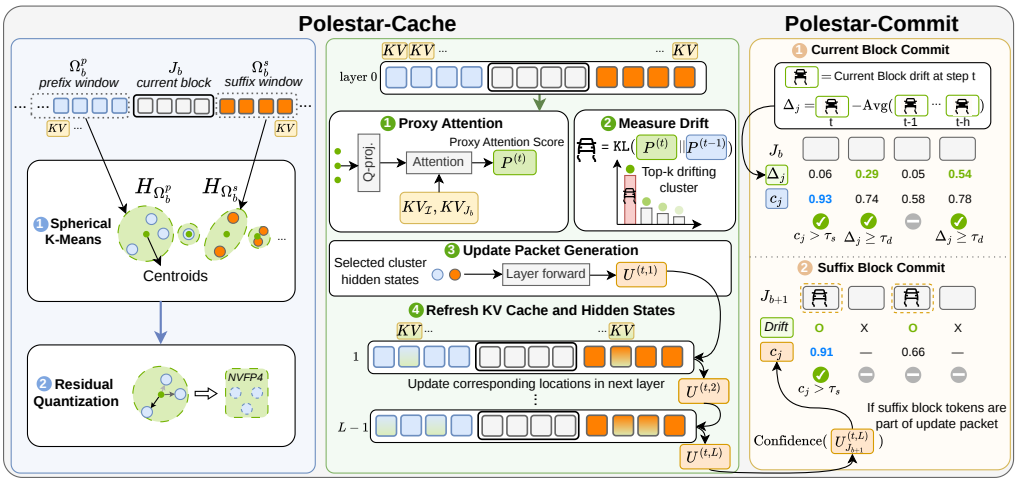


Figure 5: Overview of Polestar. System optimizations omitted for brevity.

Following the same setup in §3.1, we measure the relationship between token representation drift and $\Delta C_i^{(t)}$ for a particular token in Figure 3(b). We observe that progress toward commitment is not gradual, but occurs through sharp drift events. In particular, a token moves substantially closer to its commitment state when its drift rises significantly above its recent drift history in the preceding steps.

We further examine how this reveals opportunities for early commitment of tokens. Following [16], Figure 3(c) visualizes PCA trajectories of last-layer value states across decoding steps for two token positions. We observe that the first sharp drift trigger (yellow star) occurs at the onset of representational convergence, when the token trajectory begins to stabilize toward its decoded state. Notably, for token 183 in Figure 3(c), this occurs while the token is still in the suffix region. Overall, the trajectories show that drift-based commitment enables substantially earlier token commitment than static confidence-based commitment with threshold $\tau_s = 0.9$ (black square) [42]. ***Polestar-Commit*** *exploits this behavior to identify tokens for early commitment both in current and suffix block.*

4 Polestar Methodology

This section presents the Polestar methodology, including its algorithmic design (**Polestar-Cache** and **Polestar-Commit**) and system optimizations. Figure 5 illustrates the overall framework, and Algorithm 1 summarizes the methodology.

4.1 Polestar-Cache

Initialization. At the start of the decoding phase, we perform a full-sequence forward pass to compute and cache the KV representations for all layers, of all token positions outside the current block $\mathcal{J}_b$. Keys and values of the current block are not cached and computed at every step.

Block entry. As observed in §3.1, majority of the KV-cache error is concentrated within a local window around the currently decoding block. Therefore, we maintain a local window $\Omega_b = \Omega_b^p \cup \Omega_b^s$, composed of prefix and suffix windows as shown in Figure 5. At the entry of each block, we cache the input hidden states $\mathbf{H}_{\Omega_b}$ of all layers for all token positions within the local window. Due to residual connections in the dLLM forward pass [29], we cache input hidden states in addition to KV cache to ensure that layer outputs are updated consistently with the evolving contextual representations.

Cluster and Quantize Hidden States. During decoding, measuring drift for all tokens in the local window introduces significant overheads. To mitigate this, we use spherical K-means [36, 13] to cluster $\mathbf{H}_{\Omega_b}$ in the local window. We then employ the computed cluster centroids as sufficient and compact representations (proxies) of the local window tokens for measuring drift. Furthermore, to minimize the memory overhead of caching the hidden states, for each cluster, we quantize the residuals between its hidden states and the corresponding centroid. Each residual is quantized along the channel dimension using NVFP4 [36] (§C.3).

Measuring Drift and Layer-wise KV Updates. At each step t and layer ℓ, we project all cluster centroids into query tokens and compute the "proxy" attention distribution $P^{(t,\ell)}$ using a combination of cached KV and recomputed KV tensors for the active block. Following §3.1, we select the top-k

clusters by drift magnitude to identify token positions requiring updates. We then perform a sparse forward pass [19] only over the hidden states associated with these clusters to obtain updated layer ℓ outputs [4]. As shown in Figure 5, this is layer ℓ's update packet $U^{(t,\ell+1)}$. Layer $\ell+1$ uses this update packet to refresh corresponding stale cached hidden states and subsequently update stale KV-cache entries within the local window. The corresponding centroids are also updated incrementally to reflect the refreshed hidden states. Repeating this process across layers yields sparse layer-wise KV-cache updates that keep the cache aligned with the evolving context without full-sequence recomputation.

Update Schedule. Unlike existing techniques [42, 28] that refresh the KV cache at fixed intervals, Polestar-Cache uses token drift to selectively refresh only when cached states are likely to become stale (§D.7). Polestar-Cache refreshes the cache when newly decoded context is expected to cause substantial misalignment: either after a step that decodes more than τ_{upd} tokens, or when the cumulative number of decoded tokens since the last refresh exceeds τ_{upd}. Additionally, since tokens outside the local window still contribute non-negligible KV-cache error (Figure 3(a)), Polestar-Cache performs a full-sequence forward pass at every alternate block entry to refresh non-local positions.

4.2 Polestar-Commit

Current Block Commit. Existing techniques [42, 28] commit current-block tokens using a static confidence threshold $\tau_s = 0.9$. In contrast, Polestar-Commit performs drift-aware token commitment using final-layer's token drift score, which provides a signal closer to the model's eventual prediction. To detect sharp drift events, we maintain the mean drift over the past h steps and define the drift delta $\Delta_j^{(t)}$ as the deviation of the current drift from this historical mean. A positive $\Delta_j^{(t)}$ indicates that masked position j is undergoing a stronger contextual update than its recent history. We then modulate this signal using a confidence-conditioned dynamic threshold, $\tau_{d,j}^{(t)} = \alpha(\tau_s - c_j^{(t)})^2$, where α is a scaling factor and $c_j^{(t)}$ is the prediction confidence for position j. A current-block token is committed when $\Delta_j^{(t)} \geq \tau_{d,j}^{(t)}$. Thus, tokens with confidence at least τ_s are committed as in prior methods [42] since the dynamic threshold tends to 0. More importantly, lower-confidence tokens can still be committed when they exhibit a sufficiently strong drift event, indicating representational convergence that confidence alone fails to capture. Tokens farther from the static threshold require stronger drift events for commitment and vice versa.

Suffix Block Commit. In §4.1, each layer generates an update packet. The final-layer update packet is not propagated further, since there is no subsequent layer to update. However, we observe that it provides a useful signal for suffix-block commitment: when suffix-window centroids are selected among the top-k drifting clusters, tokens in the suffix region are undergoing strong contextual updates. For each identified high-drift suffix position, we obtain logits from the final-layer update packet and evaluate its confidence. Since suffix tokens are not actively denoised at the current step, their high-drift events primarily arise from contextual integration with semantically related tokens decoded in the current block. We therefore adopt a conservative commitment rule: a suffix token j is committed only if $c_j^{(t)} \geq \tau_s$. Thus, drift determines *which* suffix positions become eligible for commitment, while confidence determines *which* eligible positions are safe to commit.

4.3 Polestar System Optimization

CPU Offloading. As described in §4.1, Polestar-Cache uses cluster centroids to identify the top-k drifting clusters to selectively refresh corresponding KV-cache entries. Because the cached hidden states introduce an additional memory footprint that competes for memory with the KV cache on GPU, Polestar stores only the centroids on GPU and offloads the quantized hidden-state clusters to pinned CPU memory. During decoding, only the top-k drifting clusters are fetched asynchronously from CPU memory. Since the hidden-state clusters are quantized, it significantly reduces CPU-GPU transfer bandwidth. Following [36], the retrieved hidden states' dequantization [35] is fused with update packet computation to reduce overhead.

Latency Hiding. To reduce Polestar's runtime overhead, we leverage CUDA streams to overlap additional operations with the main dLLM forward pass. The main stream executes the current block attention and FFN computation. Since Polestar performs different operations at block entry and during subsequent within-block steps, we optimize the two cases separately. At block entry,

[4] This sparse forward pass is necessary because, under semi-autoregressive decoding, the standard forward pass only processes tokens in the active block (§2).

Table 1: Accuracy (%), tokens-per-forward (TPF) and tokens-per-second (TPS) comparison of Polestar, with SoTA dLLM KV caching baselines. Speedup over baseline is indicated as $(\cdot\times)$.

Benchmark	Gen Len	Method	LLaDA-8B-Instruct			Dream-7B-Instruct		
			Acc(%)↑	TPF↑	TPS↑	Acc(%)↑	TPF↑	TPS↑
GSM8K *(5-shot)*	256	Baseline	79.30	1.00	6.58 (1.0×)	75.85	1.00	9.85 (1.0×)
		Baseline+Parallel	78.57	2.95	17.65 (2.7×)	72.96	1.56	12.10 (1.2×)
		Fast-dLLM	77.88	2.72	47.88 (7.3×)	66.67	1.71	39.86 (4.0×)
		Dynamic-dLLM	78.01	2.79	49.15 (7.5×)	68.16	1.79	41.01 (4.1×)
		Elastic-Cache	77.58	2.93	55.01 (8.4×)	63.63	1.48	31.50 (3.1×)
		Polestar (Ours)	**78.33**	**3.67**	**87.57 (13.3×)**	**72.40**	**2.39**	**52.80 (5.4×)**
	512	Baseline	77.50	1.00	3.93 (1.0×)	76.90	1.00	6.59 (1.0×)
		Baseline+Parallel	77.50	2.85	18.82 (4.8×)	69.95	1.96	15.99 (2.4×)
		Fast-dLLM	76.52	2.58	38.07 (9.7×)	64.55	1.38	25.11 (3.8×)
		Dynamic-dLLM	76.65	2.90	42.79 (10.9×)	65.03	1.47	26.20 (4.0×)
		Elastic-Cache	76.97	2.61	38.44 (9.8×)	58.94	1.13	19.39 (2.9×)
		Polestar (Ours)	**78.18**	**3.59**	**80.60 (20.5×)**	**69.65**	**2.13**	**34.44 (5.2×)**
MATH *(4-shot)*	256	Baseline	33.50	1.00	9.10 (1.0×)	41.69	1.00	10.65 **(1.0×)**
		Baseline+Parallel	33.10	2.51	22.90 (2.5×)	41.30	1.68	25.87 (2.4×)
		Fast-dLLM	30.18	2.30	38.13 (4.2×)	37.19	1.79	49.08 (4.6×)
		Dynamic-dLLM	30.09	2.60	43.24 (4.8×)	37.47	2.04	54.93 (5.2×)
		Elastic-Cache	29.35	2.08	42.56 (4.7×)	30.02	1.73	43.05 (4.0×)
		Polestar (Ours)	**32.77**	**2.91**	**70.05 (7.7×)**	**40.75**	**2.29**	**59.62 (5.6×)**
	512	Baseline	36.60	1.00	8.30 (1.0×)	38.95	1.00	9.80 **(1.0×)**
		Baseline+Parallel	36.15	2.50	19.60 (2.4×)	37.90	2.10	32.80 (3.3×)
		Fast-dLLM	33.20	2.60	50.79 (6.1×)	32.64	2.27	57.79 (5.9×)
		Dynamic-dLLM	32.22	2.79	54.57 (6.6×)	33.88	2.47	61.58 (6.3×)
		Elastic-Cache	30.85	2.17	39.22 (4.7×)	33.17	2.35	54.46 (5.6×)
		Polestar (Ours)	**34.85**	**3.30**	**69.84 (8.4×)**	**35.71**	**2.81**	**66.21 (6.8×)**
HumanEval *(0-shot)*	256	Baseline	43.35	1.00	17.60 (1.0×)	58.80	1.00	23.30 **(1.0×)**
		Baseline+Parallel	43.90	3.11	56.41 (3.2×)	58.80	2.10	46.97 (2.0×)
		Fast-dLLM	38.63	2.64	54.19 (3.1×)	48.48	1.36	40.45 (1.7×)
		Dynamic-dLLM	38.84	3.11	63.79 (3.6×)	48.00	1.45	42.21 (1.8×)
		Elastic-Cache	41.67	3.08	72.75 (4.1×)	55.30	1.27	33.58 (1.4×)
		Polestar (Ours)	**42.42**	**3.38**	**87.45 (5.0×)**	**57.69**	**1.79**	**55.18 (2.4×)**
	512	Baseline	44.10	1.00	8.57 (1.0×)	58.80	1.00	15.48 **(1.0×)**
		Baseline+Parallel	44.10	2.71	34.53 (4.0×)	56.95	1.93	26.77 (1.7×)
		Fast-dLLM	44.24	2.59	58.87 (6.9×)	51.51	1.23	32.21 (2.1×)
		Dynamic-dLLM	44.48	2.49	56.74 (6.6×)	51.45	1.42	36.55 (2.4×)
		Elastic-Cache	45.45	2.74	59.36 (6.9×)	53.03	1.01	23.14 (1.5×)
		Polestar (Ours)	**47.73**	**3.23**	**77.67 (9.1×)**	**54.36**	**1.77**	**41.68 (2.7×)**

auxiliary streams perform hidden-state caching, clustering, quantization, and offloading in parallel with the main attention computation. During within-block steps, proxy-attention computation, drift measurement, and update-packet generation are similarly executed on auxiliary streams. Since the token positions selected in the current layer and those that are refreshed via the update packet in the subsequent layer are identical, Polestar prefetches the corresponding clusters for both the current and next layers from CPU memory, allowing subsequent-layer refresh to proceed without stalling. Figure 12(b,c) in §D.10 illustrates Polestar's optimized execution flow.

5 Experimental Evaluations

5.1 Experimental Setup

Models and Datasets. We evaluate Polestar on LLaDA-8B-Instruct [29], LLaDA-1.5 [50], Dream-7B-Instruct [44], and multimodal LLaDA-V [45]. Text-only evaluation covers 5-shot GSM8K [10], 4-shot MATH [12], ParallelBench [17], 0-shot HumanEval (pass@1) [7], and 3-shot MBPP (pass@1) [3]. For LLaDA-V, we evaluate on MathVista [24] and MathVerse [48].

Hyperparameters. We set block size $B = 32$, static confidence threshold $\tau_s = 0.9$ [42], and local window $|\Omega_b| = 3B$, with prefix window $|\Omega_b^p| = 2B$ and suffix window $|\Omega_b^s| = B$. For K-means clustering, we use $K = 8$ and select top-$k = 4$ drifting clusters. We set the update threshold to $\tau_{\text{upd}} = 3$ tokens. For current-block commitment, we use drift history length $h = 5$ and dynamic-threshold scaling factor $\alpha = 10$ to bring confidence values to the same numeric scale as drift.

Baselines. We compare Polestar against methods targeting per-step latency reduction via KV caching and methods targeting increased decoding parallelism. KV-caching baselines include Fast-

dLLM [42], Dynamic-dLLM [43], Elastic-Cache [28], d^{2}Cache [16], and EntropyCache [9]. Parallel-decoding baselines include baseline model with confidence-based parallel decoding, Fast-dLLM [42], AdaBlock-dLLM [23], DAWN [25], and KLASS [18].

Evaluation Setup. All experiments use a single NVIDIA A100 80GB GPU or an NVIDIA GH200 Superchip. Unless otherwise specified, throughput measurements are reported on A100; please see §D.6 for GH200. Following [42, 28], we evaluate generation lengths of 256 and 512 with batch size 1. All models and methods are evaluated using lm-eval [5] for consistency (see §D.1 for details).

5.2 Main Results

Comparison with KV Caching Baselines. In Table 1, we compare Polestar against several SoTA dLLM KV caching baselines across different models, benchmarks and generation lengths. Evidently, **Polestar** achieves the **best overall accuracy–throughput trade-off, improving accuracy by up to 10.71%** over SoTA baselines, while delivering **up to** 20.5× **speedup** over baseline model. Polestar's token-representation-drift-based scheme keeps the KV cache refreshed through selective updates while reliably identifying commit-ready tokens, which translates into high TPF of up to 3.67. Fast-dLLM [42], which assumes that cached KV states remain static while a block is active, does not account for drift induced by newly decoded tokens, causing consistently lower accuracy and reduced TPF due to degraded prediction confidence. Similarly, Elastic-Cache [28] and Dynamic-dLLM [43] are prone to missing broader contextual shifts or triggering unnecessary refreshes. This leads to a weaker accuracy–throughput trade-off than Polestar.

Table 2: Polestar compared against SoTA dLLM parallel decoding baselines on LLaDA-1.5.

Benchmark	Gen Len	Method	Acc(%)↑	TPF↑	TPS↑
GSM8K *(5-shot)*	256	Baseline	81.36	1.00	5.58 (1.0×)
		Baseline+Parallel	80.95	1.95	32.49 (5.8×)
		DAWN	80.82	2.10	43.24 (7.7×)
		KLASS	77.63	1.58	22.63 (4.1×)
		Polestar (Ours)	**81.06**	**3.40**	**79.65 (14.3×)**
MBPP *(3-shot)*	256	Baseline	39.10	1.00	3.45 (1.0×)
		Baseline+Parallel	38.90	1.39	20.40 (5.9×)
		DAWN	37.60	1.51	27.80 (8.1×)
		KLASS	30.00	1.28	13.25 (3.8×)
		Polestar (Ours)	**39.34**	**2.10**	**48.66 (14.1×)**

Importantly, in several settings (e.g., GSM8K-512 and HumanEval-512), **Polestar improves accuracy over the baseline by up to** 3.63% while also improving throughput. We attribute this to **Polestar-Commit**, which uses token representation drift to identify commit-ready tokens more reliably than the baseline, where a fixed number of tokens are forced to be committed.

Comparison with Parallel-Decoding Baselines. Table 2 compares Polestar against state-of-the-art parallel-decoding baselines, DAWN [25] and KLASS [18], on LLaDA-1.5. Polestar improves accuracy by up to 9.34%, achieves TPF up to 3.40, and delivers up to 3.69× higher speedup over these methods. DAWN [25] achieves substantially lower TPS due to the overhead of constructing pairwise token dependencies during decoding and the absence of KV caching. KLASS [18] suffers larger accuracy degradation because its aggressive decoding policy can commit tokens across arbitrary positions. In contrast, Polestar uses a drift-guided commitment policy localized to the current and suffix block, enabling more reliable commitment.

Table 3: Accuracy-performance comparison on ParallelBench-*easy* [17] using LLaDA-8B-Instruct.

Method	Acc(%)↑	TPF↑	TPS↑
Baseline (LLaDA)	77.85	1.00	10.56 (1.0×)
Baseline+Parallel	76.08	2.31	42.23 (4.0×)
Fast-dLLM	68.09	1.88	49.84 (4.7×)
Elastic-Cache	69.84	1.96	54.12 (5.1×)
d^{2}Cache	70.42	2.05	53.42 (5.1×)
EntropyCache	72.55	2.18	64.88 (6.1×)
Dynamic-dLLM	69.21	1.92	51.45 (4.9×)
Polestar (Ours)	**74.18**	**2.43**	**82.41 (7.8×)**

Accuracy and Performance Comparison on Hard-to-Parallelize Tasks. Table 3 compares Polestar against several baselines on ParallelBench [17], a benchmark designed to stress-test the accuracy–parallelism trade-off in dLLMs. Since ParallelBench uses shorter generations, we set the generation length to 32 and block size to $B = 8$. Table 3 reports results on the *easy* subset; results on the *hard* subset are provided in §D.5. Polestar remains robust on this benchmark, achieving the highest TPF of 2.43 and 82.41 TPS (7.8× over baseline), while maintaining 74.18% accuracy.

Table 4: dInfer integration on LLaDA-8B-Instruct, GH200, generation length 256, block size 32. Speedup is relative to dInfer.

Benchmark	Batch	TPS↑		TPF↑	
		dInfer	**Polestar**	dInfer	**Polestar**
GSM8K	1	320.29	**424.38 (1.325×)**	2.912	**3.866**
	4	338.36	**480.96 (1.420×)**	2.912	**3.866**
	8	379.42	**555.46 (1.464×)**	2.895	**3.640**
HumanEval	1	282.20	**368.13 (1.305×)**	2.540	**3.334**
	4	299.60	**430.49 (1.437×)**	2.147	**3.105**
	8	324.84	**496.78 (1.529×)**	1.986	**2.890**

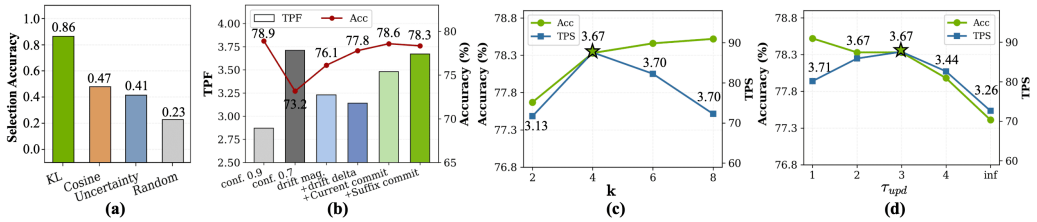


Figure 6: Polestar ablations on LLaDA-8B-Instruct, GSM8K, generation length 256 and block size 32 for (a) drift-policy, (b) commitment-rule, (c) top-k cluster selection and (d) update schedule. For (c,d) star marks the default Polestar setting, and text annotations indicate TPF.

Integration with dInfer. We integrate Polestar into dInfer [27] and evaluate LLaDA-8B-Instruct on GSM8K and HumanEval using GH200, generation length 256, block size 32, and batch sizes 1, 4, and 8. Against dInfer, Polestar achieves 1.305–1.529× higher throughput and consistently higher TPF (Table 4). Speedup increases from 1.325× to 1.464× on GSM8K and from 1.305× to 1.529× on HumanEval as batch size grows from 1 to 8.

5.3 Discussions and Ablations

Impact of Polestar Components. Table 5 evaluates the contribution of each Polestar component. Since Polestar-Cache extends Fast-dLLM [42], we use it as the starting point. Polestar-Cache's drift-aware layer-wise KV refresh policy improves accuracy by 1.01%, increases TPF by 0.15 (reinforces prediction confidence), and improves throughput by 1.26×. The throughput gain comes from fewer full-sequence refreshes and system optimizations (see §D.10). Furthermore, incorporating current-block commitment strategy of Polestar-Commit raises TPF from 2.87 to 3.48 with a negligible drop in accuracy and 1.72× throughput improvement over Fast-dLLM. Finally, suffix-block commitment yields Polestar's complete result of 78.33% accuracy, 3.67 TPF, and 1.83× higher TPS.

Table 5: Component ablation on LLaDA-8B-Instruct, GSM8K, generation length 256.

Variant	Acc.(%)↑	TPF↑	TPS↑
Baseline	79.30	1.00	6.58 (1.0×)
Baseline+Parallel	78.57	2.95	17.65 (2.7×)
Fast-dLLM	77.88 (+0.00%)	2.72	47.88 (7.3×)
Polestar-Cache	78.89 (+1.01%)	2.87 (+0.15)	60.54 (9.2×)
Polestar-Commit			
+ Current-Block Commit	78.57 (+0.69%)	3.48 (+0.61)	82.14 (12.5×)
+ Suffix-Block Commit	**78.33** (+0.45%)	**3.67** (+0.19)	**87.57** (13.3×)

Cache Fidelity under Iso-Commitment. We isolate cache error by replaying a reference trajectory generated with full recomputation on 100 random GSM8K prompts (4-shot), generation length 256, and block size 32. Polestar-Commit is disabled. At each step, every policy receives the same partially denoised sequence and follows the reference token predictions, unmasking decisions, and block transitions; its own predictions never affect subsequent steps. Thus, policies differ only in cache reuse and refresh. Table 6 compares each policy's output distributions with full recomputation on the same input: KL measures distributional divergence, while top-1 mismatch is the fraction of unresolved positions with a different most-likely token. Polestar-Cache reduces mismatch from 16.62% to 8.01% versus no refresh.

Table 6: Cache fidelity under fixed decoding decisions. Refreshed tokens: mean number of cached positions recomputed per layer and denoising step. Random refresh and Polestar use identical budgets.

Cache policy	Refresh rate	Refreshed tokens	Output KL ↓	Top-1 mismatch ↓
Full recomputation	100%	All	0	0
Fast-dLLM (no refresh)	0%	0	0.0879	16.62%
Random refresh	44.20%	18.91	0.0815	16.32%
Polestar-Cache	44.20%	18.91	**0.0570**	**8.01%**

Drift Measurement Policy. We evaluate Polestar's KL-based drift policy for identifying stale KV-cache positions against most-attended-token cosine similarity [28], uncertainty [16], and random selection, using the same cached trajectory and refresh budget within Polestar's local window. Each policy selects the top-25% token positions per layer, and we measure selection accuracy against the true top-25% most-stale KV-cache positions. For a fair comparison with Elastic-Cache, we compute the cosine-similarity change of each candidate position's most-attended token and use it to rank positions for refresh. As shown in Figure 6(a), Polestar's KL-based drift achieves substantially higher selection accuracy (86%), indicating that it better identifies KV-cache positions requiring refresh.

Commitment Rule. For this experiment, we study the impact of different commitment rules, applied on top of Polestar-Cache. As shown in Figure 6(b), static confidence thresholding with $\tau_s = 0.9$

is accurate but conservative, committing only 2.87 tokens per forward. Lowering the threshold to $\tau_s = 0.7$ increases TPF to 3.71 but causes a large accuracy drop, showing that confidence-only scaling is suboptimal. We therefore fix $\tau_s = 0.9$ and next evaluate commitment based on raw drift magnitude and drift delta, where drift delta is measured relative to the mean drift over the past h steps. While raw drift improves over the lowered confidence threshold ($\tau_s = 0.7$), drift delta performs better by capturing the desired sharp drift events rather than absolute drift magnitude alone. Polestar-Commit's current block commit policy improves accuracy to 78.6% at 3.48 TPF by employing a confidence-conditioned dynamic threshold. Finally, suffix commitment increases TPF to 3.67 with similar accuracy, showing that Polestar-Commit improves decoding parallelism without the collapse caused by naive confidence-threshold lowering.

Early-Commit Token and Answer Agreement. To measure how early commitment changes model outputs, we evaluate Dream-7B-Instruct on all 500 MATH-500 prompts with generation length 512 and block size 32. We first add current-block commitment to Polestar-Cache, then add suffix-block commitment. Each row in Table 7 compares the completed outputs before and after adding that mechanism. We also count the tokens committed early across all prompts. Final-token agreement is the fraction of output positions retaining the same token; final-answer agreement is the fraction of prompts retaining the same answer. Current-block and suffix-block commitment preserve 90.2% and 90.6% of final answers, respectively. Each changes 24 previously correct answers to incorrect ones, while correcting 25 and 23 previously incorrect answers, respectively. Both commitment mechanisms therefore enable earlier decoding while preserving over 90% of final answer agreement.

Table 7: Early-commit agreement with each parent trajectory on MATH-500. Effective commits count early-committed tokens.

Added component	Effective commits	Final-token agreement	Final-answer agreement	Correct → Wrong	Wrong → Correct
+ Current-block	31,333	92.97%	90.2%	24 (4.8%)	25 (5.0%)
+ Suffix-block	5,389	98.51%	90.6%	24 (4.8%)	23 (4.6%)

# of top-k Clusters. We vary the number of refreshed clusters while fixing the total number of centroids to $K = 8$. As shown in Figure 6(c), refreshing only the top-2 clusters overlooks token positions that need updates, reaching only 77.67% accuracy and 3.13 TPF. Increasing the refresh budget improves accuracy but introduces additional computation overhead, reducing throughput for top-6 and top-8. The top-$k = 4$ budget achieves 78.33% accuracy, 3.67 TPF, and TPS of 87.57, yielding the best tradeoff between accuracy and performance.

Update Schedule τ_{upd}. We vary the selective refresh threshold τ_{upd} in Figure 6(d). Lower thresholds trigger frequent refreshes, preserving accuracy but wasting computation. In contrast, disabling refresh with $\tau_{\text{upd}} = \texttt{inf}$ allows stale states to accumulate, reducing both accuracy and TPF. The $\tau_{\text{upd}} = 3$ maintains 78.33% accuracy and 3.67 TPF while achieving the highest TPS.

# of centroids K. As shown in Figure 7(a), $K = 8$ provides the best balance between centroid representation granularity and the overhead of computing proxy attention and drift over additional centroids.

Local window size Ω_b. In Figure 7(b), we ablate different prefix and suffix window sizes of the local window. We observe that expanding the local window, particularly on the prefix side, improves accuracy, TPF, and TPS, since prefix tokens are more impacted by contextual integration than distant masked suffix tokens [42]. We therefore adopt $(|\Omega_b^p|, |\Omega_b^s|) = (2B, B)$.

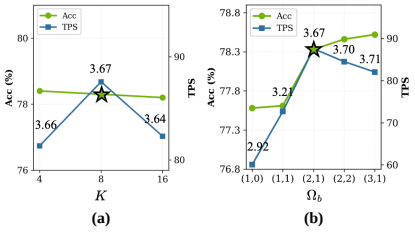


Figure 7: (a) Number of centroids K. (b) (Prefix,Suffix) local-window size Ω_b.

6 Conclusions

We introduce Polestar, a drift-aware inference framework that jointly addresses KV-cache staleness to enable efficient reuse and unreliable parallel token commitment in dLLMs. Polestar leverages token representation drift to selectively refresh stale KV-cache positions and to commit stable tokens earlier in both current and suffix blocks, while system optimizations reduce the overhead of Polestar-specific operations. Across mathematics, coding, and hard-to-parallelize benchmarks, Polestar establishes a new state of the art on the accuracy–throughput Pareto frontier, achieving higher accuracy, throughput, and decoding parallelism than existing dLLM inference baselines.

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

[1] Arriola, M., Gokaslan, A., Chiu, J. T., Yang, Z., Qi, Z., Han, J., Sahoo, S. S., and Kuleshov, V. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.

[2] Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and Van Den Berg, R. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

[3] Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

[4] Bao, W., Chen, Z., Xu, D., and Shang, Y. Learning to parallel: Accelerating diffusion large language models via learnable parallel decoding. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=bFJ8Sdr224.

[5] Biderman, S., Schoelkopf, H., Sutawika, L., Gao, L., Tow, J., Abbasi, B., Aji, A. F., Ammanamanchi, P. S., Black, S., Clive, J., DiPofi, A., Etxaniz, J., Fattori, B., Forde, J. Z., Foster, C., Hsu, J., Jaiswal, M., Lee, W. Y., Li, H., Lovering, C., Muennighoff, N., Pavlick, E., Phang, J., Skowron, A., Tan, S., Tang, X., Wang, K. A., Winata, G. I., Yvon, F., and Zou, A. Lessons from the trenches on reproducible evaluation of language models, 2024. URL https://arxiv.org/abs/2405.14782.

[6] Chen, K., Liu, Z., Tao, X., Liu, H., Fu, X., Zhang, S., Tu, D., Kong, L., Liu, R., and Li, H. Beyond confidence: Adaptive and coherent decoding for diffusion language models, 2025. URL https://arxiv.org/abs/2512.02044.

[7] Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

[8] Chen, X., Huang, S., Guo, C., Wei, C., He, Y., Zhang, J., Li, H. H., and Chen, Y. Dpad: Efficient diffusion language models with suffix dropout, 2025. URL https://arxiv.org/abs/2508.14148.

[9] Cheong, M., Son, D., Lim, W., and Yoo, S. Entropycache: Decoded token entropy guided kv caching for diffusion language models. *arXiv preprint arXiv:2603.18489*, 2026.

[10] Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[11] Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[12] Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

[13] Hooper, C., Zhao, S., Manolache, L., Kim, S., Mahoney, M. W., Shao, Y. S., Keutzer, K., and Gholami, A. Multipole attention for efficient long context reasoning. *arXiv preprint arXiv:2506.13059*, 2025.

[14] Hu, Z., Meng, J., Akhauri, Y., Abdelfattah, M. S., sun Seo, J., Zhang, Z., and Gupta, U. Flashdlm: Accelerating diffusion language model inference via efficient kv caching and guided diffusion, 2025. URL https://arxiv.org/abs/2505.21467.

[15] Israel, D., den Broeck, G. V., and Grover, A. Accelerating diffusion llms via adaptive parallel decoding, 2025. URL https://arxiv.org/abs/2506.00413.

[16] Jiang, Y., Cai, Y., Luo, X., Fu, J., Wang, J., Liu, C., and Yang, X. d2cache: Accelerating diffusion-based llms via dual adaptive caching. *arXiv preprint arXiv:2509.23094*, 2025.

[17] Kang, W., Galim, K., Oh, S., Lee, M., Zeng, Y., Zhang, S., Hooper, C., Hu, Y., Koo, H. I., Cho, N. I., et al. Parallelbench: Understanding the trade-offs of parallel decoding in diffusion llms. *arXiv preprint arXiv:2510.04767*, 2025.

[18] Kim, S. H., Hong, S., Jung, H., Park, Y., and Yun, S.-Y. Klass: Kl-guided fast inference in masked diffusion models. *arXiv preprint arXiv:2511.05664*, 2025.

[19] Lee, M., Ramachandran, A., and Krishna, T. Recap: Training-free compensation for coarse activation channel pruning in compressed llms.

[20] Li, P., Zhou, Y., Muhtar, D., Yin, L., Yan, S., Shen, L., Liang, Y., Vosoughi, S., and Liu, S. Diffusion language models know the answer before decoding, 2025. URL https://arxiv.org/abs/2508.19982.

[21] Liu, J., Dong, X., Ye, Z., Mehta, R., Fu, Y., Singh, V., Kautz, J., Zhang, C., and Molchanov, P. Tidar: Think in diffusion, talk in autoregression, 2025. URL https://arxiv.org/abs/2511.08923.

[22] Liu, Z., Yang, Y., Zhang, Y., Chen, J., Zou, C., Wei, Q., Wang, S., and Zhang, L. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025.

[23] Lu, G., Chen, H. M., Karashima, Y., Wang, Z., Fujiki, D., and Fan, H. Adablock-dllm: Semantic-aware diffusion llm inference via adaptive block size. *arXiv preprint arXiv:2509.26432*, 2025.

[24] Lu, P., Bansal, H., Xia, T., Liu, J., Li, C., Hajishirzi, H., Cheng, H., Chang, K.-W., Galley, M., and Gao, J. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts, 2024. URL https://arxiv.org/abs/2310.02255.

[25] Luo, L., Shi, Z., Luo, J., Wang, Z., Ren, S., Wang, W., and Zhang, T. Dawn: Dependency-aware fast inference for diffusion llms. *arXiv preprint arXiv:2602.06953*, 2026.

[26] Ma, X., Yu, R., Fang, G., and Wang, X. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025.

[27] Ma, Y., Du, L., Wei, L., Chen, K., Xu, Q., Wang, K., Feng, G., Lu, G., Liu, L., Qi, X., Zhang, X., Tao, Z., Feng, H., Jiang, Z., Xu, Y., Huang, Z., Zhuang, Y., Xu, H., Hu, J., Lan, Z., Zhao, J., Li, J., and Zheng, D. dinfer: An efficient inference framework for diffusion language models, 2025. URL https://arxiv.org/abs/2510.08666.

[28] Nguyen-Tri, Q., Ranjan, M., and Shen, Z. Attention is all you need for kv cache in diffusion llms. *arXiv preprint arXiv:2510.14973*, 2025.

[29] Nie, S., Zhu, F., You, Z., Zhang, X., Ou, J., Hu, J., Zhou, J., Lin, Y., Wen, J.-R., and Li, C. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

[30] Pan, T., Gong, B., Guang, M., Yong, H., Jiang, T., Li, Y., Cao, Z., and Long, K. Blockspec: Blockwise speculative decoding for diffusion LLMs, 2026. URL https://openreview.net/forum?id=hmAviop5rm.

[31] Qi, X., Du, L., Zhang, X., Wei, L., Jin, T., and Zheng, D. Hierarchy decoding: A training-free parallel decoding strategy for diffusion large language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ZsIQUjQtdW. under review.

[32] Ramachandran, A., Kundu, S., and Krishna, T. Clamp-vit: Contrastive data-free learning for adaptive post-training quantization of vits. In *European Conference on Computer Vision*, pp. 307–325. Springer, 2024.

[33] Ramachandran, A., Kundu, S., and Krishna, T. Microscopiq: Accelerating foundational models through outlier-aware microscaling quantization. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 1193–1209, 2025.

[34] Ramachandran, A., Kundu, S., Raha, A., Kundu, S., Mathaikutty, D. K., and Krishna, T. Accelerating llm inference with flexible n: M sparsity via a fully digital compute-in-memory accelerator. In *2025 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–7. IEEE, 2025.

[35] Ramachandran, A., Lee, M., Xu, H., Kundu, S., and Krishna, T. Ouromamba: A data-free quantization framework for vision mamba. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 21177–21186. IEEE, 2025.

[36] Ramachandran, A., Neseem, M., Sakr, C., Venkatesan, R., Khailany, B., and Krishna, T. Thinkv: Thought-adaptive kv cache compression for efficient reasoning models. *arXiv preprint arXiv:2510.01290*, 2025.

[37] Shen, J., Sarkar, G., Ro, Y., Sridhar, S. N., Wang, Z., Akella, A., and Kundu, S. Improving the throughput of diffusion-based large language models via a training-free confidence-aware calibration. *ACL*, 2026.

[38] Song, Y., Liu, X., Li, R., Liu, Z., Huang, Z., Guo, Q., He, Z., and Qiu, X. Sparse-dllm: Accelerating diffusion llms with dynamic cache eviction, 2025. URL https://arxiv.org/abs/2508.02558.

[39] Tian, J., Azizi, S., Zhao, Y., Potraghloo, E. B., McPherson, S., Sridhar, S. N., Wang, Z., Zhang, Z., Pedram, M., and Kundu, S. Skipkv: Selective skipping of kv generation and storage for efficient inference with large reasoning models. *MLSys*, 2026.

[40] Wei, Q., Zhang, Y., Liu, Z., Liu, D., and Zhang, L. Accelerating diffusion large language models with slowfast: The three golden principles. *arXiv preprint arXiv:2506.10848*, 2025.

[41] Wei, Q., Zhang, Y., Liu, Z., Liu, D., and Zhang, L. Accelerating diffusion large language models with slowfast sampling: The three golden principles, 2025. URL https://arxiv.org/abs/2506.10848.

[42] Wu, C., Zhang, H., Xue, S., Liu, Z., Diao, S., Zhu, L., Luo, P., Han, S., and Xie, E. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

[43] Wu, T., Sun, X., Jiao, Y., Li, Y., Chen, Y., Cao, Y.-H., Hu, Y.-Q., and Tian, Z. Dynamic-dLLM: Dynamic cache-budget and adaptive parallel decoding for training-free acceleration of diffusion LLM. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=SdnkB5pGbq.

[44] Ye, J., Xie, Z., Zheng, L., Gao, J., Wu, Z., Jiang, X., Li, Z., and Kong, L. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

[45] You, Z., Nie, S., Zhang, X., Hu, J., Zhou, J., Lu, Z., Wen, J.-R., and Li, C. Llada-v: Large language diffusion models with visual instruction tuning, 2025. URL https://arxiv.org/abs/2505.16933.

[46] Yu, R., Ma, X., and Wang, X. Dimple: Discrete diffusion multimodal large language model with parallel decoding, 2025. URL https://arxiv.org/abs/2505.16990.

[47] Yuan, J., Gao, H., Dai, D., Luo, J., Zhao, L., Zhang, Z., Xie, Z., Wei, Y., Wang, L., Xiao, Z., et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*, 2025.

[48] Zhang, R., Jiang, D., Zhang, Y., Lin, H., Guo, Z., Qiu, P., Zhou, A., Lu, P., Chang, K.-W., Gao, P., and Li, H. Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems?, 2024. URL https://arxiv.org/abs/2403.14624.

[49] Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., and Sheng, Y. Sglang: Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

[50] Zhu, F., Wang, R., Nie, S., Zhang, X., Wu, C., Hu, J., Zhou, J., Chen, J., Lin, Y., Wen, J.-R., and Li, C. Llada 1.5: Variance-reduced preference optimization for large language diffusion models, 2025. URL https://arxiv.org/abs/2505.19223.

Appendix

A Extended Related Works **16**

B Theoretical Justification of Representation Drift **16**

C Supplementary Details on Polestar **18**

C.1 Polestar Algorithm 18

C.2 KV-Cache Staleness Metric 18

C.3 Quantization Format 18

D Additional Evaluations **19**

D.1 Extended Experimental Details 19

D.2 Results on MBPP 19

D.3 Comparison with d^2Cache and EntropyCache 19

D.4 Multimodal Evaluation with LLaDA-V 21

D.5 ParallelBench-*Hard* Results 21

D.6 Throughput on GH200 21

D.7 Multi-Token Decoding Induces Representation Drift 21

D.8 Extended Hyperparameter Ablation 21

D.9 Extended Motivational Analysis on HumanEval 23

D.10 Performance Breakdown 23

D.11 Memory Analysis. 24

E Future Works **25**

F Limitations **25**

G Impact Statement **26**

H Declaration of LLM Usage **27**

A Extended Related Works

Beyond confidence-threshold decoding, CCD [6] leverages predictive consistency across previous diffusion steps to rectify sampling trajectories and adapt the unmasking budget, while Prophet [20] predicts future denoising trajectories to decide when the model can confidently commit tokens earlier. Speculative and adaptive decoding methods provide another route: BlockSpec [30] explores multiple future blockwise trajectories in parallel, APD [15] uses a small auxiliary autoregressive model to adaptively choose how many tokens to decode in parallel, and Learn2PD [4] trains a lightweight filter to predict whether each token prediction should remain unmasked. Other methods modify the decoding structure or model itself: AdaBlock-dLLM [23] adjusts block sizes using token uncertainty, Hierarchy-dLLM [31] partitions blocks into smaller sub-blocks for more controlled parallelism, SlowFast [41] alternates between exploratory and accelerated decoding stages, TiDAR [21] combines diffusion-style drafting with autoregressive output sampling, and Dimple [46] introduces a diffusion-trained multimodal LLM with confident decoding. Orthogonal to these parallel-decoding methods, several works reduce per-step cost through cache or attention optimization. FlashDLM [14] combines FreeCache, which reuses stable KV projections, with guided diffusion using a lightweight AR model; dKV-Cache [26] introduces delayed and conditioned KV caching for denoising; EntropyCache [9] uses decoded-token entropy as a constant-cost skip-or-recompute signal; Sparse-dLLM [38] exploits persistent attention saliency to evict low-relevance prefix/suffix cache entries; and DPad [8] drops distant suffix tokens to reduce redundant attention computation.

B Theoretical Justification of Representation Drift

We provide a first-order analysis showing that token representation drift arises from bidirectional attention, even when all hidden states and KV states are recomputed exactly at every denoising step. This analysis is not an end-to-end performance guarantee for Polestar. Instead, it formalizes the claim in §3 that drift reflects contextual integration during dLLM inference, rather than merely approximation error introduced by KV-cache reuse.

Setup. Consider an attention head at a fixed Transformer layer. Let $x_j \in \mathbb{R}^{d_{\text{model}}}$ denote the layer input at token position j, and let

$$q_i = W_Q x_i, \qquad k_j = W_K x_j, \qquad v_j = W_V x_j$$

be the query, key, and value vectors. The attention logit, attention distribution, and attention output for position i are

$$\ell_{ij} = \frac{q_i^\top k_j}{\sqrt{d_h}}, \qquad a_{ij} = \frac{\exp(\ell_{ij})}{\sum_m \exp(\ell_{im})}, \qquad o_i = \sum_j a_{ij} v_j,$$

where d_h is the head dimension. We analyze the local effect of unmasking one or more positions on the attention output of another position. LayerNorm, residual connections, FFN blocks, and multi-layer propagation are omitted from the derivation; they can further propagate the same perturbation, but are not needed to show that bidirectional attention already creates nonzero representation drift under exact recomputation.

Proposition 1. Bidirectional attention induces token representation drift. Suppose that between two consecutive denoising steps, position s is unmasked, changing its layer input from x_s to $x_s + \eta$, while the other layer inputs are fixed to first order. For any position $i \neq s$ that can attend to s, the attention output changes as

$$o_i(x_s + \eta) - o_i(x_s) = a_{is} W_V \eta + a_{is} \frac{q_i^\top W_K \eta}{\sqrt{d_h}} (v_s - o_i) + O(\|\eta\|^2).$$

Therefore, unmasking position s changes the representation of position i even under exact full-sequence recomputation.

Proof. Since $i \neq s$, the query q_i is fixed to first order. Only the key and value at position s change:

$$\delta k_s = W_K \eta, \qquad \delta v_s = W_V \eta.$$

Thus the only first-order logit perturbation is

$$\delta \ell_{is} = \frac{q_i^\top W_K \eta}{\sqrt{d_h}}.$$

The softmax derivative gives

$$\delta a_{ij} = a_{ij} \left(\mathbf{1}\{j = s\} - a_{is} \right) \delta \ell_{is}.$$

Therefore,

$$\begin{aligned}
\delta o_i &= \sum_j \delta a_{ij} v_j + a_{is} \delta v_s \\
&= a_{is} \delta \ell_{is} \left(v_s - \sum_j a_{ij} v_j \right) + a_{is} W_V \eta \\
&= a_{is} \frac{q_i^\top W_K \eta}{\sqrt{d_h}} \left(v_s - o_i \right) + a_{is} W_V \eta.
\end{aligned}$$

This proves the claimed first-order expression. In bidirectional attention, a visible position s lies in the attention support of other positions, so $a_{is} > 0$ under softmax attention. Hence the effect is generically nonzero. In contrast, in causal autoregressive attention, if s is not visible to i, then $a_{is} = 0$ structurally and this first-order effect vanishes. □

Corollary 1. Hidden-state drift leads to stale KV states under cache reuse. Let $\mathbf{H}_i^{(t,\ell)}$ and $\mathbf{H}_i^{(t-1,\ell)}$ denote the exact fully recomputed hidden states at position i and layer ℓ across two consecutive denoising steps, and define

$$\Delta \mathbf{H}_i^{(t,\ell)} = \mathbf{H}_i^{(t,\ell)} - \mathbf{H}_i^{(t-1,\ell)}.$$

The corresponding key and value changes are

$$\Delta K_i^{(t,\ell)} = W_K^{(\ell)} \Delta \mathbf{H}_i^{(t,\ell)}, \qquad \Delta V_i^{(t,\ell)} = W_V^{(\ell)} \Delta \mathbf{H}_i^{(t,\ell)}.$$

Equivalently,

$$\left(\|\Delta K_i^{(t,\ell)}\|_2^2 + \|\Delta V_i^{(t,\ell)}\|_2^2 \right)^{1/2} = \left\| \begin{bmatrix} W_K^{(\ell)} \\ W_V^{(\ell)} \end{bmatrix} \Delta \mathbf{H}_i^{(t,\ell)} \right\|_2.$$

Thus, if

$$\Delta \mathbf{H}_i^{(t,\ell)} \notin \ker(W_K^{(\ell)}) \cap \ker(W_V^{(\ell)}),$$

where, $\ker(A)$ is the set of all vectors in the domain that map to the zero vector, i.e., all x such that $Ax = 0$. Then, at least one of the exact key or value states changes across steps. Consequently, reusing the previous KV state without refresh creates a stale cache entry whenever the hidden-state drift is not wiped out by both projections.

Moreover,

$$\left(\|\Delta K_i^{(t,\ell)}\|_2^2 + \|\Delta V_i^{(t,\ell)}\|_2^2 \right)^{1/2} \leq \left\| \begin{bmatrix} W_K^{(\ell)} \\ W_V^{(\ell)} \end{bmatrix} \right\|_2 \|\Delta \mathbf{H}_i^{(t,\ell)}\|_2.$$

If the combined projection has restricted minimum singular value $\sigma_{\min} > 0$ on the drift subspace, then

$$\left(\|\Delta K_i^{(t,\ell)}\|_2^2 + \|\Delta V_i^{(t,\ell)}\|_2^2 \right)^{1/2} \geq \sigma_{\min} \|\Delta \mathbf{H}_i^{(t,\ell)}\|_2.$$

Therefore, hidden-state drift generally induces KV-state drift, and approximate KV reuse creates staleness by retaining old KV states after the exact recomputed states have changed.

Proposition 2. KL-based drift bounds attention-output change. Let $a = A_i^{(t,\ell)}$ and $a' = A_i^{(t-1,\ell)}$ be the attention distributions of the same token at two consecutive denoising steps. Assume that the value vectors are fixed for this comparison and satisfy $\|v_j\|_2 \le M$ for all j, where $M = \max_j \|v_j\|_2$ over the attention support. Then

$$\left\| \sum_j (a_j - a'_j) v_j \right\|_2 \le M \|a - a'\|_1 \le M \sqrt{2 \,\mathrm{KL}(a \| a')}.$$

Thus, when the KL-based drift between two attention distributions is small, the change in the attention output due solely to attention reweighting is also bounded. Large KL-based drift indicates that the token's attention distribution has shifted over the full support; whether this shift produces large KV mismatch also depends on the value states, which we evaluate empirically in §5.

Proof. The first inequality follows from the triangle inequality:

$$\left\| \sum_j (a_j - a'_j) v_j \right\|_2 \le \sum_j |a_j - a'_j| \, \|v_j\|_2 \le M \|a - a'\|_1.$$

The second inequality follows from Pinsker's inequality,

$$\|a - a'\|_1 \le \sqrt{2 \,\mathrm{KL}(a \| a')}.$$

□

Implication for Polestar. Proposition 1 shows that token representation drift arises due to bidirectional attention, before any KV-cache approximation is introduced. Corollary 1 shows that such drift leads to stale KV states when old KV states are reused without refresh. Proposition 2 explains why the KL-based drift in Equation (4) is tied to changes in the attention distribution over the full support. These results justify using token representation drift as a principled signal, while the empirical analyses in §3 and §5.3 evaluate whether KL-based drift is an effective selector for sparse KV refresh and token commitment.

C Supplementary Details on Polestar

C.1 Polestar Algorithm

Algorithm 1 outlines the Polestar procedure, highlighting its three key contributions. For readability, we omit some superscripts, indexing etc. when clear from context to better emphasize the execution flow.

C.2 KV-Cache Staleness Metric

For token index i at step t and layer ℓ, we measure KV cache staleness by the average cosine distance between cached and fully recomputed key-value representations:

$$E_i^{(t,\ell)} \triangleq 1 - \frac{1}{2}\Big(\texttt{cos_sim}(K_{[i]}^{(t,\ell)}, \mathbf{K}_{[i],\mathrm{ref}}^{(t_b,\ell)}) + \texttt{cos_sim}(V_{[i]}^{(t,\ell)}, \mathbf{V}_{[i],\mathrm{ref}}^{(t_b,\ell)}) \Big).$$

Cosine similarity is computed independently for each attention head and averaged across heads.

C.3 Quantization Format

Polestar stores cached hidden-state residuals using NVFP4. NVFP4 represents each value with 1 sign bit, 2 exponent bits, and 1 mantissa bit, and uses a group-wise FP8 (E4M3) scale factor with group size 16.

Table 8: Default hyperparameters and sweep ranges used for Polestar.

Hyperparameter	**Default**	Sweep range
Block size B	**32**	$\{16, 32, 64\}$
Local window $(\lvert\Omega_b^p\rvert, \lvert\Omega_b^s\rvert)$	**(2B,1B)**	$\{(1,0),(1,1),(2,1),(2,2),(3,1)\}$
Total centroids K	**8**	$\{4, 8, 16\}$
Selected clusters **k**	**4**	$\{2, 4, 6, 8\}$ with $K = 8$
Refresh trigger τ_{upd}	**3**	$\{1, 2, 3, 4, \infty\}$
Commit confidence τ_s	**0.9**	$\{0.7, 0.9\}$
Drift history size **h**	**5**	$\{0, 1, 3, 5, 7\}$
Drift-gate coefficient α	**10**	$\{2.5, 5, 10, 20, 40\}$
Suffix confidence	**0.9**	$\{0.85, 0.90, 0.95\}$
Hidden-state quantization	**Residual NVFP4**	FP16, FP8, INT8, MXFP4, INT2, NVFP4

D Additional Evaluations

D.1 Extended Experimental Details

Implementation. Polestar is implemented in PyTorch by extending a Fast-dLLM-style [42] blockwise parallel decoding and dual-cache inference pipeline. Our full implementation adds (i) Polestar-Cache, drift-aware sparse cache realignment via clustered hidden-state, and (ii) Polestar-Commit, including current-block and suffix-block drift-aware commitment. All system optimizations are implemented in Triton.

Performance Measurement. For all methods, TPF and TPS are measured over the full fixed-length denoising trajectory [32, 40]. We do not stop decoding, truncate the internal sequence, or shorten the remaining generation budget when an `[EOS]` token is first produced. Instead, decoding continues until all response positions in the prescribed generation length are unmasked. This is important for dLLMs because, under bidirectional attention, positions beyond an early `[EOS]` remain part of the denoising context and can influence the representations used to unmask the remaining positions. For throughput accounting, the generated-token count excludes `[EOS]` tokens and prompt tokens. TPF is computed as the number of non-`[EOS]` generated response tokens divided by the number of model forward passes, and TPS uses the same token count divided by the end-to-end wall-clock decoding time.

Hyperparameters. Table 8 summarizes the default hyperparameters used in the main experiments and the sweep ranges used in ablations. Unless otherwise specified, all results use the default configuration. Importantly, Polestar does not require per-model or per-dataset fine-tuning.

D.2 Results on MBPP

Table 9 reports additional MBPP results for LLaDA-8B-Instruct and Dream-7B-Instruct. Polestar achieves the best overall throughput on both models and both generation lengths while maintaining competitive or stronger pass@1 accuracy. On LLaDA-8B-Instruct, Polestar improves accuracy over the full-recomputation baseline at generation length 256 and matches the baseline at generation length 512, while achieving $5.6\times$ and $7.4\times$ speedup, respectively. On Dream-7B-Instruct, Polestar delivers the highest TPS among all methods and remains substantially more accurate than prior cache-reuse baselines, showing that the proposed drift-based cache realignment is also effective on code-generation workloads.

D.3 Comparison with d^{2}Cache and EntropyCache

Table 10 extends the main comparison to d^{2}Cache [16] and EntropyCache [9] across LLaDA-8B-Instruct, LLaDA-1.5, and Dream-7B-Instruct. Evidently, Polestar maintains the strongest accuracy–throughput trade-off across model families and generation lengths. Compared with d^{2}Cache and EntropyCache, Polestar consistently achieves higher TPS and higher TPF while preserving stronger accuracy on reasoning and coding tasks. For instance, on GSM8K with generation length 512, Polestar reaches 80.60 TPS on LLaDA-8B-Instruct and 61.35 TPS on LLaDA-1.5, substantially outperforming both dynamic cache baselines. These results support that drift-aware sparse refresh and commitment remain effective beyond the subset of baselines reported in the main table.

Algorithm 1: Polestar

Input: Prompt $\mathbf{p}$, mask token $[\texttt{MASK}]$, maximum diffusion steps $\mathbf{T}$, # of blocks $\mathcal{B}$, # of layers $\mathbf{L}$, local prefix window size $|\mathbf{\Omega_b^p}|$, local suffix window size $|\mathbf{\Omega_b^s}|$, drift history size $\mathbf{h}$, total # of centroids K, # of selected centroids $\mathbf{k}$, static confidence threshold $\tau_\mathbf{s}$, drift-gate coefficient α

Output: Decoded sequence $\mathbf{y}^{(\mathbf{T})}$

Initialize: $y^{(0)} \leftarrow [\,p;\, [\texttt{MASK}], \ldots, [\texttt{MASK}]\,], \quad t \leftarrow 0$

Cached state: $\mathbf{C} = \{\mathbf{KV}, \mathbf{H}, \mathbf{Z}, \mathbf{P}\} \leftarrow \varnothing$ ▶ $\mathbf{H}$: hidden-state, $\mathbf{Z}$: centroid, $\mathbf{P}$: proxy-attn

$\Theta \leftarrow \varnothing$ ▶ Drift window (size h)

$U^{(t,\ell)} \leftarrow \varnothing$ ▶ Layer update packet

while $t < T$ **do**
  for $b = 0$ **to** $\mathcal{B} - 1$ **do**
    Let $\mathcal{J}_b$ be the token indices of active block b
    Let $\Omega_b = \{j : j \in [b - |\Omega_b^p|,\ b + |\Omega_b^s|],\ j \notin \mathcal{J}_b\}$;
    $E_b \leftarrow \begin{cases} \mathcal{I}, & b\%2 == 0 \\ \Omega_b \cup \mathcal{J}_b, & \text{otherwise} \end{cases}$ ▶ Set block entry context
    Forward on $y_{E_b}^{(t)}$ with cached $\mathbf{KV}_{\bar{E}_b}$
    $\mathbf{Z}_{\Omega_b}, \mathbf{M}_{\Omega_b} \leftarrow \texttt{SphericalKMeans}(\mathbf{H}_{\Omega_b}, K)$ ▶ Obtain centroids and cluster labels
    $\mathbf{H'}_{\Omega_b} \leftarrow \texttt{ResidualQuantization}(\mathbf{H}_{\Omega_b}, \mathbf{Z}_{\Omega_b})$ ▶ $\forall$ layers
    $\mathbf{P} \leftarrow \texttt{ProxyAttn}(\mathbf{Z}_{\Omega_b})$ ▶ $\forall$ layers
    $\mathbf{C} \leftarrow \text{Update}\big(\mathbf{KV}_{E_b \setminus \mathcal{J}_b}, \mathbf{H'}_{\Omega_b}, \mathbf{Z}_{\Omega_b}, \mathbf{P}_{\Omega_b}\big)$ ▶ $\forall$ layers
    $c_{\mathcal{J}_b} \leftarrow \texttt{Confidence}\Big(y_{\mathcal{J}_b}^{(t)} \mid [\texttt{MASK}]\Big)$
    Commit $\{j \in \mathcal{J}_b : c_j > \tau_s,\ y_j^{(t)} == [\texttt{MASK}]\}$
    $t \leftarrow t + 1$
    while $\exists\, j \in \mathcal{J}_b : y_j^{(t)} == [\texttt{MASK}]$ **do**
      for $\ell = 0$ **to** $L - 1$ **do**
        $\mathbf{H}_{\Omega_b}^{(t-1,\ell)} \leftarrow \texttt{Dequantize}(\mathbf{H'}_{\Omega_b}, \mathbf{Z}_{\Omega_b})^{(t-1,\ell)}$
        $(\mathbf{KV}_{\Omega_b}, \mathbf{H}_{\Omega_b}, \mathbf{Z}_{\Omega_b})^{(t,\ell)} \leftarrow \texttt{Patch}\Big((\mathbf{KV}_{\Omega_b}, \mathbf{H}_{\Omega_b}, \mathbf{Z}_{\Omega_b})^{(t-1,\ell)}, \exists U^{(t,\ell)}\Big)$;
        Forward on $y_{\mathcal{J}_b}^{(t)}$ using the patched states; compute main attention scores $\mathbf{A}_{\mathcal{J}_b}^{(t,\ell)}$
        $\mathbf{P}^{(t,\ell)} \leftarrow \texttt{ProxyAttn}(\mathbf{Z}_{\Omega_b}^{(t,\ell)})$;
        $D^{(t,\ell)} \leftarrow \texttt{KL}\Big(\mathbf{P}^{(t,\ell)} \,\|\, \mathbf{P}^{(t-1,\ell)}\Big)$ ▶ Token drift
        $\mathcal{K} \leftarrow \texttt{TopK}(\mathbf{D}^{(t,\ell)}, k)$ ▶ Identify TopK clusters
        $S_{\Omega_b} \leftarrow \mathbf{M}_{\Omega_b}[\mathcal{K}]$ ▶ Token indices corresponding to $\mathcal{K}$
        Forward on $y_{S_{\Omega_b}}^{(t)}$ using the patched states and obtain $U^{(t,\ell+1)}$ ▶ Update packet
        $\mathbf{H'}_{\Omega_b}^{(t,\ell)} \leftarrow \texttt{ResidualQuantization}(\mathbf{H}_{\Omega_b}^{(t,\ell)}, \mathbf{Z}_{\Omega_b}^{(t,\ell)})$
        $\mathbf{C}^{(\ell)} \leftarrow \text{Update}(\mathbf{KV}_{\Omega_b}, \mathbf{H'}_{\Omega_b}, \mathbf{Z}_{\Omega_b}, \mathbf{P})^{(t,\ell)}$ ▶ Update cached states
      end
      $c_{\mathcal{J}_b} \leftarrow \texttt{Confidence}\Big(y_{\mathcal{J}_b}^{(t)} \mid [\texttt{MASK}]\Big)$ ▶ Get confidence of masked tokens in current block
      $\delta^{(t)} \leftarrow \texttt{KL}\Big(\mathbf{A}_{\mathcal{J}_b}^{(t,L-1)} \,\|\, \mathbf{A}_{\mathcal{J}_b}^{(t-1,L-1)}\Big)$ ▶ Compute drift in main attention scores of layer L-1
      $\Delta^{(t)} \leftarrow \delta^{(t)} - \texttt{mean}(\Theta)$
      $\Theta \leftarrow (\Theta \cup \{\boldsymbol{\delta}^{(t)}\})[-h:]$ ▶ Keep recent h
      $\tau_d^{(t)} \leftarrow \alpha(\tau_s - c_{\mathcal{J}_b}^{(t)})^2$ ▶ Dynamic drift threshold
      if $\exists\, j \in \mathcal{J}_b : c_j > \tau_s$ **then**
        Commit $\{j \in \mathcal{J}_b : c_j > \tau_s,\ y_j^{(t)} == [\texttt{MASK}]\}$;
      end
      if $\exists\, j \in \mathcal{J}_b : \Delta_j^{(t)} > \tau_{d,j}^{(t)}$ **then**
        Commit $\{j \in \mathcal{J}_b : \Delta_j^{(t)} > \tau_{d,j}^{(t)},\ y_j^{(t)} == [\texttt{MASK}]\}$
      end
      $c_{\mathcal{J}_{b+1}} \leftarrow \texttt{Confidence}\Big(U_{\mathcal{J}_{b+1}}^{(t,L)} \mid [\texttt{MASK}]\Big)$ ▶ $\mathcal{J}_{b+1}$ is the suffix portion of Ω_b
      Commit $\{j \in \text{Idx}(U_{\mathcal{J}_{b+1}}^{(t,L)}) : c_j > \tau_s, y_{\mathcal{J}_{b+1}}^{(t)} == [\texttt{MASK}]\}$
      $t \leftarrow t + 1$
    end
  end
end

Table 9: LLaDA-8B-Instruct and Dream-7B-Instruct on MBPP. Accuracy is pass@1 (%), values in ± denote accuracy error.

Gen Len	Method	LLaDA-8B-Instruct			Dream-7B-Instruct		
		Acc(%)↑	TPF↑	TPS↑	Acc(%)↑	TPF↑	TPS↑
256	Baseline	28.36±2.02	1.00	9.95 (1.0×)	56.60±2.22	1.00	16.95 **(1.0×)**
	Baseline+Parallel	28.20±2.01	2.76	27.74 (2.8×)	54.87±2.23	1.58	30.55 (1.8×)
	Fast-dLLM	27.51±2.00	2.22	38.63 (3.9×)	52.80±2.23	1.38	33.04 (1.9×)
	Dynamic-dLLM	27.56±1.98	2.29	40.01 (4.0×)	52.21±2.22	1.56	36.71 (2.2×)
	Elastic-Cache	30.40±2.06	2.29	41.07 (4.1×)	45.69±2.21	1.04	20.05 (1.2×)
	Polestar (Ours)	**31.63**±2.08	**2.81**	**55.68 (5.6×)**	**54.91**±2.22	**1.86**	**49.88 (2.9×)**
512	Baseline	14.12±1.56	1.00	6.91 (1.0×)	53.80±2.21	1.00	10.80 **(1.0×)**
	Baseline+Parallel	14.12±1.55	2.73	18.90 (2.7×)	53.10±2.23	1.50	30.38 (2.8×)
	Fast-dLLM	13.71±1.54	2.36	37.70 (5.5×)	50.80±2.24	1.42	28.03 (2.6×)
	Dynamic-dLLM	13.71±1.54	2.95	47.11 (6.8×)	49.70±2.24	1.47	28.44 (2.6×)
	Elastic-Cache	14.16±1.55	2.38	39.13 (5.7×)	39.62±2.19	0.94	14.21 (1.3×)
	Polestar (Ours)	**14.12**±1.56	**3.06**	**51.38 (7.4×)**	**51.36**±2.24	**1.93**	**46.39 (4.3×)**

D.4 Multimodal Evaluation with LLaDA-V

Table 11 evaluates Polestar on multimodal LLaDA-V using MathVista and MathVerse. Polestar achieves the best accuracy, TPF, and TPS on both benchmarks. On MathVista, Polestar reaches 63.58% accuracy and 37.91 TPS, improving over the strongest cache baselines while increasing decoding parallelism to 2.17 TPF. On MathVerse, Polestar achieves 36.12% accuracy and 42.95 TPS, again outperforming both static and dynamic cache-update methods. These results suggest that the same drift signal used for text-only dLLMs transfers to multimodal dLLM inference, where newly decoded textual tokens can also induce representation changes in nearby multimodal context.

D.5 ParallelBench-*Hard* Results

Table 12 reports results on the hard subset of ParallelBench [17], which is designed to stress-test the accuracy–parallelism trade-off. As expected, all cache-enabled and parallel-decoding methods suffer larger accuracy drops than on the easy subset, since premature or overly aggressive commitment is more likely to change the final answer. Nevertheless, Polestar remains the strongest method among the optimized inference baselines, achieving 78.63% accuracy, 1.55 TPF, and 55.34 TPS. Compared with EntropyCache, Polestar improves accuracy by 2.48% and increases TPS by $1.29\times$, indicating that drift-aware commitment is more robust under hard-to-parallelize decoding dynamics.

ParallelBench-*Hard* is intentionally adversarial to parallel decoding because later tokens often depend on unresolved earlier reasoning states. Therefore, all accelerated methods show degradation relative to full recomputation. Polestar reduces, but does not eliminate, this degradation.

D.6 Throughput on GH200

In Table 13, we report throughput measurements on an NVIDIA GH200 GPU.

D.7 Multi-Token Decoding Induces Representation Drift

We validate the update schedule of Polestar-Cache using full-sequence trajectories with confidence-based parallel decoding on LLaDA-8B-Instruct [29]. For each transition, we associate the previous-step decoded-token count $N^{(t-1)}$ with the ensuing local-window drift measured as $\mathrm{KL}(\mathbf{A}^{(t)} \| \mathbf{A}^{(t-1)})$. Figure 8(a) shows that larger previous-step decoded-token counts induce higher subsequent drift, with the strongest increase when $N^{(t-1)} \geq 3$. Figure 8(b) further illustrates that drift spikes align with high decoded-token-count steps in the trajectory. These results support Polestar's decoded-token-count-aware refresh schedule, which reacts to abrupt or accumulated contextual updates rather than recalibrating periodically.

D.8 Extended Hyperparameter Ablation

Block size B. We first vary the block size B while keeping the generation length fixed to 256. As shown in Figure 9(a), smaller blocks incur more block transitions with higher cache recomputation at

Table 10: Extended comparison with d^{2}Cache and EntropyCache across model families.

Benchmark	Gen Len	Method	LLaDA-8B-Instruct			LLaDA-1.5			Dream-7B-Instruct		
			Acc(%)↑	TPF↑	TPS↑	Acc(%)↑	TPF↑	TPS↑	Acc(%)↑	TPF↑	TPS↑
GSM8K *(5-shot)*	256	Baseline	79.30	1.00	6.58 (1.0×)	82.27	1.00	7.47 (1.0×)	75.85	1.00	9.85 (1.0×)
		Baseline+Parallel	78.57	2.95	17.65 (2.7×)	81.82	2.79	19.78 (2.6×)	72.96	1.56	12.10 (1.2×)
		d^{2}Cache	77.91	2.50	46.98 (7.1×)	79.05	2.28	41.52 (5.6×)	70.16	1.45	39.37 (4.0×)
		EntropyCache	77.88	2.79	56.59 (8.6×)	78.81	2.53	49.88 (6.7×)	68.28	1.47	44.29 (4.5×)
		Polestar (Ours)	**78.33**	**3.67**	**87.57 (13.3×)**	**81.06**	**3.40**	**79.65 (10.7×)**	**72.40**	**2.39**	**52.80 (5.4×)**
	512	Baseline	77.50	1.00	3.93 (1.0×)	82.20	1.00	3.66 (1.0×)	76.90	1.00	6.59 (1.0×)
		Baseline+Parallel	77.50	2.85	18.82 (4.8×)	81.06	2.44	15.98 (4.4×)	69.95	1.96	15.99 (2.4×)
		d^{2}Cache	77.04	2.39	38.50 (9.8×)	78.31	1.94	29.07 (7.9×)	67.07	1.44	28.69 (4.4×)
		EntropyCache	76.67	2.57	42.86 (10.9×)	78.16	2.15	32.23 (8.8×)	67.01	1.39	29.16 (4.4×)
		Polestar (Ours)	**78.18**	**3.59**	**80.60 (20.5×)**	**81.06**	**3.70**	**61.35 (16.8×)**	**69.65**	**2.13**	**34.44 (5.2×)**
MATH *(4-shot)*	256	Baseline	33.50	1.00	9.10 (1.0×)	30.88	1.00	6.98 (1.0×)	41.69	1.00	10.65 (1.0×)
		Baseline+Parallel	33.10	2.51	22.90 (2.5×)	28.46	2.25	16.79 (2.4×)	41.30	1.68	25.87 (2.4×)
		d^{2}Cache	30.82	2.13	37.41 (4.1×)	26.51	1.87	37.05 (5.3×)	37.80	1.58	52.72 (5.0×)
		EntropyCache	30.26	2.43	47.73 (5.2×)	26.25	2.14	45.61 (6.5×)	39.69	1.67	57.51 (5.4×)
		Polestar (Ours)	**32.77**	**2.91**	**70.05 (7.7×)**	**27.27**	**2.64**	**70.84 (10.1×)**	**40.75**	**2.29**	**59.62 (5.6×)**
	512	Baseline	36.60	1.00	8.30 (1.0×)	32.65	1.00	6.12 (1.0×)	38.95	1.00	9.80 (1.0×)
		Baseline+Parallel	36.15	2.50	19.60 (2.4×)	30.64	2.49	15.34 (2.5×)	37.90	2.10	32.80 (3.3×)
		d^{2}Cache	34.24	2.39	49.33 (5.9×)	31.60	1.98	36.63 (6.0×)	34.17	2.03	57.37 (5.9×)
		EntropyCache	33.45	2.38	59.90 (7.2×)	31.15	2.29	43.12 (7.0×)	33.22	2.07	59.98 (6.1×)
		Polestar (Ours)	**34.85**	**3.30**	**69.84 (8.4×)**	**31.77**	**2.82**	**64.36 (10.5×)**	**35.71**	**2.81**	**66.21 (6.8×)**
HumanEval *(0-shot)*	256	Baseline	43.35	1.00	17.60 (1.0×)	53.06	1.00	5.55 (1.0×)	58.80	1.00	23.30 (1.0×)
		Baseline+Parallel	43.90	3.11	56.41 (3.2×)	52.65	2.35	13.04 (2.3×)	58.80	2.10	46.97 (2.0×)
		d^{2}Cache	38.36	2.98	64.05 (3.6×)	44.85	2.18	18.41 (3.3×)	51.83	1.45	46.84 (2.0×)
		EntropyCache	40.48	2.46	56.67 (3.2×)	47.35	2.03	16.58 (3.0×)	54.54	1.43	49.45 (2.1×)
		Polestar (Ours)	**42.42**	**3.38**	**87.45 (5.0×)**	**48.78**	**3.58**	**70.07 (12.6×)**	**57.69**	**1.79**	**55.18 (2.4×)**
	512	Baseline	44.10	1.00	8.57 (1.0×)	51.84	1.00	3.53 (1.0×)	58.80	1.00	15.48 (1.0×)
		Baseline+Parallel	44.10	2.71	34.53 (4.0×)	51.21	2.23	7.01 (2.0×)	56.95	1.93	26.77 (1.7×)
		d^{2}Cache	43.05	2.62	63.70 (7.4×)	48.10	2.03	13.92 (3.9×)	50.68	1.31	37.36 (2.4×)
		EntropyCache	47.39	2.23	57.80 (6.7×)	50.15	1.66	11.75 (3.3×)	53.35	1.39	40.27 (2.6×)
		Polestar (Ours)	**47.73**	**3.23**	**77.67 (9.1×)**	**50.12**	**3.28**	**61.95 (17.5×)**	**54.36**	**1.77**	**41.68 (2.7×)**
MBPP *(3-shot)*	256	Baseline	28.36	1.00	9.95 (1.0×)	40.02	1.00	7.87 (1.0×)	56.60	1.00	16.95 (1.0×)
		Baseline+Parallel	28.20	2.76	27.74 (2.8×)	38.44	1.57	12.36 (1.6×)	54.87	1.58	30.55 (1.8×)
		d^{2}Cache	30.54	2.56	50.99 (5.1×)	38.21	1.46	30.91 (3.9×)	50.01	1.67	42.63 (2.5×)
		EntropyCache	27.37	2.80	55.01 (5.5×)	36.42	1.44	32.55 (4.1×)	50.03	1.68	45.78 (2.7×)
		Polestar (Ours)	**31.63**	**2.81**	**55.68 (5.6×)**	**39.34**	**2.10**	**48.66 (6.2×)**	**54.91**	**1.86**	**49.88 (2.9×)**
	512	Baseline	14.12	1.00	6.91 (1.0×)	41.00	1.00	5.15 (1.0×)	53.80	1.00	10.80 (1.0×)
		Baseline+Parallel	14.12	2.73	18.90 (2.7×)	39.33	1.88	9.69 (1.9×)	53.10	1.50	30.38 (2.8×)
		d^{2}Cache	12.29	2.89	49.76 (7.2×)	35.48	1.62	21.65 (4.2×)	46.00	1.93	41.11 (3.8×)
		EntropyCache	13.71	2.70	49.67 (7.2×)	37.12	1.48	20.31 (3.9×)	47.64	1.75	39.64 (3.7×)
		Polestar (Ours)	**14.12**	**3.06**	**51.38 (7.4×)**	**38.00**	**2.17**	**46.98 (9.1×)**	**51.36**	**1.93**	**46.39 (4.3×)**

Table 11: LLaDA-V results on MathVista and MathVerse benchmarks.

Method	MathVista			MathVerse		
	Acc(%)↑	TPF↑	TPS↑	Acc(%)↑	TPF↑	TPS↑
Baseline	60.61	1.00	2.23	34.00	1.00	2.10
Baseline+Parallel	59.40	1.88	4.67	33.20	3.71	3.34
Fast-dLLM [42]	60.92	1.70	27.00	32.82	2.26	32.79
Elastic-Cache [28]	62.54	1.74	27.68	34.76	2.41	35.12
d^{2}Cache [16]	62.80	1.95	23.50	35.60	2.52	26.80
EntropyCache [9]	60.50	1.72	31.20	33.50	2.30	38.40
Dynamic-dLLM [43]	61.80	1.78	29.50	34.20	2.38	36.20
Polestar (Ours)	**63.58**	**2.17**	**37.91**	**36.12**	**2.85**	**42.95**

block entry, resulting in lower throughput. Increasing the block size to $B = 64$ improves TPF and TPS, but substantially degrades accuracy to 75.1%, suggesting that larger blocks cause premature commitment risk. The default $B = 32$ provides the strongest accuracy–throughput trade-off.

Drift history size h. We study the history size **h** used to compute recent-history-relative drift. As shown in Figure 9(b), using raw drift without history normalization ($\mathbf{h} = 0$) is aggressive, yielding high TPF and TPS but reducing accuracy to 76.91%. Increasing **h** stabilizes the drift signal by filtering transient spikes and steadily improves accuracy. The default $\mathbf{h} = 5$ provides a strong accuracy–throughput balance, while $\mathbf{h} = 7$ is more conservative and yields only a marginal accuracy gain at a larger throughput cost. This supports using relative drift spikes rather than absolute drift magnitude for commitment readiness.

Table 12: ParallelBench-*Hard* results. Higher Acc/TPS is better (↑), higher TPF indicates more parallelism (↑).

Method	Acc(%)↑	TPF↑	TPS↑
Baseline (LLaDA)	86.08	1.00	6.59
Baseline+Parallel	84.13	1.47	26.87
Fast-dLLM [42]	72.13	1.33	33.18
Elastic-Cache [28]	73.84	1.36	37.52
d^2Cache [16]	74.62	1.39	36.14
EntropyCache [9]	76.15	1.42	42.85
Dynamic-dLLM [43]	73.21	1.35	36.19
Polestar (Ours)	**78.63**	**1.55**	**55.34**

Table 13: LLaDA-8B-Instruct throughput on GH200 across benchmarks and generation lengths.

Benchmark	Gen Len	Fast-dLLM	Elastic-Cache	**Polestar (Ours)**
GSM8K	256	77.25	85.42	**151.31**
	512	58.07	64.87	**131.22**
MATH	256	57.97	71.40	**105.99**
	512	83.89	59.79	**106.98**
HumanEval	256	89.34	130.18	**134.96**
	512	92.90	102.08	**142.27**

Drift gate coefficient α. We ablate the coefficient α in the confidence-conditioned drift gate. As shown in Figure 9(c), small α values make the gate more aggressive: $\alpha = 2.5$ reaches high TPF but sharply reduces accuracy to 73.4%. Larger values are safer but increasingly conservative, lowering TPF and TPS. The default $\alpha = 10$ lies in a stable accuracy–parallelism region, preserving accuracy while retaining most of the speedup from drift-aware commitment.

Suffix confidence threshold. Finally, we vary the confidence threshold for suffix pre-commitment. As shown in Figure 9(d), lowering the threshold to 0.85 increases TPF but reduces accuracy to 76.4%, indicating unsafe suffix commitment. Raising the threshold to 0.95 is slightly more conservative, preserving accuracy but reducing TPF and TPS. The threshold of 0.9 which is identical to τ_s provides the best balance, supporting our design in which drift determines suffix eligibility and confidence determines safe commitment.

Table 14: Ablation on quantization format.

Quantization Format	Residual Quant.	Accuracy (%)↑
FP16 (Full Precision)	–	78.90
FP8	Yes	78.90
INT8	Yes	78.54
MXFP4	Yes	78.19
INT2	Yes	0.00
NVFP4	No	54.98
NVFP4 (Polestar)	**Yes**	**78.33**

Quantization. We ablate the compression format for Polestar's cached local-window hidden states. As shown in Table 14, FP8 and INT8 remain close to FP16, whereas direct NVFP4 quantization without residuals drops accuracy to 54.98%. Residual quantization recovers NVFP4 accuracy to 78.33%, closely matching the FP16 baseline while using a much lower-precision representation. We therefore use residual-based NVFP4 by default.

D.9 Extended Motivational Analysis on HumanEval

We provide additional motivating analyses on HumanEval in Figure 10 and Figure 11 to demonstrate the generalization of our observations.

D.10 Performance Breakdown

Figure 12 shows Polestar's worst-case per-layer, per-step timing, where all Polestar-specific operations are active. In the unoptimized execution (Figure 12(a)), Polestar-specific operations add 2.71 ms

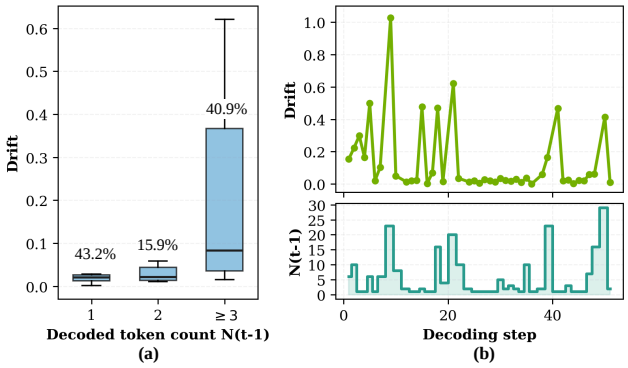


Figure 8: Multi-token decoding induces local representation drift. (a) Local-window drift grouped by the number of tokens decoded in the previous step; percentages above boxes denote the fraction of transitions in each group. (b) A representative trajectory showing that drift spikes align with high decoded-token-count steps.

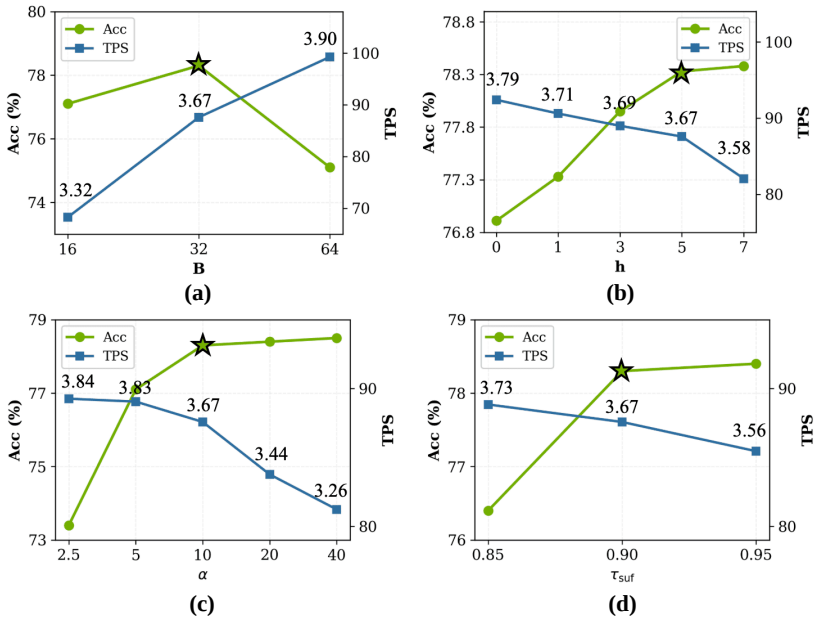


Figure 9: Extended hyperparameter ablation. (a) Block size $\mathcal{B}$. (b) Drift history size $\mathbf{h}$. (c) Current-block drift gate coefficient α. (d) Suffix pre-commit confidence threshold τ_{suf}. The default configuration is marked with a star and text annotations indicate TPF. Across sweeps, Polestar remains stable near its default setting.

of critical-path overhead. Since block entry and within-block decoding involve different operations in Polestar, Figure 12(b) and (c) show the optimized execution for each case. At block entry, the attention and FFN stages are longer due to the full-sequence forward pass at every alternate block entry, but Polestar-specific operations are fully overlapped with the main computation. For subsequent within-block steps, overlap reduces the added overhead to ≤ 0.10 ms. Polestar's efficient prefetching of next layer's hidden states saves up to 0.26ms per-step, per-layer. Overall, the proposed system optimizations reduce per-layer latency by 1.63 ms over the unoptimized case, directly contributing to Polestar's higher throughput.

D.11 Memory Analysis.

We compare the inference-time GPU memory footprint of Polestar, Elastic-Cache [28], and d^{2}Cache [16] during generation on LLaDA-8B-Instruct with GSM8K, generation length 512, and

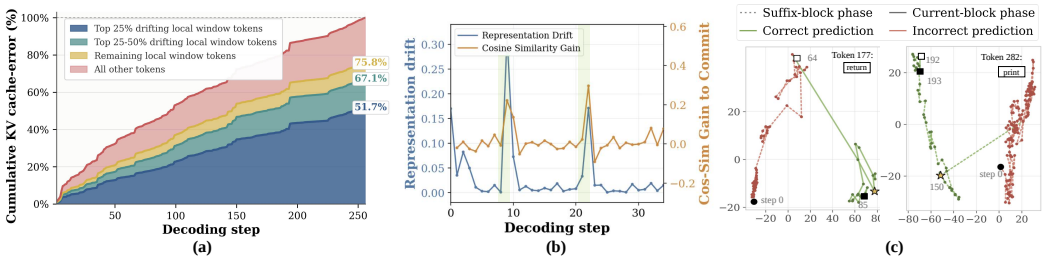


Figure 10: Additional HumanEval motivation analysis for Polestar: (a) KV-cache error concentrates on high-drift states in the local window around the active decoding block; (b) sharp drift events coincide with progress toward the token's final predictive state; (c) PCA trajectories show that the first drift trigger aligns with the onset of stable prediction trajectories, including suffix-block positions before they become active.

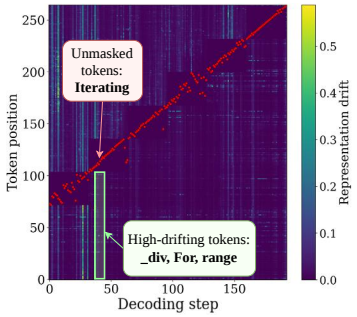


Figure 11: HumanEval drift heatmap depicting measured drift across token positions and decoding steps, with unmasked tokens shown in red. Unmasking a code-relevant token induces drift at related syntactic and semantic positions, illustrating distributed contextual adaptation in code generation.

block size 32. For each method, we load the model first and start tracing allocated GPU memory immediately before decoding. Thus, the reported values include resident model weights, KV-cache buffers, and method-specific runtime buffers, while excluding model-loading time.

As shown in Figure 13, Elastic-Cache reaches the largest peak memory footprint at 19.5 GB. By comparison, d^{2}Cache and Polestar peak at 18.5 GB and 18.6 GB, respectively. Thus, Polestar remains memory-comparable to d^{2}Cache while using 0.9 GB less peak GPU memory than Elastic-Cache. This indicates that Polestar's drift-aware cache calibration does not require a large auxiliary GPU-memory footprint: only compact centroids are kept on GPU, while quantized hidden-state residuals for the local window are offloaded to pinned CPU memory.

E Future Works

A natural next step which we are working towards is to integrate Polestar into production-oriented dLLM serving frameworks, such as SGLang's dLLM runtime [49]. Future work could also extend drift-aware control beyond the fixed block-wise setting studied here. Longer contexts, larger batches, and multimodal dLLMs may require adaptive local windows, dynamic centroid budgets, or runtime-calibrated drift thresholds. More broadly, representation drift may serve as a general control signal for deciding when intermediate states should be refreshed, reused, or committed in scalable diffusion-model serving.

F Limitations

Polestar is an approximate inference method rather than a lossless KV-cache replacement, and is designed for block-wise masked diffusion LLMs. Its gains are strongest when harmful cache staleness is local and sparse, and when centroid-level proxy attention reliably identifies high-drift

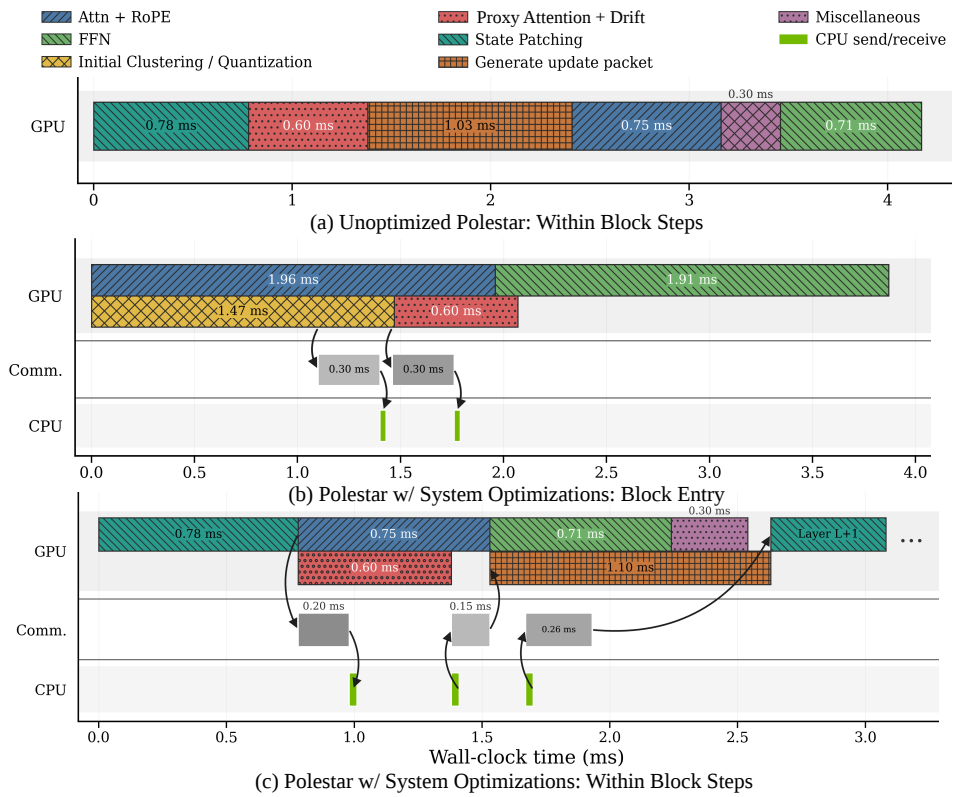


Figure 12: Timing breakdown of Polestar execution. (a) Unoptimized Polestar, (b) Optimized Polestar at block entry and (c) Optimized Polestar at all other steps.

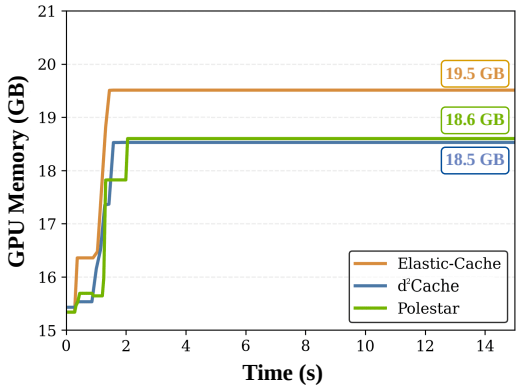


Figure 13: GPU-memory traces comparison during decoding on LLaDA-8B-Instruct with 5-shot GSM8K, generation length 512, block size 32, batch size 1 on an A100 80GB GPU.

states. Extending Polestar to much longer contexts and architectures with more diffuse drift dynamics may require larger refresh windows or additional system optimization.

G Impact Statement

Polestar improves the efficiency of diffusion LLM inference by reducing redundant KV recomputation and increasing decoding parallelism, which can lower serving cost, latency, and energy consumption. Because it is training-free and does not modify model weights, Polestar may make high-throughput generation more accessible to researchers and practitioners with limited compute budgets, and may support practical deployment of diffusion LLMs in latency-sensitive applications. Polestar does

not add new safety mechanisms and therefore inherits the bias, privacy, hallucination, and misuse risks of the underlying dLLMs; deployments should retain the same filtering, monitoring, and misuse-prevention safeguards used for the base models.

H Declaration of LLM Usage

We used general-purpose LLM assistants only for writing, editing, formatting, and presentation support. This included grammar correction, concision, LaTeX polishing, table/caption wording, checklist drafting, and identifying possible clarity issues in the manuscript. LLMs were not used as an important, original, or non-standard component of the proposed method, theoretical claims, experimental design, benchmark evaluation, or reported results. All technical content, mathematical arguments, implementation details, experiments, numerical results, and final claims were written, checked, and approved by the authors.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the paper's scope as a training-free inference method that improves KV-caching and decoding parallelism in dLLMs. The methodology and evaluation sections support these claims through the analysis of token representation drift, Polestar-Cache/Polestar-Commit design, and comparisons against KV-cache and parallel-decoding baselines.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a dedicated discussion of limitations in §F.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We provide the assumptions and first-order proofs showing that representation drift arises from bidirectional attention, can induce KV staleness under cache reuse, and relates KL-based drift to changes in attention redistribution in §B.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides sufficient information to reproduce the main experimental results in §5, §D.1.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We include an anonymized source code link in the abstract with detailed reproduction instructions. We have also included our anonymized code in the supplementary material to enable faithful reproduction of all main experimental results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We describe this information in sufficient detail in §5 and §D.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Most deterministic experiments provide identical results, however for stochastic experiments we discuss standard deviation (for instance, see §D.2).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Yes, for each experiment, we report the compute resources used and provide a detailed execution-time breakdown.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have carefully reviewed the NeurIPS Code of Ethics and ensured that our work is conducted in accordance with its principles.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss this in §G.

    Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This work poses no such risk.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All used assets are properly credited.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not introduce new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not require crowdsourcing experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This research does not involve human participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: This work uses LLMs for the sole purpose of writing, formatting and for reducing redundant content for brevity. We have discussed this in §H.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.