EntropyCache: Decoded Token Entropy Guided KV Caching for Diffusion Language Models

Minsoo Cheong
Seoul National University
icycle0409@snu.ac.kr

Donghyun Son
Stanford University
dhson@stanford.edu

Woosang Lim
Seoul National University
ftyg656512@snu.ac.kr

Sungjoo Yoo*
Seoul National University
sungjoo.yoo@gmail.com

Abstract

Diffusion-based large language models (dLLMs) rely on bidirectional attention, which prevents lossless KV caching and requires a full forward pass at every denoising step. Existing approximate KV caching methods reduce this cost by selectively updating cached states, but their decision overhead scales with context length or model depth. We propose ENTROPYCACHE, a training-free KV caching method that uses the maximum entropy of newly decoded token distributions as a low-cost signal for deciding *when* to recompute. Our design is grounded in two empirical observations: (1) decoded token entropy correlates with KV cache drift, providing a cheap proxy for cache staleness, and (2) feature volatility of decoded tokens persists for multiple steps after unmasking, motivating recomputation of the k most recently decoded tokens. For N newly decoded tokens and vocabulary size V, the skip-or-recompute decision requires $O(NV)$ computation per step, independent of context length and model scale; in practice, N is small and the measured overhead is only 0.5% of inference time. Experiments on LLaDA-8B-Instruct and Dream-7B-Instruct show that ENTROPYCACHE achieves $15.2\times$–$26.4\times$ speedup on standard benchmarks and $22.4\times$–$24.1\times$ on chain-of-thought benchmarks against vanilla baselines, with competitive accuracy.

1 Introduction

Diffusion-based large language models (dLLMs) [1, 16, 12, 25] have emerged as a compelling alternative to autoregressive (AR) models [19], enabling parallel token generation through iterative denoising. Rather than producing tokens sequentially from left to right, dLLMs operate over the entire output sequence simultaneously, progressively replacing mask tokens with predicted values across multiple denoising steps. When combined with semi-autoregressive generation strategies such as blockwise decoding [23] or sliding window decoding [11], dLLMs have shown promising potential to achieve competitive throughput compared to conventional AR generation.

These gains, however, are constrained by a fundamental architectural limitation: dLLMs employ non-causal (bidirectional) attention. In AR models, causal attention [20, 14] guarantees that previously computed key-value (KV) pairs remain the same as new tokens are appended, making KV caching [13] both straightforward and lossless. In dLLMs, every position attends to every other position; consequently, unmasking even a single token alters the representations of *all* positions. Therefore, vanilla dLLM inference executes a full forward pass over the entire sequence at every denoising step, yielding a per-step cost proportional to the context length.

*Corresponding author.

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

To reduce this cost, prior studies [10, 9, 7, 23] observe that the KV states of most tokens change only minimally between consecutive denoising steps, suggesting that selective reuse of cached states can closely approximate exact recomputation. Building on this insight, several approximate KV caching strategies have been proposed. Fast-dLLM [23] takes a static approach by freezing the KV pairs outside the current denoising block throughout its denoising steps, sacrificing cache accuracy in favor of computational efficiency.

On the other hand, some studies propose to dynamically decide where or when to compute. For example, d^2Cache [8] uses attention rollout to selectively update high-influence token positions at each step, focusing on *where* along the token axis to recompute. Elastic-Cache [11] monitors per-layer attention-weight drift and recomputes from a boundary layer onward, indirectly addressing *when* cached states have become stale. However, its detection requires per-layer cosine similarity comparison and attention score recomputation at every step, with decision overhead that scales with model depth, context length, and hidden dimension.

We propose EntropyCache, which provides a cheaper signal for deciding *when* to recompute: the maximum entropy of the newly decoded token distributions. We show that this single scalar correlates with KV cache drift, enabling a skip-or-recompute decision whose overhead is independent of context length and model scale. The design is motivated by two empirical observations:

Observation 1: Unmasked token entropy predicts KV cache drift. We show that the maximum entropy [17] of the token distributions obtained from the previous denoising step correlates with the magnitude of the resulting KV cache drift at the subsequent step. This single scalar therefore serves as a cheap yet effective proxy for cache staleness.

Observation 2: Feature volatility persists for multiple steps after unmasking. Tracking individual token trajectories reveals that some tokens remain unstable for several denoising steps after being unmasked, not just the single step assumed by prior dynamic methods [10, 8]. This motivates recomputing the k most recently unmasked tokens rather than only those from the previous step.

Concretely, EntropyCache works as follows. After each unmasking step, it evaluates the maximum entropy of the newly unmasked token distributions. If this entropy exceeds a threshold τ, a full forward pass is triggered; otherwise, the method reuses cached KV pairs for most positions and recomputes KV pairs only for (a) the current mask tokens and (b) at most k recently unmasked tokens since the previous full recomputation. We make the following contributions:

- We identify maximum unmasked-token entropy as a lightweight, empirically grounded proxy for KV cache staleness, replacing expensive per-layer rollout or hidden-state comparisons with an $O(NV)$ computation over the N newly decoded distributions, independent of context length, model depth, and hidden dimension.
- We provide empirical analysis showing that feature volatility persists for multiple denoising steps after unmasking, and characterize how recency-ranked local recomputation effectively captures this multi-step instability.
- We integrate these findings into EntropyCache, a training-free caching method for dLLMs. Evaluated on LLaDA-8B-Instruct and Dream-7B-Instruct, EntropyCache achieves **15.2**×–**26.4**× speedup on standard benchmarks and **22.4**×–**24.1**× on chain-of-thought benchmarks, consistently outperforming prior caching methods in throughput while maintaining competitive accuracy.

2 Related work

2.1 Diffusion-based large language models

Unlike autoregressive (AR) models that generate tokens left-to-right with causal attention [14, 19], dLLMs denoise the entire sequence simultaneously using bidirectional attention. LLaDA [12] trains a masked diffusion model from scratch and achieves performance competitive with LLaMA3-8B. Dream [25] instead initializes from a pretrained AR checkpoint and introduces context-adaptive noise rescheduling. In both models, unmasking even a single token changes representations at all positions, making standard lossless KV caching inapplicable. Vanilla dLLM inference runs a full forward pass at every denoising step, so cost scales with the product of sequence length and step count. Semi-autoregressive strategies reduce this cost. Blockwise decoding [23] denoises fixed-size

blocks one at a time, while sliding-window decoding [11] advances a window over the sequence and keeps the number of denoised masked tokens constant at each step, better preserving diffusion-level parallelism. Orthogonally, confidence-based parallel decoding [23] unmasks multiple tokens per step when predicted probabilities exceed a threshold. These strategies are complementary to KV caching methods.

2.2 Approximate KV caching for dLLMs

Prior approximate KV caching strategies for dLLMs fall into two categories: static methods that apply a fixed reuse policy across all denoising steps, and dynamic methods that determine where or when to refresh the cache based on intermediate states at each step.

Static caching strategy. Fast-dLLM [23] freezes the KV pairs outside the current denoising block throughout its denoising steps. This is simple and effective but does not dynamically detect cache staleness. The entire cache is refreshed only at block boundaries.

Dynamic caching strategy. d^{2}Cache [8] uses attention rollout to identify high-influence tokens and selectively updates their KV states at each step. Elastic-Cache [11] monitors attention-weight drift at the most-attended token and, upon detecting staleness, recomputes from a boundary layer onward while reusing shallow-layer caches. Both improve upon static caching in accuracy–throughput tradeoff, but their decision overhead scales with context length or model depth, limiting practical speedup.

3 Motivation

3.1 Decoded token entropy predicts KV cache drift

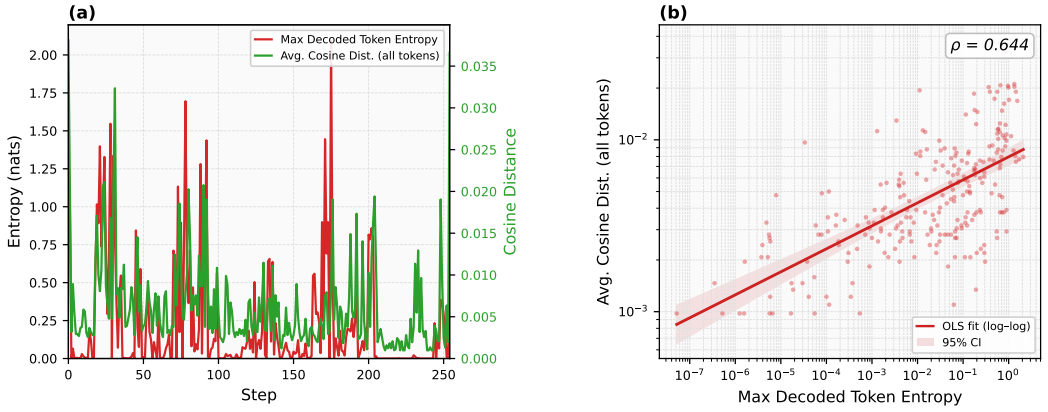

Figure 1: (a) Decoded-token entropy and value-vector cosine distance over denoising steps for a single GSM8K sample with LLaDA-8B-Instruct. (b) Maximum decoded-token entropy versus average value-vector cosine distance on log–log axes.

Entropy is a widely-used measure of predictive uncertainty in neural networks [17, 5, 24]. We examine whether this signal can also serve as a predictor of KV cache drift in dLLMs, by measuring the relationship between prediction uncertainty at each denoising step and the resulting shift in the model's internal representations. Our central claim is that the **maximum entropy of the newly decoded token distributions** at each step serves as an effective, low-cost predictor of how much the KV cache will drift at the subsequent step, providing an effective proxy for selective recomputation.

We run LLaDA-8B-Instruct on a single GSM8K sample with 256 denoising steps, recording two quantities at every step: (i) the maximum entropy among the newly decoded tokens and (ii) the average cosine distance between value vectors at consecutive denoising steps (t and $t{+}1$) across all token positions, which we take as our ground-truth measure of KV cache drift. We present this single-

sample analysis here to build intuition; an extended study aggregating 64 samples per benchmark across four datasets, which confirms and strengthens these findings, is provided in Appendix B.

Temporal co-occurrence of entropy spikes and cache drift. Figure 1(a) plots these two signals over the course of generation. The max decoded token entropy tracks the cosine distance spikes remarkably well: steps at which a high-entropy token is committed coincide with pronounced jumps in KV drift, whereas low-entropy decodings are followed by minimal KV drift. This suggests that cache staleness is driven specifically by the surprise introduced at the moment of token commitment.

Quantitative correlation. To move beyond visual inspection, we scatter-plot the max decoded token entropy against the average value-vector cosine distance for all 256 steps on log–log axes (Figure 1(b)). The two quantities exhibit a clear positive correlation in log–log space with a Spearman rank correlation of $\rho = 0.644$. The extended analysis shows that this trend holds consistently across tasks and samples (Appendix B).

Intuition. We hypothesize that this relationship stems from how dLLMs propagate local changes in the input. When a mask token is resolved from a high-entropy (i.e., highly uncertain) distribution, the change in input sequence may inject a large amount of new information or surprise into the sequence. Conversely, when a token is decoded from a near-deterministic distribution, the model has already anticipated that outcome in its representations, and the resulting perturbation is small.

3.2 Feature volatility persists beyond the decoding step

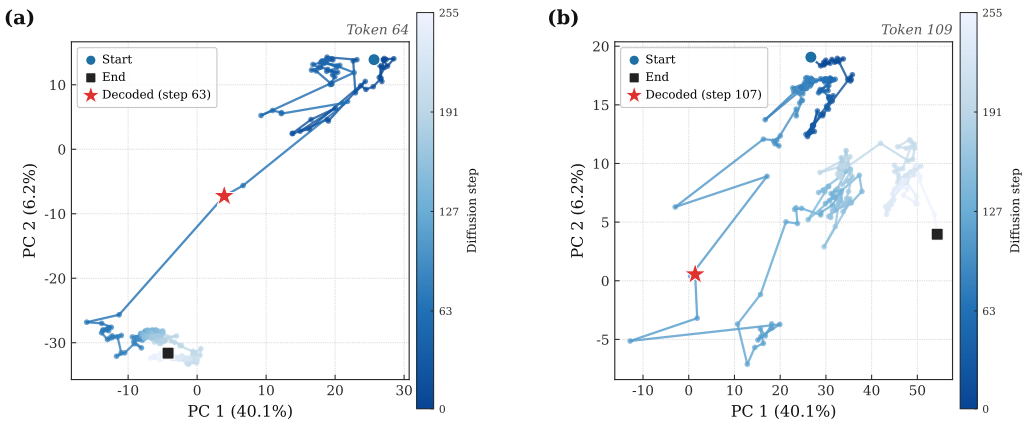

Figure 2: (a)–(b) PCA projections (PC 1 vs. PC 2) of the last-layer value vectors for two mask tokens over 256 denoising steps in LLaDA-8B-Instruct on a single GSM8K sample. Color encodes step progression (dark→light); the red star marks the decoding step.

The previous section established *when* the KV cache drifts; we now examine *which tokens* are most affected and for *how long*. To this end, we track individual token trajectories in the value-vector space across all denoising steps.

PCA trajectories of individual tokens. Following d^{2}Cache [8], we apply PCA to the last-layer value vectors of all mask-token positions across 256 denoising steps and visualize two tokens in Figure 2(a)–(b). Token 64 experiences a sharp shift in trajectory directly after being decoded but converges rapidly in the following steps. By contrast, Token 109 continues to drift through the principal-component space for many steps after being decoded, exhibiting sustained volatility before eventual stabilization. These two cases illustrate a spectrum: while some tokens stabilize almost immediately upon unmasking, others remain volatile well beyond the single decode step.

Implication. Prior dynamic caching methods [8, 10] acknowledge the importance of recomputing recently decoded tokens, yet restrict their recomputation only to the single following step. The above analysis suggests that this single-step scope may be insufficient, and that extending recomputation to a broader window of recently decoded tokens could better capture the feature dynamics.

4 Method

Motivated by the observations in Section 3, we propose EntropyCache, a KV caching method for dLLMs that ① uses a single entropy scalar to decide *when* to trigger full recomputation and ② recomputes the k most recently decoded tokens to account for multi-step feature volatility. Figure 3 illustrates the three phases executed at each denoising step; we describe each component below.

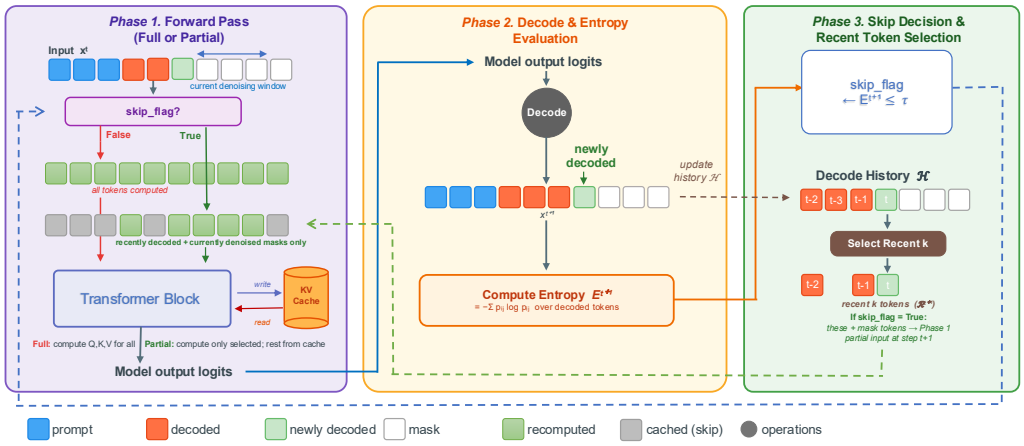

Figure 3: Overview of EntropyCache at a single denoising step t. **Phase 1**: a full or partial forward pass is executed depending on the recompute flag; in the partial case, only mask tokens and recently decoded tokens are recomputed while the rest are read from cache. **Phase 2**: new tokens are decoded from the model logits and the maximum entropy E^{t+1} of the decoded distributions is computed. **Phase 3**: the entropy is compared against threshold τ to set the recompute flag for the next step, and the k most recently decoded tokens $\mathcal{R}^{t+1}$ are selected for partial recomputation.

4.1 Entropy-based skipping

Section 3.1 showed that the maximum entropy of newly decoded tokens is an effective predictor of KV cache drift. We exploit this by using it as a binary trigger for full recomputation.

At each step t, after decoding new tokens $\mathcal{D}^{t+1}$ from the model logits over all vocab indexes $j \in V$, we compute the maximum entropy over their predicted distributions:

$$E^{t+1} = \max_{i \in \mathcal{D}^{t+1}} \left(-\sum_{j} p_{ij} \log p_{ij} \right). \tag{1}$$

If $E^{t+1} > \tau$, the decoded tokens originate from uncertain distributions and a large cache drift is expected; therefore, we trigger a full forward pass at step $t{+}1$ to refresh the entire KV cache. If $E^{t+1} \leq \tau$, we reuse the existing KV cache, executing only a partial forward pass over the selected subset of tokens. Computing entropy for $N = |\mathcal{D}^{t+1}|$ newly decoded distributions costs $O(NV)$, where V is the vocabulary size. This overhead is independent of context length, model depth, and hidden dimension, and N is small under confidence-based parallel decoding.

4.2 Recent token recomputation

When the entropy-based decision skips full recomputation, we find that updating a small subset of recently decoded tokens alongside the current mask tokens is sufficient to retain model accuracy. Section 3.2 demonstrates that feature volatility persists for multiple steps after a token is unmasked, not just the single step assumed by prior methods. Accordingly, we maintain a decode history $\mathcal{H} \in \mathbb{R}^L$ and at each step select a fixed budget of k recently decoded tokens for recomputation.

Formally, each entry of the history vector records when a position was unmasked:

$$\mathcal{H}_i = \begin{cases} n & \text{if position } i \text{ was decoded at step } n, \\ -\infty & \text{otherwise.} \end{cases} \tag{2}$$

We then select the k positions with the largest history values:

$$\mathcal{K} = \underset{\mathcal{S}\subset\{1,\ldots,|\mathcal{H}|\},\ |\mathcal{S}|=k}{\arg\max} \sum_{j\in\mathcal{S}} \mathcal{H}_j, \tag{3}$$

and define a recency threshold:

$$\tau_{\mathcal{R}} = \max\big(\min_{j\in\mathcal{K}} \mathcal{H}_j,\ t - \Delta t_{recompute}\big) \tag{4}$$

where $\Delta t_{recompute}$ is the number of steps elapsed since the last full recomputation and t is the current time step. Since a full forward pass resets all KV states to their exact values, subsequent drift is driven primarily by tokens decoded after that point; the threshold $\tau_{\mathcal{R}}$ therefore restricts the recomputation budget to these positions, where volatility is concentrated. The selected recent tokens mask is then $\mathcal{R}^{t+1} = \mathbb{I}(\mathcal{H}_i \geq \tau_{\mathcal{R}})$. In the partial forward pass, only the current mask tokens $\mathcal{M}^{t+1}$ and the selected recent tokens have their K, V recomputed; all other positions are reused from the cached KV states.

Why a fixed token budget rather than a fixed step window. An alternative design would recompute all tokens decoded within the last n steps. However, when combined with confidence-based parallel decoding, the number of tokens unmasked per step varies substantially; high-confidence steps may decode dozens of tokens at once. A fixed step window therefore leads to unpredictable and often excessive recomputation in such steps, degrading throughput without proportional accuracy gains. By fixing the token budget k instead, EntropyCache maintains a stable computational cost per partial step regardless of how many tokens are decoded at each iteration.

4.3 Pseudocode

Algorithm 1 summarizes the complete EntropyCache procedure. Starting from a full prefill of the prompt, each iteration decodes new tokens, evaluates entropy, and decides whether to perform a full or partial forward pass at the next step. The full algorithm with layer-level details is provided in Appendix A.

Algorithm 1: EntropyCache

Input: Prompt $\mathbf{x}_{\text{prompt}}$, generation length N, entropy threshold τ, recent-token budget k

`// Initialization`

$t \leftarrow 1$; $\mathcal{H} \leftarrow \{-\infty\}^N$; $\mathcal{A} \leftarrow \text{FullPrefill}(\mathbf{x}_{\text{prompt}})$; $\mathcal{M}^1 \leftarrow \text{InitialWindow}$

while $\mathcal{M}^t \neq \emptyset$ **do**

  $\mathbf{x}^{t+1},\ \mathcal{D}^{t+1} \leftarrow \text{decode}(\mathbf{x}^t,\ \mathcal{M}^t,\ \mathcal{A})$ `// Decode`

  $\mathcal{H}[\mathcal{D}^{t+1}] \leftarrow t$ `// Update history`

  $\mathcal{K} \leftarrow \underset{|S|=k}{\arg\max} \sum_{j\in S} \mathcal{H}_j$; $\tau_{\mathcal{R}} \leftarrow \max\big(\min_{j\in\mathcal{K}} \mathcal{H}_j,\ t - \Delta t_{recompute}\big)$; $\mathcal{R}^{t+1} \leftarrow \{i \mid \mathcal{H}_i \geq \tau_{\mathcal{R}}\}$

  $E^{t+1} \leftarrow \max_{i\in\mathcal{D}^{t+1}} \big(-\sum_j p_{ij} \log p_{ij}\big)$ `// Entropy`

  if $E^{t+1} \leq \tau$ **then**

    $\mathcal{A} \leftarrow \mathcal{A}$ `// reuse KV-cache with` $\mathcal{R}^{t+1}$

  else

    $\mathcal{A} \leftarrow \text{FullRecompute}(\mathbf{x}^{t+1})$

  end

  $\mathcal{M}^{t+1} \leftarrow \mathcal{M}^t \setminus \mathcal{D}^{t+1}$; $t \leftarrow t+1$

end

return $\mathbf{x}^t$

5 Experiments

5.1 Experimental setup

Unless otherwise specified, all experiments are conducted on NVIDIA RTX 3090 GPUs. To ensure a fair comparison of KV cache mechanisms, we re-implement all methods within a unified codebase to share the same decoding logic. We evaluate LLaDA-8B-Instruct and Dream-7B-Instruct using sliding window decoding (window size = 32), except for Fast-dLLM, which strictly requires blockwise decoding (block size = 32). All KV cache methods use confidence-based parallel decoding with a threshold of 0.9. For dynamic KV cache methods, we focus on d^2Cache and Elastic-Cache as baselines, as prior work [8, 11] has shown them to be more effective than other previous approaches [10, 9] under comparable settings. We use the optimal parameters reported in their respective papers: $\text{rollout}_p = 0.2$ for d^2Cache and a cosine similarity threshold of 0.9 for Elastic-Cache. For EntropyCache, we set $\tau = 1.5$ and $k = 64$.

We evaluate on two benchmark suites: (1) standard benchmarks including GSM8K [4] 4-shot, MATH500 [6] 4-shot, MBPP [2] 3-shot, and HumanEval [3] zero-shot; and (2) chain-of-thought benchmarks including GSM8K 8-shot, MMLU-Pro [21], BBH 3-shot [18], and GPQA zero-shot [15].

5.2 Main results

Table 1: Comprehensive dLLM KV cache method comparison. Speedup (×) relative to Baseline. Underlined scores meet or exceed Baseline accuracy. Acc.: Accuracy (%). T-put: Throughput (tok/s). Superscripts indicate max generation length.

	GSM8K^{256}		**MATH500**^{512}		**MBPP**^{256}		**HumanEval**^{512}		**Average**	
Method	Acc.	T-put	Acc.	T-put	Acc.	T-put	Acc.	T-put	Acc.	T-put
LLADA-8B-Instruct										
Baseline	77.86	2.28	42.40	2.47	42.20	2.89	46.34	4.32	52.20	2.99
Baseline+Parallel	78.47	8.56 (3.8×)	42.40	9.50 (3.9×)	42.40	16.37 (5.7×)	48.17	15.40 (3.6×)	52.86	12.46 (4.2×)
Fast-DLLM (Dual)	78.01	32.54 (14.3×)	40.00	32.81 (13.3×)	39.20	45.58 (15.8×)	44.51	37.71 (8.7×)	50.43	37.16 (12.4×)
Elastic-Cache	79.15	23.49 (10.3×)	43.80	24.43 (9.9×)	40.60	32.24 (11.2×)	44.51	27.73 (6.4×)	52.02	26.97 (9.0×)
d^2Cache	78.01	29.96 (13.2×)	43.60	32.18 (13.1×)	40.80	52.77 (18.3×)	41.46	41.36 (9.6×)	50.97	39.06 (13.1×)
EntropyCache (Ours)	78.77	38.49 (16.9×)	42.80	39.64 (16.1×)	38.80	64.91 (22.5×)	48.78	38.41 (8.9×)	**52.28**	**45.36 (15.2×)**
Dream-v0-Instruct-7B										
Baseline	74.30	2.60	44.60	2.60	50.60	3.31	53.05	3.87	55.64	3.10
Baseline+Parallel	74.91	12.14 (4.7×)	44.00	15.86 (6.1×)	48.60	29.66 (9.0×)	53.05	38.74 (10.0×)	55.14	24.10 (7.8×)
Fast-DLLM (Dual)	72.25	42.17 (16.2×)	40.20	43.52 (16.7×)	51.40	69.36 (21.0×)	55.49	71.71 (18.5×)	54.83	56.69 (18.3×)
Elastic-Cache	69.60	32.15 (12.4×)	38.80	41.90 (16.1×)	45.80	54.84 (16.6×)	50.61	72.30 (18.7×)	51.20	50.30 (16.2×)
d^2Cache	75.44	41.68 (16.0×)	40.60	55.80 (21.4×)	48.40	87.63 (26.5×)	53.66	110.83 (28.6×)	54.52	73.98 (23.9×)
EntropyCache (Ours)	74.30	48.32 (18.6×)	43.20	62.48 (24.0×)	48.80	97.79 (29.6×)	58.54	119.02 (30.7×)	**56.21**	**81.90 (26.4×)**

Table 2: Chain-of-Thought (CoT) benchmark comparison. Speedup (×) relative to Baseline. Acc.: Accuracy (%). T-put: Throughput (tok/s).

	GSM8K 8-shot		**MMLU-Pro**		**BBH 3-shot**		**GPQA Zero-shot**		**Average**	
Method	Acc.	T-put	Acc.	T-put	Acc.	T-put	Acc.	T-put	Acc.	T-put
LLADA-8B-Instruct										
Baseline	81.12	2.07	37.60	1.68	59.58	1.35	27.68	5.76	51.49	2.71
Baseline+Parallel	81.43	12.44 (6.0×)	37.57	4.77 (2.8×)	57.90	21.45 (15.9×)	27.68	11.22 (1.9×)	51.15	12.47 (4.6×)
Fast-DLLM (Dual)	80.44	41.68 (20.1×)	37.15	21.56 (12.8×)	52.48	36.89 (27.3×)	24.78	25.34 (4.4×)	48.71	31.37 (11.6×)
Elastic-Cache	81.20	32.89 (15.9×)	36.69	14.78 (8.8×)	52.57	42.39 (31.4×)	24.78	18.08 (3.1×)	48.81	27.04 (10.0×)
d^2Cache	81.35	47.03 (22.7×)	36.95	16.80 (10.0×)	55.68	83.94 (62.2×)	22.10	23.22 (4.0×)	**49.02**	42.75 (15.8×)
EntropyCache (Ours)	81.20	62.31 (30.1×)	36.06	15.21 (9.1×)	52.48	144.90 (107.3×)	24.55	20.98 (3.6×)	48.57	**60.85** (22.4×)
Dream-v0-Instruct-7B										
Baseline	78.01	2.51	47.12	2.01	65.69	1.61	27.01	6.14	54.46	3.07
Baseline+Parallel	77.41	13.95 (5.6×)	46.64	10.93 (5.4×)	60.76	25.42 (15.8×)	24.78	19.12 (3.1×)	52.40	17.36 (5.7×)
Fast-DLLM (Dual)	74.22	55.50 (22.1×)	41.86	35.01 (17.4×)	58.09	40.96 (25.4×)	25.22	34.69 (5.6×)	49.85	41.54 (13.5×)
Elastic-Cache	69.37	36.95 (14.7×)	42.50	26.50 (13.2×)	31.52	52.61 (32.7×)	27.68	26.90 (4.4×)	42.77	35.74 (11.6×)
d^2Cache	73.69	49.59 (19.8×)	43.33	40.95 (20.4×)	59.51	99.60 (61.8×)	27.01	35.23 (5.7×)	50.89	56.34 (18.4×)
EntropyCache (Ours)	75.74	60.01 (23.9×)	45.49	42.14 (21.0×)	55.86	155.30 (96.4×)	26.79	38.73 (6.3×)	**50.97**	**74.05 (24.1×)**

Standard benchmarks. Table 1 compares all methods on four standard benchmarks. EntropyCache achieves the best accuracy and throughput among all caching methods: on LLaDA-Instruct it attains 52.28% average accuracy at 15.2× speedup, and on Dream-Instruct it reaches 56.21% at 26.4×

speedup. No other caching method preserves baseline-level average accuracy while achieving comparable throughput. The advantage is clearest on HumanEval, where all other methods degrade accuracy on at least one model. EntropyCache is the only method that on average achieves lossless performance compared to baseline on both models, while still delivering $8.9\times$–$30.7\times$ speedup.

Chain-of-thought benchmarks. Table 2 evaluates on longer-context CoT tasks. EntropyCache achieves $22.4\times$ and $24.1\times$ average speedup on LLaDA-Instruct and Dream-Instruct, roughly $1.4\times$ the next-best method (d^{2}Cache). On BBH 3-shot the gap widens to $1.7\times$, with EntropyCache reaching $107.3\times$ speedup on LLaDA and $96.4\times$ on Dream, as long generations with many low-entropy steps maximize cache reuse. Accuracy is highest among caching methods on Dream-Instruct (50.97%) and within 0.5pp of d^{2}Cache on LLaDA-Instruct (48.57% vs. 49.02%).

EntropyCache's performance on BBH and MBPP is further analyzed in Section 5.3 and Appendix D, respectively: on BBH, EntropyCache Pareto-dominates d^{2}Cache once both methods are swept across configurations, and on MBPP the accuracy is largely preserved across a range of τ values, indicating that the default-configuration drop reflects a local minimum rather than a fundamental limitation.

5.3 Ablation study

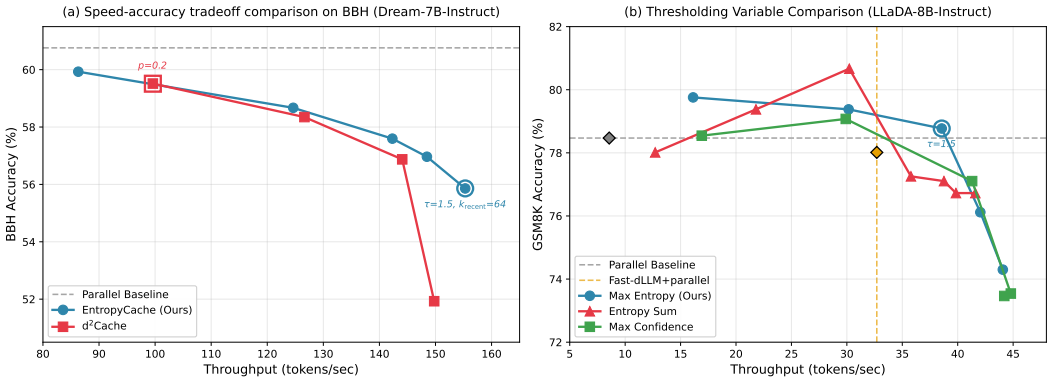

Figure 4: **(a)** Speed-accuracy tradeoff comparison on BBH between EntropyCache and d^{2}Cache (Dream-7B-Instruct). Hollow markers denote default configurations. **(b)** Accuracy–throughput tradeoff on GSM8K (LLaDA-8B-Instruct) for three candidate skipping metrics across varying thresholds τ.

Speed-accuracy tradeoff comparison on BBH. We further analyze EntropyCache's behavior on long-generation tasks by tracing out the speed-accuracy tradeoff on BBH with Dream-7B-Instruct, going beyond the single-configuration view of Table 2. We sweep τ for EntropyCache and $\mathrm{rollout}_p$ for d^{2}Cache (Figure 4(a)); the default configurations of the two methods land at different points on the tradeoff curve, which makes a single-configuration comparison misleading. Across the full sweep, EntropyCache consistently outperforms d^{2}Cache: at matched throughput, EntropyCache achieves higher accuracy. EntropyCache also maintains stable accuracy at high throughput, whereas d^{2}Cache degrades sharply as $\mathrm{rollout}_p$ decreases.

Choice of thresholding variable We compare three candidate metrics for adaptive skipping: (1) max entropy, which takes the maximum entropy of decoded tokens; (2) entropy sum, which uses the summation of entropy over all decoded tokens; and (3) max confidence, which uses the maximum token-level probability across decoded tokens. For each metric, we sweep the threshold τ and plot the resulting accuracy–throughput tradeoff on GSM8K (Figure 4(b)). Max entropy produces the most favorable tradeoff among the three: at $\tau = 1.5$ it achieves approximately 78.8% accuracy at $\sim$38.6 tok/s, surpassing both the parallel baseline's accuracy and Fast-dLLM DualCache's throughput. Entropy sum peaks sharply at $\tau = 1.5$ but degrades beyond it, making threshold selection sensitive to miscalibration. Max confidence is dominated by max entropy across most of the throughput range. Based on these results, we adopt max entropy as the default thresholding variable for all experiments.

Joint parameter sensitivity. Figure 5 jointly sweeps τ and k_{recent} on GSM8K for both models. The entropy threshold controls the primary accuracy–throughput trade-off, while the recent-token budget provides complementary accuracy recovery at modest cost. The shared setting $\tau = 1.5$, $k_{\text{recent}} = 64$ lies near the knee for both models; on Dream, increasing k_{recent} from 16 to 64 improves accuracy from 72.9% to 74.3% for less than 2 tok/s.

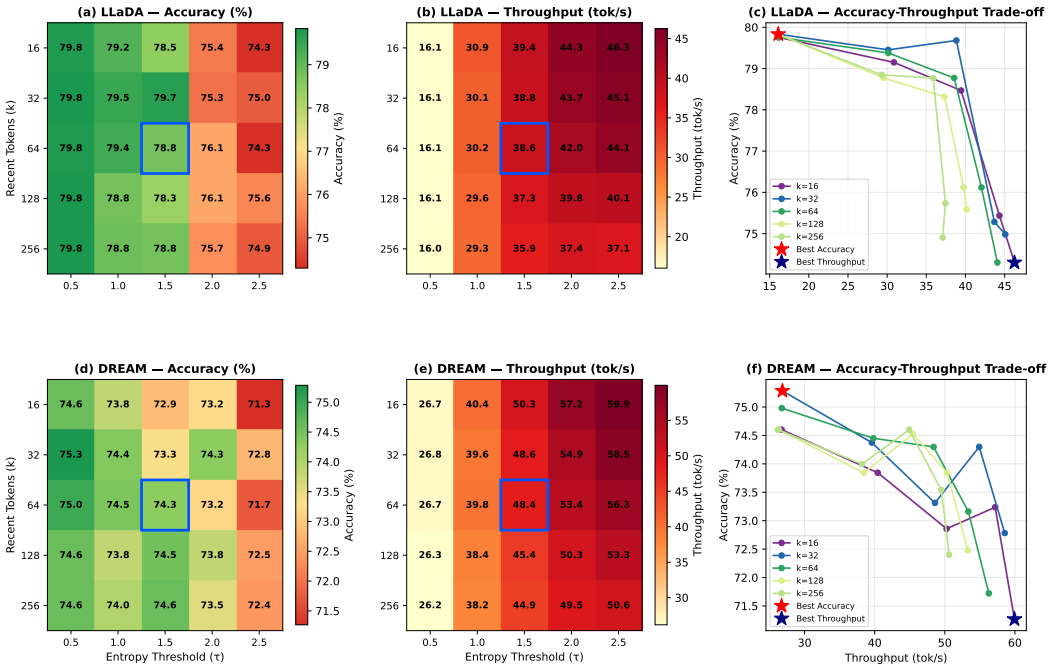

Figure 5: Joint τ–k_{recent} sweep on GSM8K. Top: LLaDA; bottom: Dream. Left to right: accuracy, throughput, and their trade-off.

Robustness across hyperparameters and inference settings. Appendix D evaluates robustness to the sliding-window size and reports per-benchmark τ–k_{recent} sweeps across all four standard benchmarks. Appendix C provides matched operating-point comparisons and deployment-oriented analyses.

5.4 Overhead analysis

Dynamic KV caching methods introduce extra computation at each step to decide where or when to refresh the cache. A key practical advantage of EntropyCache is that its decision computation is decoupled from both context length and model scale. Table 3 compares the per-step complexity of the three dynamic caching methods. d^{2}Cache computes attention rollout across all layers and heads, incurring $O(\ell \cdot L \cdot H)$ computation and $O(H \cdot L^2)$ memory, quadratic in context length. Elastic-Cache performs layer-wise partial attention score calculation at cost $O(\ell \cdot L \cdot d)$, scaling with model depth, context length, and hidden dimension simultaneously. In contrast, EntropyCache's skip decision only requires entropy evaluation over the N newly decoded vocabulary distributions, costing $O(NV)$ computation, plus $O(T)$ memory for the decode history over T generated positions. Although this cost is asymptotically linear in V, it is entirely independent of context length L, model depth ℓ, and hidden dimension d; moreover, confidence-based decoding typically yields only a few newly decoded tokens per step. Appendix C.5 quantifies this cost across vocabulary sizes from 32K to 320K. Wall-clock profiling confirms this gap: EntropyCache's decision logic accounts for only 0.5% of inference time, versus 9.2% for d^{2}Cache and 14.5% for Elastic-Cache (Figure 6).

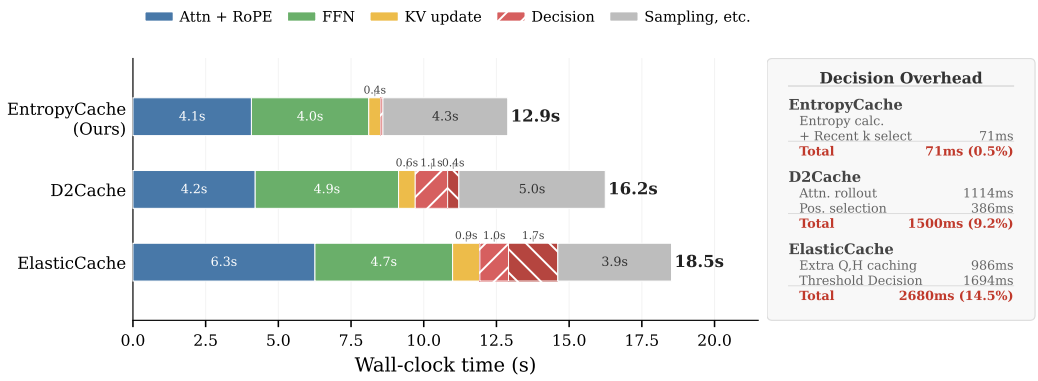

Figure 6: Wall-clock time breakdown on a single GSM8K sample (LLaDA-Inst, $w = 32$, generation length $= 256$). Left: stacked bar chart of inference components. Right: decision overhead decomposition per method. EntropyCache's decision cost is 71 ms (0.5%), roughly $20\times$ and $38\times$ cheaper than d^{2}Cache and Elastic-Cache respectively.

Table 3: Complexity comparison of dynamic KV cache methods. EntropyCache decouples decision computation from both context length and model scale.
Notation: ℓ = number of layers; L = context length; H = number of attention heads; d = hidden dimension; V = vocabulary size; N = number of newly decoded tokens; T = number of generated positions. d^{2}Cache and Elastic-Cache overhead grows with L and model scale (ℓ, d), respectively, whereas EntropyCache's decision computation is independent of both context length and model scale.

Method	**Computation**	**Memory**	**Bottleneck**
d^{2}Cache [8]	$\mathcal{O}(\ell \cdot L \cdot H)$	$\mathcal{O}(H \cdot L^2)$	Quadratic in L
Elastic-Cache [11]	$\mathcal{O}(\ell \cdot L \cdot d)$	$\mathcal{O}(\ell \cdot L \cdot d)$	Scales with ℓ, L, d
EntropyCache (Ours)	$\mathcal{O}(N \cdot V)$	$\mathcal{O}(T)$	**Compute independent of** L, ℓ, d

6 Conclusion

We presented EntropyCache, a KV caching method for diffusion-based large language models that provides a cheap proxy for deciding *when* recomputation is necessary. By exploiting the empirical correlation between decoded token entropy and KV cache drift, EntropyCache uses a lightweight $O(NV)$ entropy check over newly decoded token distributions to decide whether to reuse cached states or trigger a full forward pass, while recomputing the k most recently decoded tokens to account for multi-step feature volatility. On standard benchmarks, EntropyCache achieves $15.2\times$ to $26.4\times$ average speedup over vanilla dLLM inference with competitive accuracy across all baselines, and maintains consistent gains on chain-of-thought benchmarks that stress longer contexts. Additional ablation studies, overhead profiling, and extended empirical analyses are provided in the appendix.

Acknowledgments and Disclosure of Funding

This work was supported by Samsung Electronics (MX Business), IITP/NRF grants funded by the Korean government (MSIT, 2021-0-00105 Development of Model Compression Framework for Scalable On-Device AI Computing on Edge Applications, 2021-0-01343 Artificial Intelligence Graduate School Program, Seoul National University), and Inter-university Semiconductor Research Center (ISRC), SNU. The authors declare no competing interests.

References

[1] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

[5] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059. PMLR, 2016.

[6] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[7] Zhanqiu Hu, Jian Meng, Yash Akhauri, Mohamed S. Abdelfattah, Jae-sun Seo, Zhiru Zhang, and Udit Gupta. Accelerating diffusion language model inference via efficient kv caching and guided diffusion. *arXiv preprint arXiv:2505.21467*, 2025.

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

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

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

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

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

[13] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624, 2023.

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

[15] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

[16] Subham Sekhar Sahoo, Marianne Arriola, Aaron Gokaslan, Edgar Mariano Marroquin, Alexander M Rush, Yair Schiff, Justin T Chiu, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[17] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[18] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, 2023.

[19] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[21] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

[22] Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm. *arXiv preprint arXiv:2509.26328*, 2025.

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

[24] Guowei Xu, Wenxin Xu, Jiawang Zhao, and Kaisheng Ma. Gift: Guided importance-aware fine-tuning for diffusion language models. *arXiv preprint arXiv:2509.20863*, 2025.

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

A Full EntropyCache algorithm

Algorithm 2: EntropyCache algorithm

Input: Prompt $\mathbf{x}_{\text{prompt}}$, Generation length N, Window size w, Entropy threshold τ, Recent token window k

Data: $\mathbf{x}^0 \leftarrow \{\mathbf{x}_{\text{prompt}}; [\text{MASK}], \ldots, [\text{MASK}]\}$; $P \leftarrow \text{length}(\mathbf{x}_{\text{prompt}})$

$t \leftarrow 1$; $\mathcal{D}^1 \leftarrow \{1, \ldots, P\}$; $\mathcal{M}^1 \leftarrow \{P+1, \ldots, P+N\}$; $\mathcal{R}^1 \leftarrow \emptyset$;

$\mathcal{A}^{0,l} \leftarrow \emptyset$ for all layers l; $\mathcal{D}_{\text{prev}} \leftarrow \emptyset$; skip_flag $\leftarrow$ False

$\mathcal{H} \leftarrow \{-\infty\}^N$

while $\mathcal{M}^t \neq \emptyset$ **do**

  if $t == 1$ *or skip_flag* $=$ *False* **then**

    `// Prefill`

    for $l = 1, \ldots, L$ **do**

      $\mathbf{Q}^{t,l}_{[\mathcal{I}]}, \mathbf{K}^{t,l}_{[\mathcal{I}]}, \mathbf{V}^{t,l}_{[\mathcal{I}]} \leftarrow \text{W}(\mathbf{H}^{t,l-1}_{[\mathcal{I}]})$

      $\mathbf{H}^{t,l}_{[\mathcal{I}]}, \mathbf{S}_{[\mathcal{I}]} \leftarrow \text{TransformerBlock}(\mathbf{Q}^{t,l}_{[\mathcal{I}]}, \mathbf{K}^{t,l}_{[\mathcal{I}]}, \mathbf{V}^{t,l}_{[\mathcal{I}]})$

    end

  else

    for $l = 1, \ldots, L$ **do**

      $\mathbf{H}^{t,0}_{[\mathcal{M}^t]} \leftarrow \mathbf{x}^t_{[\mathcal{M}^t]}$

      $\mathbf{Q}^{t,l}_{[\mathcal{M}^t \cup \mathcal{R}^t]}, \mathbf{K}^{t,l}_{[\mathcal{M}^t \cup \mathcal{R}^t]}, \mathbf{V}^{t,l}_{[\mathcal{M}^t \cup \mathcal{R}^t]} \leftarrow \text{W}(\mathbf{H}^{t,l-1}_{[\mathcal{M}^t \cup \mathcal{R}^t]})$

      $\mathbf{H}^{t,l}_{[\mathcal{M}^t \cup \mathcal{R}^t]}, \mathbf{S}^{t,l}_{[\mathcal{M}^t \cup \mathcal{R}^t]} \leftarrow \text{TransformerBlock}(\mathbf{Q}^{t,l}_{[\mathcal{M}^t \cup \mathcal{R}^t]}, \mathbf{K}^{t,l}_{[\mathcal{I}]}, \mathbf{V}^{t,l}_{[\mathcal{I}]})$

    end

  end

  Decode new tokens: $\mathbf{x}^{t+1}, \mathcal{D}^{t+1} \leftarrow \text{decode}(\mathbf{x}^t, \mathcal{M}^t)$

  Record Decode history: $\mathcal{H}[\mathcal{D}^{t+1}] \leftarrow t$

  `// Select recent tokens`

  $\mathcal{K} \leftarrow \underset{S \subset \{1,\ldots,|\mathcal{H}|\},\, |S|=k}{\arg\max} \sum_{j \in S} \mathcal{H}_j$

  $\tau_{\mathcal{R}} \leftarrow \max(\min_{j \in \mathcal{K}} \mathcal{H}_j,\ t - \Delta t_{recompute})$

  $\mathcal{R}^{t+1} \leftarrow \{i \mid \mathcal{H}_i \geq \tau_{\mathcal{R}}\}$

  $E^{t+1} \leftarrow \max_{i \in \mathcal{D}^{t+1}} \big(e(p_i)\big)$ where $e(p_i) = -\sum_j p_{ij} \log p_{ij}$ `// Compute entropy for next step`

  if $E^{t+1} \leq \tau$ **then**

    skip_flag $\leftarrow$ True `// Set flag for next iteration`

    $\Delta t_{recompute} \leftarrow \Delta t_{recompute} + 1$

  else

    skip_flag $\leftarrow$ False

    $\Delta t_{recompute} \leftarrow 0$

  end

  Update mask positions: $\mathcal{M}^{t+1} \leftarrow \mathcal{M}^t \setminus \mathcal{D}^{t+1}$

  $t \leftarrow t + 1$

end

return $\mathbf{x}^{t-1}$

B Extended empirical analysis on entropy and KV cache drift

The correlation between decoded token entropy and KV cache drift presented in Section 3 was derived from a single generation trajectory. To validate that this relationship holds broadly, we extend the analysis across all four standard benchmarks (GSM8K, MATH500, MBPP, HumanEval), generating 64 samples per dataset using LLaDA-8B-Instruct under the same sliding-window decoding configuration used in our main experiments. For each denoising step of every sample, we record

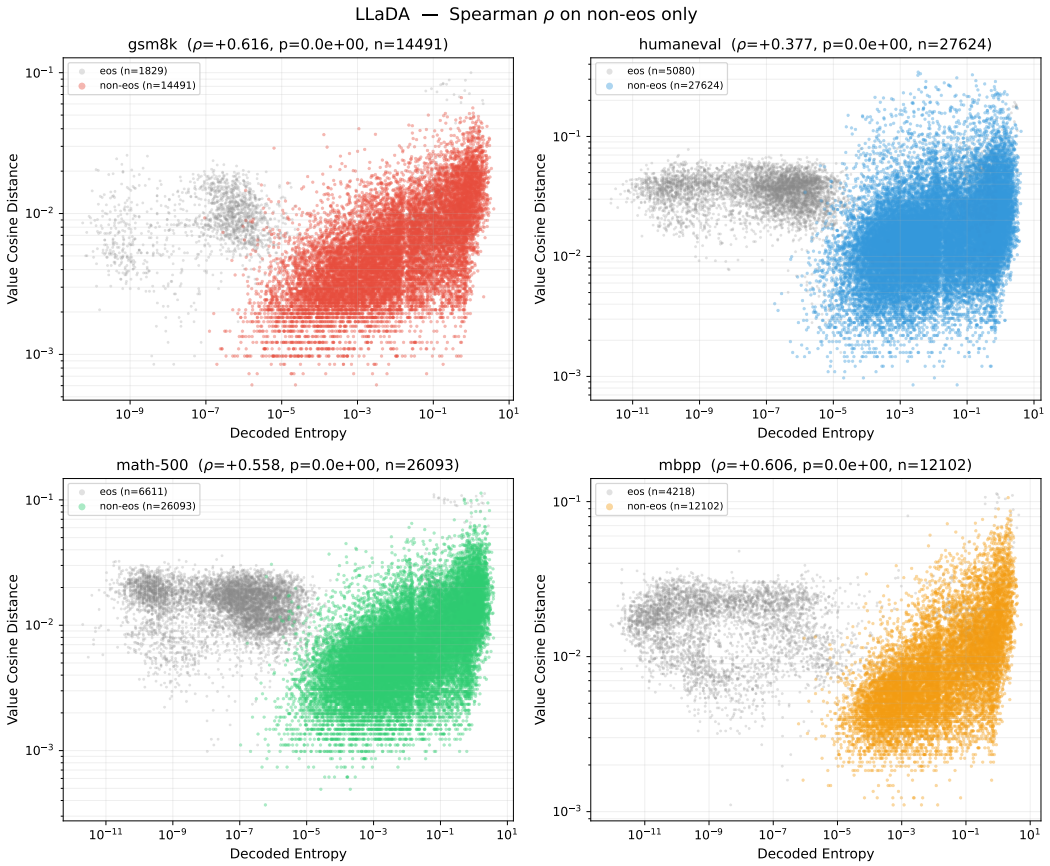

Figure 7: Per-step decoded token entropy vs. average value-vector cosine distance for each benchmark (LLaDA-8B-Instruct, 64 samples per dataset). Gray points denote EOS-filling steps; colored points denote non-EOS content steps. Spearman ρ is computed on non-EOS steps only.

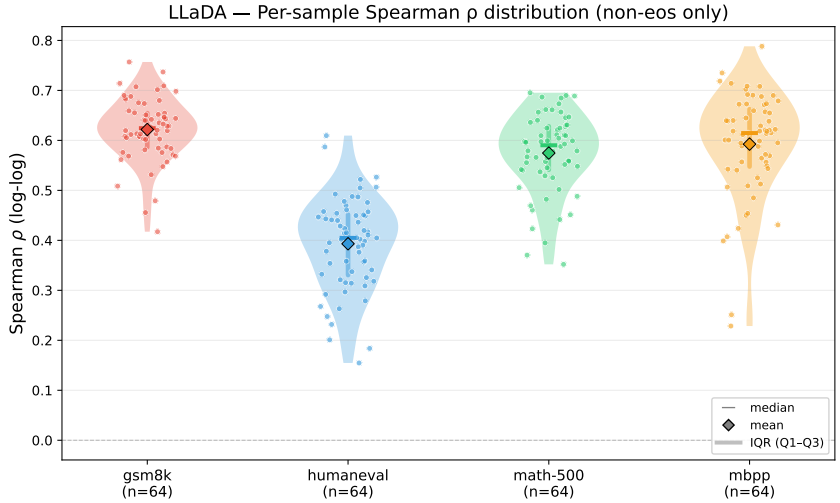

Figure 8: Distribution of per-sample Spearman ρ (log–log, non-EOS only) across 64 samples for each benchmark (LLaDA-8B-Instruct). Diamonds indicate means; horizontal bars indicate medians; shaded regions show kernel density estimates.

the maximum decoded token entropy and the average value-vector cosine distance across all token positions, yielding between 12K and 28K step-level observations per dataset.

Per-dataset scatter analysis. Figure 7 plots the per-step entropy–drift pairs on log–log axes for each benchmark. Across all four datasets, the positive monotonic trend observed in Section 3.1 is clearly reproduced: higher decoded entropy at step t is consistently associated with larger KV cache drift at step $t+1$. The aggregate Spearman correlations on non-EOS steps are $\rho = 0.616$ (GSM8K), $\rho = 0.558$ (MATH500), $\rho = 0.606$ (MBPP), and $\rho = 0.377$ (HumanEval), all with $p < 0.001$.

EOS token outliers. A notable cluster of outlier points appears in the low-entropy, high-drift region of each scatter plot (shown in gray). These correspond to denoising steps in which the model has already generated its complete response and is filling the remaining positions with EOS tokens. Because EOS tokens are produced with near-zero entropy yet cause a non-trivial representational shift as the sequence transitions from active generation to padding, they decouple from the entropy–drift trend that governs content tokens. Crucially, these outliers have no practical impact on EntropyCache: since their entropy falls well below any reasonable threshold τ, they are automatically classified as skip steps, and the method correctly reuses the cached KV states. We therefore report all Spearman correlations on non-EOS steps to reflect the regime in which the entropy trigger is operative.

Per-sample Spearman distribution. To assess the consistency of the entropy–drift relationship at the individual sample level, Figure 8 shows the distribution of per-sample Spearman ρ values (non-EOS, log–log) across the 64 samples for each dataset. GSM8K exhibits the tightest distribution (mean $\rho = 0.62$, IQR ≈ 0.59–0.68), while MATH500 (mean $\rho = 0.57$) and MBPP (mean $\rho = 0.59$) show similarly concentrated positive correlations. HumanEval yields a lower and more dispersed distribution (mean $\rho = 0.40$, IQR ≈ 0.33–0.50), indicating that the entropy signal is a weaker predictor of raw KV drift for code generation sequences.

Weak correlation does not imply weak performance. Interestingly, the weaker entropy–drift correlation on HumanEval does not translate into weaker downstream performance. As shown in Table 1, EntropyCache achieves its highest accuracy improvement over the baseline on HumanEval (+2.44%p for LLaDA, +5.49%p for Dream), despite the comparatively modest ρ. We note that Spearman correlation measures *synchronous* alignment between entropy and drift at each step: a recomputation triggered 1–2 steps after a drift event would appear as a mismatch in this metric, yet may still refresh the KV cache before it affects downstream decoding. Point-wise correlation may therefore underestimate the practical utility of the entropy signal as a recomputation trigger, and we leave a deeper mechanistic analysis of this relationship to future work.

C Additional validation experiments

This section reports additional validation experiments: a full component ablation, matched operating-point comparisons, age-conditioned feature drift, batched decoding, and a vocabulary-scaling microbenchmark. All accuracy–throughput experiments use the same unified implementation and decoding settings as the main evaluation unless otherwise specified.

C.1 Full component ablation

Table 4 reports the component ablation on every standard benchmark. The full method has the highest four-task average accuracy among the caching variants on both models. Removing recent-token recomputation particularly affects HumanEval and several Dream tasks, while removing entropy-triggered full refreshes lowers the average on both models. The expected throughput increase of either ablation reflects less recomputation, but comes at the cost of lower average accuracy.

C.2 Matched operating-point comparison

Default configurations may represent different points on each method's accuracy–efficiency curve. We therefore sweep τ for EntropyCache, rollout proportion for d^{2}Cache, and cosine-similarity threshold for Elastic-Cache on GSM8K (Table 5). On LLaDA, the paper EntropyCache setting is faster than every measured d^{2}Cache configuration while attaining comparable measured accuracy,

Table 4: Full component ablation. Each cell reports accuracy (%) / throughput (tok/s). No cache uses the same confidence-based parallel decoder with full recomputation at every step.

Model	Dataset	No cache	Full EntropyCache	Entropy only ($k = 0$)	Recent only ($\tau = \infty$)
LLaDA-8B-Instruct	GSM8K	78.5 / 8.6	78.8 / 38.5	76.6 / 42.7	74.1 / 47.9
	MATH500	42.4 / 9.5	42.8 / 39.6	39.2 / 43.0	40.8 / 52.7
	MBPP	42.4 / 16.4	38.8 / 64.9	39.0 / 71.0	40.2 / 78.2
	HumanEval	48.2 / 15.4	48.8 / 38.4	42.1 / 43.2	44.5 / 44.0
	Average	52.9 / 12.5	**52.3** / 45.4	49.2 / 49.9	49.9 / 55.7
Dream-7B-Instruct	GSM8K	74.9 / 12.1	74.3 / 48.3	72.6 / 52.9	71.3 / 63.1
	MATH500	44.0 / 15.9	43.2 / 62.5	38.6 / 70.2	43.0 / 82.8
	MBPP	48.6 / 29.7	48.8 / 97.8	44.2 / 115.1	47.4 / 115.2
	HumanEval	53.1 / 38.7	58.5 / 119.0	56.1 / 138.8	56.7 / 140.6
	Average	55.1 / 24.1	**56.2** / 81.9	52.9 / 94.3	54.6 / 100.4

and the EntropyCache frontier dominates the measured Elastic-Cache frontier. At similar recompute ratios of 0.113–0.120, EntropyCache also gives the highest throughput (38.49 vs. 34.5 and 27.1 tok/s). On Dream, the EntropyCache and d^{2}Cache frontiers cross, but EntropyCache remains faster at comparable accuracy and dominates the measured Elastic-Cache points. These sweeps confirm that the advantage is not specific to the paper defaults.

Table 5: Accuracy–efficiency sweeps on GSM8K under matched prompts, decoding, and hardware. RR denotes recompute ratio; bold rows are the paper configurations.

LLaDA-8B-Instruct

Method	Configuration	Acc.	RR	T-put
EntropyCache	$\tau = 2.5$	74.3	0.080	44.1
EntropyCache	$\tau = 2.0$	76.1	0.088	42.0
EntropyCache	$\boldsymbol{\tau = 1.5}$	**78.77**	**0.114**	**38.49**
EntropyCache	$\tau = 1.0$	79.4	0.191	30.2
EntropyCache	$\tau = 0.5$	79.8	0.489	16.1
d^{2}Cache	rollout $= 0.10$	77.0	0.093	38.1
d^{2}Cache	rollout $= 0.15$	77.4	0.120	34.5
d^{2}Cache	**rollout** $= 0.20$	**78.01**	**0.147**	**29.96**
d^{2}Cache	rollout $= 0.25$	77.9	0.178	26.1
d^{2}Cache	rollout $= 0.30$	78.2	0.201	24.3
Elastic-Cache	similarity $= 0.70$	74.1	0.068	30.3
Elastic-Cache	similarity $= 0.80$	75.3	0.090	28.6
Elastic-Cache	similarity $= 0.85$	78.1	0.113	27.1
Elastic-Cache	**similarity** $= 0.90$	**79.15**	**0.140**	**23.49**
Elastic-Cache	similarity $= 0.95$	79.4	0.233	20.8

Dream-7B-Instruct

Method	Configuration	Acc.	RR	T-put
EntropyCache	$\tau = 2.5$	71.7	0.083	56.3
EntropyCache	$\tau = 2.0$	73.2	0.094	53.4
EntropyCache	$\boldsymbol{\tau = 1.5}$	**74.30**	**0.123**	**48.32**
EntropyCache	$\tau = 1.0$	74.5	0.190	39.8
EntropyCache	$\tau = 0.5$	75.0	0.372	26.7
d^{2}Cache	rollout $= 0.10$	73.9	0.091	50.4
d^{2}Cache	rollout $= 0.15$	74.6	0.118	44.6
d^{2}Cache	**rollout** $= 0.20$	**75.44**	**0.145**	**41.68**
d^{2}Cache	rollout $= 0.25$	75.0	0.175	38.5
d^{2}Cache	rollout $= 0.30$	74.6	0.198	34.2
Elastic-Cache	similarity $= 0.70$	67.8	0.064	38.1
Elastic-Cache	similarity $= 0.80$	68.3	0.085	36.3
Elastic-Cache	similarity $= 0.85$	69.4	0.110	34.0
Elastic-Cache	**similarity** $= 0.90$	**69.60**	**0.152**	**32.15**
Elastic-Cache	similarity $= 0.95$	71.3	0.235	26.6

C.3 Multi-step feature volatility

The trajectory examples in Section 3.2 qualitatively motivate recomputation beyond the immediately preceding step. To test this at scale, we aggregate value-vector cosine distance by token age over 64 samples from each standard benchmark (Table 6). Drift falls sharply after unmasking, but remains elevated for several subsequent steps: at age 5 it is still approximately 3–5$\times$ the age-40 level across all tasks. This long-tail behavior supports a multi-step recent-token budget rather than one-step delayed recomputation. The downstream τ–k_{recent} grids in Appendix D provide complementary evidence that moderate recent-token budgets improve the accuracy–throughput trade-off.

Table 6: Mean value-vector cosine distance by token age, measured over 64 samples per dataset with LLaDA-8B-Instruct. Age is the number of denoising steps since unmasking.

Dataset	Age 1	Age 2	Age 3	Age 5	Age 10	Age 20	Age 40
GSM8K	0.617	0.083	0.044	0.029	0.020	0.015	0.011
MATH500	0.644	0.129	0.074	0.047	0.030	0.019	0.013
MBPP	0.659	0.137	0.080	0.052	0.023	0.016	0.012
HumanEval	0.644	0.121	0.061	0.040	0.025	0.020	0.012

C.4 Batched decoding

We evaluate the indexed in-place KV-update implementation under batching on LLaDA-8B-Instruct/GSM8K (Table 7). RTX 6000 Ada GPUs are used because the RTX 3090 setup used in the main experiments does not provide sufficient memory for the larger batches. The nearly flat no-cache and d^{2}Cache throughput curves are consistent with compute saturation: no-cache recomputes all positions, while d^{2}Cache recomputes more positions and incurs attention-rollout overhead. EntropyCache leaves greater batching headroom through lower recomputation and cheaper decisions, increasing throughput by 1.35× from $B = 1$ to $B = 8$ while remaining faster at every batch size. During partial passes, selected token positions form the query rows and their newly projected K/V entries are written directly to the same indices in preallocated cache buffers; unchanged entries require no reallocation or full-block copy.

Table 7: Batched GSM8K throughput (tok/s) on LLaDA-8B-Instruct using RTX 6000 Ada GPUs. B is the per-device batch size.

Method	$B = 1$	$B = 2$	$B = 4$	$B = 8$
No cache	13.95	13.55	13.02	12.66
d^{2}Cache	61.45	62.10	60.28	59.41
EntropyCache	**67.30**	**82.50**	**90.03**	**90.99**

C.5 Vocabulary-size scaling

Table 8 measures the entropy-decision cost across vocabulary sizes. Although the operation is asymptotically $O(NV)$, GPU latency remains 81–122 μs for $N \leq 8$ across a 10× vocabulary range because the small practical regime is dominated by launch overhead. Scaling becomes visible at the conservative $N = 32$ upper bound, yet remains below 0.41 ms at a 320K vocabulary. The measured LLaDA and Dream costs at $N = 32$ are only 0.179 and 0.209 ms, respectively. This is consistent with the end-to-end profile in Figure 6, where entropy decisions account for 71 ms (0.5%) of a full generation.

Table 8: Entropy-decision GPU time (μs) as vocabulary size and the number of newly decoded tokens N vary.

Vocabulary	**Representative model**	$N = 1$	$N = 8$	$N = 32$
32K	–	86	82	81
126K	LLaDA	86	81	179
152K	Dream	92	87	209
262K	Gemma 2	91	104	340
320K	–	92	122	409

D Extended ablation study

D.1 Window size ablation

Table 9 reports accuracy and throughput for EntropyCache ($\tau = 1.5$, $k = 64$) across sliding-window sizes $w \in \{8, 16, 32, 64, 128\}$ on all four standard benchmarks.

For LLaDA-Inst, accuracy remains stable across $w = 8$ to $w = 64$ on most tasks, with $w = 32$ offering the best accuracy–throughput balance: it achieves the highest or near-highest accuracy on GSM8K (78.6%), MATH500 (42.8%), and HumanEval (47.6%), while delivering competitive throughput across all benchmarks. Smaller windows ($w = 8, 16$) preserve accuracy but sacrifice throughput due to the increased number of autoregressive blocks required to cover the generation length.

For Dream-Inst, the sensitivity to window size is more pronounced. Accuracy degrades substantially at $w = 64$ and $w = 128$—for instance, GSM8K drops from 74.3% ($w = 32$) to 61.9% ($w = 64$) and 49.7% ($w = 128$), and MATH500 collapses from 43.2% to 10.6% at $w = 128$. This aligns with

findings in prior work on semi-autoregressive decoding for dLLMs [25, 23]: when the denoising window is large, the model must resolve many tokens simultaneously without the left-to-right bias that smaller windows naturally impose, leading to degraded generation quality. The $w = 32$ configuration strikes the best trade-off between parallel efficiency and the sequential denoising structure that these models benefit from, and we adopt it as the default for all main experiments.

Table 9: Window size ablation for EntropyCache ($\tau = 1.5$, $k = 64$). Acc.: Accuracy (%). T-put: Throughput (tok/s). Bold indicates the selected configuration ($w = 32$).

	$w = 8$		$w = 16$		$\mathbf{w = 32}$		$w = 64$		$w = 128$	
Benchmark	Acc.	T-put	Acc.	T-put	Acc.	T-put	Acc.	T-put	Acc.	T-put
LLaDA-Inst										
GSM8K	79.08	31.66	78.92	36.68	**78.62**	**38.62**	77.33	35.82	68.84	28.99
MATH500	41.60	31.34	40.80	37.45	**42.80**	**39.52**	40.60	37.51	38.20	29.77
MBPP	42.00	40.42	40.00	55.52	**38.40**	**64.53**	39.20	65.71	37.80	63.93
HumanEval	47.56	31.64	43.90	35.32	**47.56**	**37.86**	47.56	36.25	50.00	30.72
Dream-Inst										
GSM8K	74.83	37.52	75.36	47.42	**74.30**	**48.06**	61.87	44.67	49.66	38.16
MATH500	43.80	39.10	42.20	54.85	**43.20**	**62.41**	33.60	55.74	10.60	41.87
MBPP	53.00	52.36	56.60	79.68	**48.80**	**97.96**	36.80	103.77	28.20	92.71
HumanEval	54.27	52.96	60.98	86.71	**58.54**	**118.23**	53.66	130.57	43.90	134.79

D.2 EntropyCache parameter ablation on extended datasets

The main text reports the accuracy–throughput grid for GSM8K on both models. Here we present the full per-benchmark ablation over $\tau \in \{0.5, 1.0, 1.5, 2.0, 2.5\}$ and $k \in \{16, 32, 64, 128, 256\}$ on LLaDA-Inst ($w = 32$) in Tables 10–13.

Entropy threshold τ. Across all four benchmarks, accuracy is largely preserved at $\tau \leq 1.5$ and begins to degrade at $\tau \geq 2.0$, while throughput increases monotonically with τ. On GSM8K, accuracy remains within 1%p of the $\tau = 0.5$ baseline up to $\tau = 1.5$ (e.g., 78.8% vs. 79.8% at $k = 64$), but drops by $\sim$3%p at $\tau = 2.0$. A similar pattern holds for MATH500 and HumanEval, where $\tau = 1.5$ maintains competitive accuracy while roughly doubling throughput relative to $\tau = 0.5$. The threshold $\tau = 1.5$ thus consistently sits at the knee of the accuracy–throughput curve across tasks.

Recent token budget k. The effect of k is most visible in throughput: larger k increases the per-step recomputation cost, reducing throughput by 10–15% when moving from $k = 16$ to $k = 256$ at a fixed τ. On accuracy, the impact is more nuanced. For reasoning-heavy tasks (GSM8K, MATH500), moderate values of k (32–64) often perform competitively with very small ($k = 16$) or very large ($k = 256$) budgets. For code generation (MBPP, HumanEval), accuracy is relatively insensitive to k. Overall, $k = 64$ provides a strong accuracy–throughput balance across all four benchmarks without significantly sacrificing throughput relative to smaller values.

Joint selection. Taking both dimensions together, the configuration $\tau = 1.5$, $k = 64$ provides a favorable accuracy–throughput operating point and serves as a shared default across all main experiments.

Table 10: EntropyCache ablation on GSM8K (LLaDA-Inst, $w = 32$). Each cell: Accuracy (%) / Throughput (tok/s).

	$\tau = 0.5$	$\tau = 1.0$	$\tau = 1.5$	$\tau = 2.0$	$\tau = 2.5$
$k = 16$	79.76 / 16.12	79.15 / 30.87	78.47 / 39.44	75.44 / 44.34	74.30 / 46.26
$k = 32$	79.83 / 16.10	79.45 / 30.14	79.68 / 38.84	75.28 / 43.69	74.98 / 45.08
$k = 64$	79.76 / 16.13	79.38 / 30.16	**78.77 / 38.56**	76.12 / 42.04	74.30 / 44.07
$k = 128$	79.83 / 16.06	78.77 / 29.56	78.32 / 37.32	76.12 / 39.76	75.59 / 40.14
$k = 256$	79.83 / 16.04	78.85 / 29.31	78.77 / 35.88	75.74 / 37.44	74.91 / 37.09

Table 11: EntropyCache ablation on MATH500 (LLaDA-Inst, $w = 32$). Each cell: Accuracy (%) / Throughput (tok/s).

	$\tau = 0.5$	$\tau = 1.0$	$\tau = 1.5$	$\tau = 2.0$	$\tau = 2.5$
$k = 16$	41.20 / 17.66	41.40 / 30.81	41.00 / 42.03	39.60 / 47.40	39.20 / 50.49
$k = 32$	40.80 / 17.65	41.40 / 30.44	42.20 / 40.40	43.00 / 46.13	40.60 / 47.90
$k = 64$	41.40 / 17.65	41.20 / 30.64	**42.80 / 39.83**	43.40 / 45.04	39.80 / 46.96
$k = 128$	41.20 / 17.48	41.20 / 30.18	43.00 / 37.78	43.20 / 40.82	41.00 / 42.44
$k = 256$	41.20 / 17.26	41.00 / 29.61	42.80 / 35.45	43.60 / 36.18	40.20 / 35.06

Table 12: EntropyCache ablation on MBPP (LLaDA-Inst, $w = 32$). Each cell: Accuracy (%) / Throughput (tok/s).

	$\tau = 0.5$	$\tau = 1.0$	$\tau = 1.5$	$\tau = 2.0$	$\tau = 2.5$
$k = 16$	42.40 / 33.16	41.80 / 53.91	38.80 / 65.32	40.60 / 70.36	38.60 / 72.36
$k = 32$	42.60 / 32.79	41.80 / 53.75	38.80 / 65.19	41.20 / 68.88	40.60 / 71.97
$k = 64$	42.60 / 32.80	41.40 / 53.03	**38.80 / 63.78**	40.00 / 68.68	40.00 / 69.34
$k = 128$	42.60 / 32.20	41.60 / 50.95	38.80 / 60.85	40.60 / 65.73	40.80 / 65.63
$k = 256$	42.60 / 31.75	41.60 / 50.00	38.60 / 59.14	40.40 / 61.82	41.20 / 61.60

Table 13: EntropyCache ablation on HumanEval (LLaDA-Inst, $w = 32$). Each cell: Accuracy (%) / Throughput (tok/s).

	$\tau = 0.5$	$\tau = 1.0$	$\tau = 1.5$	$\tau = 2.0$	$\tau = 2.5$
$k = 16$	45.73 / 24.55	45.73 / 33.87	47.56 / 38.97	46.34 / 41.73	42.07 / 43.91
$k = 32$	45.73 / 24.32	48.78 / 33.13	46.34 / 37.91	44.51 / 40.64	41.46 / 41.00
$k = 64$	45.73 / 24.34	48.17 / 33.54	**48.78 / 38.10**	49.39 / 39.45	42.07 / 39.46
$k = 128$	45.73 / 24.30	46.95 / 32.95	48.78 / 36.85	45.12 / 37.41	43.90 / 36.29
$k = 256$	45.73 / 24.02	47.56 / 31.85	49.39 / 35.73	47.56 / 33.45	46.34 / 29.19

E Recompute ratio and throughput analysis

We measure the *recompute ratio*—the fraction of KV states recomputed from scratch per step (baseline $= 1.0$)—to disentangle cache reuse from decision overhead. Table 14 reports results for the three dynamic methods.

On GSM8K and MATH500, EntropyCache achieves both the lowest recompute ratio and the highest throughput, indicating that the entropy trigger skips more steps without sacrificing cache freshness. On MBPP (LLaDA), EntropyCache recomputes slightly more than d^{2}Cache (0.178 vs. 0.164) yet delivers 23% higher throughput (64.9 vs. 52.8 tok/s), demonstrating the wall-clock benefit of its low-cost decision rule over d^{2}Cache's context-length-dependent attention rollout. On HumanEval (LLaDA), EntropyCache has a marginally higher recompute ratio and lower throughput than d^{2}Cache, but achieves the highest accuracy among all methods (48.78% vs. d^{2}Cache 41.46%, Table 1), suggesting that entropy-triggered recomputation allocates compute to generation-critical steps rather than minimizing recomputation indiscriminately.

F Memory usage analysis

Figure 9 profiles GPU memory usage over time for the three dynamic KV caching methods on a single GSM8K sample (LLaDA-Inst, $w = 32$, generation length $= 256$). Elastic-Cache consumes the most memory (peak 19.85 GB) because it caches not only key–value pairs but also query projections and hidden states for its per-layer drift comparison. d^{2}Cache and EntropyCache are nearly identical at $\sim$18.85 GB, as neither requires substantial auxiliary storage beyond the KV cache itself. While the absolute differences are modest—model weights and the KV cache dominate the footprint—EntropyCache's auxiliary history costs $O(T)$ memory for T generated positions (one integer per position), independent of prompt length, unlike the context- or depth-dependent buffers of the other methods.

Table 14: Recompute ratio (RR) and throughput (T-put, tok/s) for dynamic KV caching methods. Lower RR indicates more cache reuse. Bold: best per benchmark.

	Elastic-Cache		d^{2}Cache		Ours	
Benchmark	RR	T-put	RR	T-put	RR	T-put
			LLaDA-Inst			
GSM8K	0.140	23.49	0.147	29.96	**0.114**	**38.49**
MATH500	0.148	24.43	0.146	32.18	**0.134**	**39.64**
MBPP	0.243	32.24	**0.164**	52.77	0.178	**64.91**
HumanEval	0.168	27.73	**0.160**	41.36	0.172	38.41
Average	0.175	26.97	0.154	39.06	**0.149**	**45.36**
			Dream-Inst			
GSM8K	0.152	32.15	0.145	41.68	**0.123**	**48.32**
MATH500	0.148	41.90	0.147	55.80	**0.125**	**62.48**
MBPP	0.348	54.84	0.166	87.63	**0.145**	**97.79**
HumanEval	0.281	72.30	0.163	110.83	**0.160**	**119.02**
Average	0.232	50.30	0.155	73.98	**0.138**	**81.90**

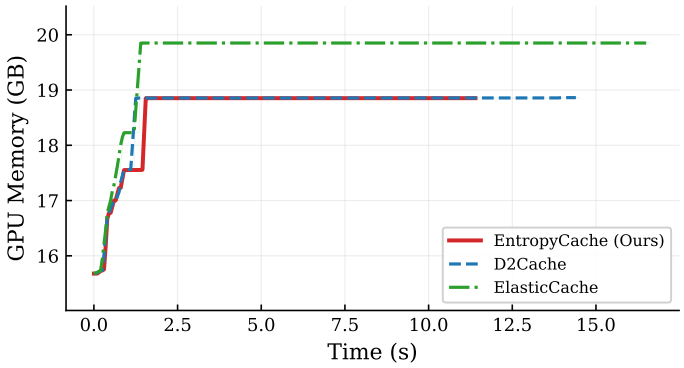

Figure 9: GPU memory usage over time on a single GSM8K sample (LLaDA-Inst, $w = 32$, generation length $= 256$). Elastic-Cache requires $\sim$1 GB more due to extra query and hidden-state caching; d^{2}Cache and EntropyCache are nearly identical.

G Limitations and broader impacts

Limitations. As with any approximate inference acceleration method, EntropyCache exhibits accuracy variations across task–model combinations under a single fixed configuration. Our analyses (Section 5.3, Appendix D.2) demonstrate that EntropyCache maintains favorable accuracy–throughput trade-offs across the operating range and outperforms prior caching methods once τ and k_{recent} are appropriately set. We recommend that practitioners calibrate these hyperparameters to their target workload to obtain the best operating point, particularly for tasks such as code generation where token stability dynamics may differ from natural language reasoning.

Additionally, our evaluation covers two dLLM architectures (LLaDA-8B and Dream-7B), the most widely adopted discrete masked diffusion language models in the current research landscape. All prior KV caching works in this space [8, 11, 23, 10] evaluate on the same two models. As larger or more diverse dLLMs become available, validating EntropyCache at greater scale would be a natural next step.

Broader impacts. By substantially reducing per-request inference cost, EntropyCache can lower the energy and hardware requirements for deploying dLLMs, making diffusion-based generation more accessible in resource-constrained settings. As with any inference acceleration technique, practitioners should verify that approximate caching does not introduce subtle quality regressions in safety-critical applications.

Looking ahead, the entropy-based skipping principle is not limited to the sliding-window decoding setting studied here. Recent block diffusion language models such as Fast-dLLMv2 [22] generate text in coarse blocks that are internally refined through sub-block denoising steps. Since each sub-block undergoes iterative denoising with its own KV dynamics, the entropy trigger proposed in this work could serve as a lightweight recomputation signal within sub-blocks, complementing the block-level KV reuse that these architectures already employ. We leave this extension to future work.

H Sample responses

H.1 Math

Figure 10 shows responses from each method on a GSM8K example requiring multi-step cumulative reasoning (ID: 546, answer: 130). Highlighted tokens in the EntropyCache output mark positions where decoded entropy exceeded τ, triggering full KV recomputation. These tend to coincide with transition words and numerically significant tokens. The remaining methods all arrive at incorrect answers.

H.2 Coding

Figure 11 shows responses from each method on a HumanEval example requiring interval intersection logic (ID: 127). The task asks whether the length of the intersection of two closed integer intervals is a prime number. Highlighted tokens in the EntropyCache output mark positions where decoded entropy exceeded τ, triggering full KV recomputation. These tend to coincide with logical connectives in the reasoning preamble and the assignment operator in the critical formula. Only EntropyCache produces the correct implementation, while the remaining methods all fail the test cases.

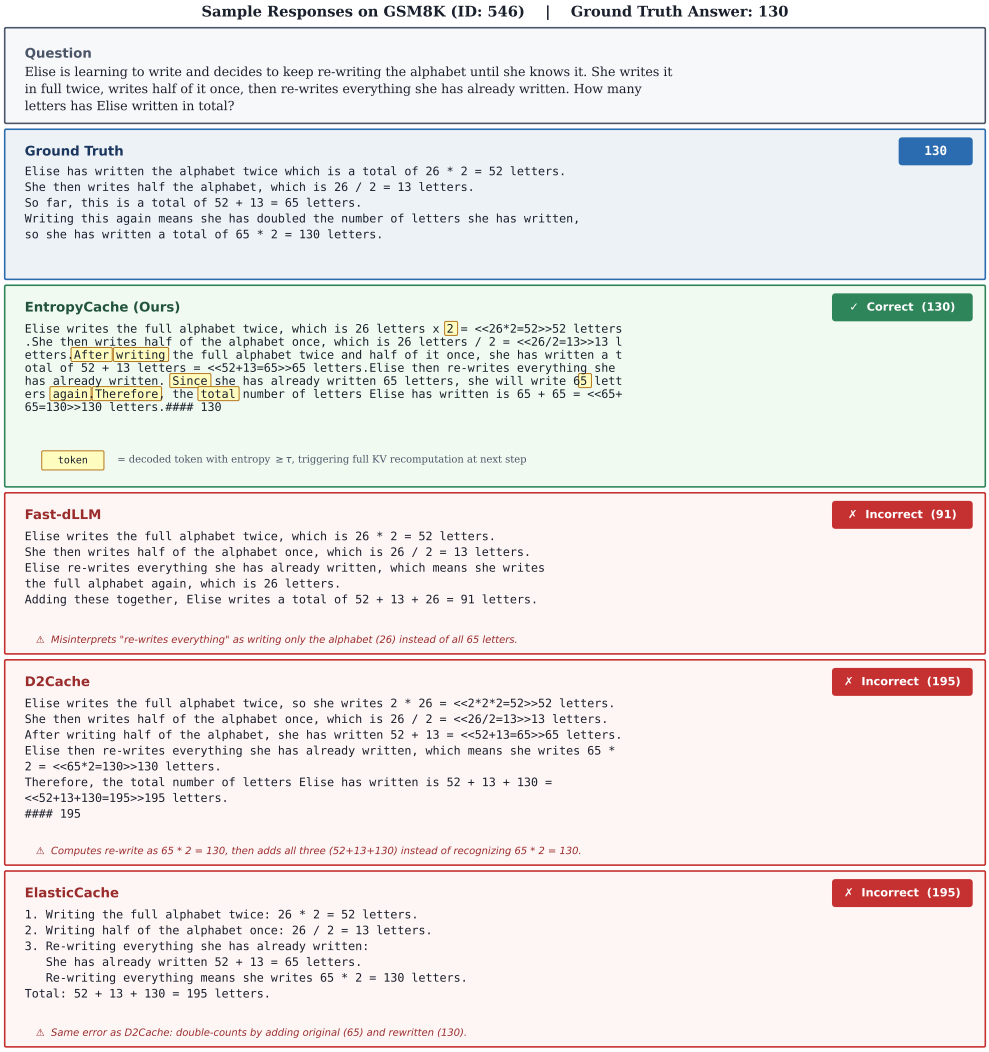

Figure 10: Sample responses on GSM8K (ID: 546, ground truth: 130). Highlighted tokens in the EntropyCache response denote decoded tokens with entropy $\geq \tau$ that triggered full KV recomputation at the next denoising step.

Sample Responses on HumanEval (ID: 127) | Task: intersection(interval1, interval2)



Figure 11: Sample responses on HumanEval (ID: 127). Highlighted tokens in the EntropyCache response denote decoded tokens with entropy $\geq \tau$ that triggered full KV recomputation at the next denoising step.

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 claim (1) a training-free KV caching method using $O(NV)$ entropy computation over N newly decoded token distributions, (2) 15.2×–26.4× speedup on standard benchmarks and 22.4×–24.1× on chain-of-thought benchmarks relative to vanilla dLLM inference, and (3) competitive accuracy across baselines. All three claims are supported by Tables 1–2 and the overhead analysis in Section 5.4. Accuracy degradation on certain benchmarks (e.g., MBPP −3.4pp on LLaDA) is acknowledged in Section 5.2 and Appendices D and G.

   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: Limitations are discussed in Appendix G, including (1) non-trivial accuracy degradation on certain task–model combinations at the default parameters, with a recommendation to calibrate τ and k per workload, and (2) evaluation scope limited to two dLLM architectures (LLaDA-8B and Dream-7B), which are the only widely-adopted discrete masked diffusion LLMs with public checkpoints at this time.

   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: [N/A]

   Justification: This paper makes no theoretical claims and contains no lemmas, theorems, or formal proofs. All contributions are empirical: the entropy–drift correlation is established via Spearman analysis (Section 3.1, Appendix B), and the method's effectiveness is evaluated experimentally (Section 5).

   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: Section 5.1 specifies all experimental details: hardware (NVIDIA RTX 3090), decoding strategy (sliding-window, window size 32), parallel decoding confidence threshold (0.9), and EntropyCache hyperparameters ($\tau = 1.5$, $k = 64$). Baseline hyperparameters follow their respective papers. The complete algorithm is given in Algorithm 1 (main text) and Algorithm 2 (Appendix A). Code is publicly available.

   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: Our de-anonymized implementation and reproduction scripts are publicly available with the camera-ready submission.

   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: Section 5.1 provides full experimental details: GPU hardware, decoding configuration (sliding-window size, confidence threshold), EntropyCache hyperparameters and their selection rationale (grid search in Section 5.3 and Appendix D), baseline hyperparameters with citations to original papers, and benchmark-specific settings (number of shots, generation lengths). No training is performed; this is a training-free inference method.

   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: [No]

   Justification: We only provide the metrics without error bars, following the conventions.

   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: [No]

   Justification: Section 5.1 reports the NVIDIA RTX 3090 GPUs (24GB VRAM) used for the main experiments, and Appendix C.4 reports the RTX 6000 Ada GPUs (48GB VRAM) used for batched decoding. We do not report total GPU-hours for the full research project.

   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: This work proposes an inference acceleration method for existing language models and does not involve human subjects, sensitive data collection, or any application with direct potential for harm. We have reviewed the NeurIPS Code of Ethics and confirm compliance.

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: Appendix G discusses broader impacts, including positive impacts (reduced energy and hardware requirements for dLLM inference, making diffusion-based generation more accessible) and a caution regarding negative impacts (practitioners should verify that approximate caching does not introduce quality regressions in safety-critical applications).

    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 paper releases only inference code (a caching wrapper around existing models) and does not release new pretrained model weights or datasets. The underlying models (LLaDA-8B-Instruct, Dream-7B-Instruct) are governed by their original licenses and release policies.

    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 models and datasets are properly cited. LLaDA-8B-Instruct [12] and Dream-7B-Instruct [25] are used under their respective open-access licenses. Benchmark datasets (GSM8K, MATH500, MBPP, HumanEval, MMLU-Pro, BBH, GPQA) are all publicly released research benchmarks and are cited accordingly.

    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: [Yes]

    Justification: The released code implements EntropyCache and all baseline methods within a unified codebase, with documentation covering installation, reproduction of all reported experiments, and hyperparameter configuration. No new datasets or model weights are released.

    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 paper does not involve crowdsourcing or research with human subjects. All evaluations are automated using standard benchmark scoring.

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: No human subjects are involved in this research.

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: LLMs were not used as an important, original, or non-standard component of the core methodology or experimental pipeline. All experiments, code, and analyses were conducted by the authors; assistance was limited to language editing, for which declaration is not required.

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.