Mask Tokens as Prophet: Fine-Grained Cache Eviction for Efficient dLLM Inference

Jianuo Huang[1,2*] **Yaojie Zhang**[1,3*]
Yicun Yang[1] **Benhao Huang**[4] **Linfeng Zhang**[1†]

[1]School of Artificial Intelligence, Shanghai Jiao Tong University
[2]Huazhong University of Science and Technology
[3]University of Electronic Science and Technology of China
[4]Carnegie Mellon University

{jianuohuang82,yaojiezhang288}@gmail.com

Abstract

Diffusion large language models (dLLMs) present a promising alternative to dominant autoregressive models (ARMs) by the ability of parallel decoding at the expense of substantial computation and memory costs. Specifically, the cache mechanism for bidirectional attention in dLLMs demands large memory footprint, restricting their ability to handle long contexts under resource-limited settings. Existing cache eviction strategies are primarily designed for ARMs and fail to account for the role of mask tokens and specific characteristics in dLLMs, resulting in suboptimal performance. To address these challenges, we introduce *MaskKV*, a training-free cache eviction framework tailored to dLLMs, focusing on the effect of mask tokens in dLLMs. *MaskKV* is built on two key innovations: (1) a mask-query guided scoring mechanism that leverages attention weights to identify and evict less critical prompt tokens for each head; (2) an adaptive cache budgeting strategy that improves efficiency by reducing allocation in intermediate layers and concentrating resources on prompt-preferring heads. On LLaDA with *MaskKV*, compressing the KV cache to only 256 pairs (less than 5% of tokens) retains 94% of the full-cache performance on LongBench and achieves up to 31 × acceleration at 32k prompt length. *Our code will be released on Github.*

1 Introduction

Over the past few years, autoregressive language models have dominated text generation (Zhao et al., 2023), but their strictly left-to-right decoding enforces sequential inference and limits throughput. Diffusion large language models (dLLMs) lift the latency ceiling by iteratively denoising a fully masked sequence, enabling parallel prediction of all tokens with bidirectional attention for richer contextual reasoning (Gemini, 2025; Chen et al., 2025; Wang et al., 2025). Closed-source systems such as Gemini Diffusion (Gemini, 2025) and Mercury (Khanna et al., 2025) demonstrate the production readiness of dLLMs, while open-source models like LLaDA-8B (Nie et al., 2025) and Dream-7B (Ye et al., 2025) confirm their competitiveness with autoregressive counterparts.

The iterative denoising process in dLLMs incurs substantial redundant computation, motivating several cache mechanisms tailored for bidirectional attention (Liu et al., 2025b; Wu et al., 2025; Ma et al., 2025; Hu et al., 2025). Unlike autoregressive models (ARMs), which only maintain and reuse KV states for past tokens, dLLMs update and cache representations for the entire sequence at every denoising step, including the input prompt, generated tokens, and mask tokens. As a result, dLLMs typically reach peak memory as early as the prefill stage, rather than exhibiting gradual memory growth as in ARMs, making cache eviction indispensable for long-context inference. Although cache eviction strategies have been extensively studied for ARMs (Zhang et al., 2023; Li et al., 2024), these methods do not explicitly consider the role of mask tokens or the effects of bidirectional attention in dLLM. Existing work on dLLM cache eviction remains largely exploratory and offers limited analysis of these behaviors.

These fundamental differences render existing strategies ineffective. However, the mechanisms that incur overhead, specifically bidirectional attention and mask tokens, also offer a distinct advantage: a **global perspective** unavailable to ARMs. Unlike the passive lookup in ARMs, mask tokens serve as intrinsic probes, actively identifying the critical context required for reconstruction. This insight allows us to revisit two essential problems: *which tokens should be evicted* and *how to allocate the cache budget*.

Which tokens should be evicted? Existing ARM-based methods rely on historical attention accumu-

[*]Equal contribution. [†]Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 3456–3479

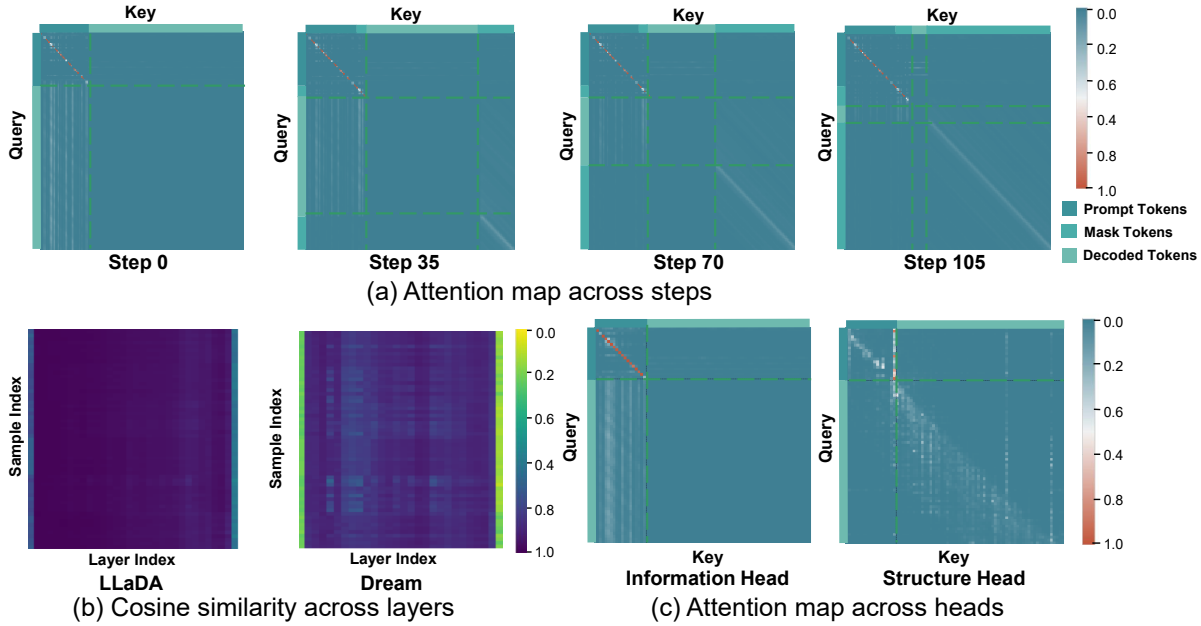

Figure 1: **Visualization on attention maps and features in LLaDA.** (a) The queries from the mask token tend to clearly concentrate on several "important" prompt tokens, indicating they are able to choose the valuable prefix. (b) The first layer and the last layer in diffusion LLMs tend to show significantly lower cosine similarity compared to their adjacent layers, indicating these two layers contribute more than other layers in generation. (c) "Information heads" tend to focus more on the previous prompts while "structure heads" focus more on the mask tokens.

lation. However, our analysis of dLLM attention patterns in Fig. 1 (a) reveals a limitation: intra-prompt attention exhibits strong **positional bias**, primarily focusing on local neighbors (visible as diagonal patterns). Relying solely on this local signal may miss long-range semantic dependencies. In contrast, we observe distinct **vertical attention stripes** originating from the mask tokens, indicating that they consistently attend to a sparse set of pivotal prompt tokens across all denoising steps. this empirical finding is further corroborated by our theoretical proof in Appendix A.8, which establishes that mask-query attention constitutes the *sole* channel for retrieving prompt context in diffusion generation. Capitalizing on this dual validation of **temporal consistency** and **theoretical necessity**, we propose **Mask-Voting**, which aggregates these stable attention scores to accurately pinpoint and retain crucial prompt information.

How to allocate the cache budget? Existing ARM strategies allocate capacity based on historical attention accumulation (Wang et al., 2024; Feng et al., 2024). However, the distinct generative mechanics of dLLMs disrupt these cumulative patterns, necessitating a strategy that respects structural and functional heterogeneity. As visualized in Fig. 1 (b), the iterative denoising process yields a clear bimodal importance profile where feature representations undergo significant transformation in the boundary layers while remaining relatively stable in the middle layers; this suggests that the first and last layers are indispensable for input encoding and output reconstruction, whereas intermediate layers exhibit high redundancy. Simultaneously, at the head level (Fig. 1 c), bidirectional interaction with mask tokens drives distinct specialization into *Information Heads* that focus on prompts and *Structure Heads* that attend primarily to mask tokens. To address this, we propose a **two-stage allocation scheme**: first distributing the budget to boundary layers to preserve representational integrity, and then concentrating the remaining resources on prompt-preferring heads.

Based on the above observations, this paper introduces *MaskKV* as a KV eviction framework tailored to dLLMs, with the following contributions:

1. We analyze the attention behaviors of dLLMs and revealed several useful insights for KV cache eviction, showing how the mask tokens can participate in the judgment of important KV and the allocation of KV budgets.

2. We introduce *MaskKV*, a KV cache eviction framework tailored to dLLMs, which is composed of the mask-query guided token eviction, offline layer-wise budget allocation, and

adaptive head-wise budget redistribution.

3. Extensive experiments on LongBench with LLaDA and Dream show that *MaskKV* substantially reduces memory and computation overhead while preserving accuracy. Specifically, on LLaDA, it reaches 94% of the full-cache performance with the KV cache compressed to only 256 pairs.

2 Related Work

2.1 Diffusion Models for Language

Diffusion large language models (dLLMs) have emerged as a compelling non-autoregressive paradigm for text generation (Li et al., 2025b). LLaDA provides an early demonstration of the scalability of diffusion-based language models (Nie et al., 2025), followed by Dream, which introduces AR-based initialization and context-adaptive noise scheduling to enhance training effectiveness and downstream performance (Ye et al., 2025). The development of dLLMs is now driven by the twin objectives of accelerating inference and improving generation quality. To accelerate inference, Fast-dLLM (Wu et al., 2025) introduces a block-wise approximate KV cache tailored for dLLM, while dLLM-Cache (Liu et al., 2025b) leverages feature caching to reduce redundant computation. SlowFast Sampling (Wei et al., 2025) jointly considers confidence, convergence and position for dynamic decoding, achieving a practical trade-off between speed and quality. For enhancing performance, DEADAL (Li et al., 2025a) addressed the limitation of fixed-length generation by introducing variable-length denoising. LongLLaDA (Liu et al., 2025a) extends diffusion LLMs to long contexts. These advancements have driven the demand for longer-context handling, which under caching mechanisms leads to prohibitive memory overhead.

2.2 KV Cache Compression

The substantial memory footprint of the Key-Value (KV) cache presents a primary bottleneck for long-context inference in Large Language Models (LLMs). Early studies relied on static, content-agnostic heuristics such as sliding windows or attention sinks, which often discard long-range information and motivate the development of content-aware (Beltagy et al., 2020; Xiao et al., 2023), dynamic eviction strategies (Zhang et al., 2023; Li et al., 2024; Devoto et al., 2024). Building on this direction, recent work explores finer-grained and adaptive budget allocation by exploiting structural heterogeneity within the model (Cai et al., 2024; Feng et al., 2024). At the layer level, budgets are allocated using static or adaptive policies based on importance estimates (Cai et al., 2024; Wang et al., 2024). while head-level methods assign differentiated budgets or functional roles to enable selective caching (Feng et al., 2024; Xiao et al., 2024). However, these designs are largely developed for autoregressive models and do not explicitly account for the unique role of mask tokens in dLLMs. Current attempts on dLLMs (Song et al., 2025) remain preliminary, lacking a principled analysis of how mask tokens can participate in identifying important KVs and guiding budget allocation.

3 Methodology

3.1 Preliminaries

Inference of Diffusion Large Language Models. Diffusion language models (dLLMs) generate text through an iterative unmasking process over T discrete decoding steps, progressively refining a fully masked sequence into the final output. We first define the token vocabulary as $\mathcal{T}$, which includes a special mask token $\texttt{[MASK]} \in \mathcal{T}$. For a given prompt $c = (c_1, \ldots, c_M)$, the initial input state is

$$x^{(T)} = [c_1, \ldots, c_M, \underbrace{\texttt{[MASK]}, \ldots, \texttt{[MASK]}}_{L}] \quad (1)$$

Unlike ARMs, which extend the sequence token by token, dLLMs refine the entire sequence in parallel at every step. This sequence includes the prompt, already decoded tokens, and remaining [MASK] tokens. At each step t, the model f_ϕ takes the current state $x^{(t)}$ and predicts a probability distribution over the vocabulary for every masked position. This results in $y = \{y_i \mid i \in M^{(t)}\}$, the set of candidate tokens for all masked locations:

$$P_\phi(y \mid x^{(t)}) = f_\phi(x^{(t)}) \quad (2)$$

Subsequently, a remasking policy π selects tokens to decode or keep masked for the next step:

$$x^{(t-1)} = \pi\big(x^{(t)}, P_\phi(y \mid x^{(t)})\big) \quad (3)$$

The process continues until step $t = 0$, at which point all [MASK] tokens have been replaced to yield the final sequence. This parallel multi-token prediction capability comes at the expense of substantial computational and memory costs.

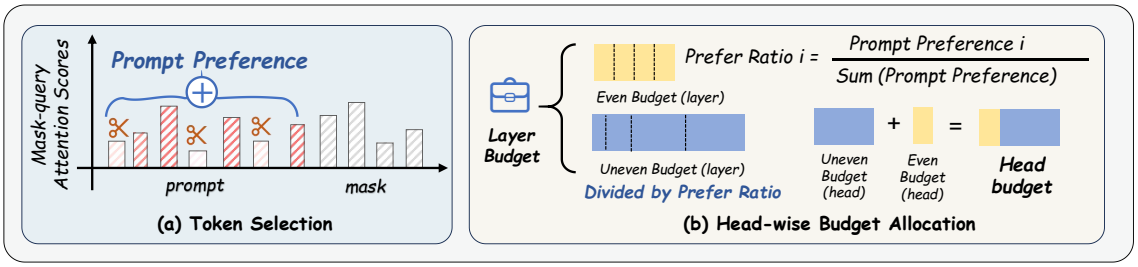

Figure 2: **The MaskKV pipeline.** We use Mask Voting to assess token importance, then apply adaptive budget allocation where layers and heads receive budgets by boundary awareness and prompt preference. Tokens are evicted based on their importance under the given budget.

Caching in Diffusion Large Language Models. Given the initial input $x^{(T)}$ with prompt length M and L masked positions, we denote the prompt and response token sets as $\mathcal{P} = \{1 : M\}$ and $\mathcal{R} = \{M+1 : M+L\}$, respectively. We index layers by $\ell = 1, \ldots, D$. At denoising step t, for layer ℓ and position i, we define the feature bundle:

$$\mathrm{Feat}_\ell^{(t)}(i) = \{K_{\ell,i}^{(t)}, V_{\ell,i}^{(t)}, \mathrm{Attn}_{\ell,i}^{(t)}, \mathrm{FFN}_{\ell,i}^{(t)}\} \quad (4)$$

and denote the corresponding cache entry by $\mathcal{C}_\ell^{(t)}(i)$. The computation of each layer can be expressed as:

$$h_\ell^{(t)} = \mathrm{Attn}_\ell\big(Q_\ell^{(t)}, K_\ell^{(t)}, V_\ell^{(t)}\big) + \mathrm{FFN}_\ell^{(t)} \quad (5)$$

Unlike autoregressive models where causal attention allows past KV states to be directly reused, dLLMs employ bidirectional attention over both prompt and masked positions. Since this operation is repeated for every step and token, it introduces substantial redundancy. Empirically, however, the feature bundles $\mathrm{Feat}_\ell^{(t)}(i)$ exhibit strong similarity across adjacent steps, suggesting opportunities for caching (Liu et al., 2025b; Wu et al., 2025; Ma et al., 2025; Hu et al., 2025).

At each step t, if $i \in \mathcal{S}^{(t)}$, we refresh $\mathcal{C}_\ell^{(t)}(i) = \mathrm{Feat}_\ell^{(t)}(i)$; otherwise, we reuse $\mathcal{C}_\ell^{(t-1)}(i)$. The definition of the refresh set $\mathcal{S}^{(t)}$ differs by token type.

Prompt tokens. For prompt tokens ($i \in \mathcal{P}$), following the temporal caching strategy, we refresh them only every T_p steps. Specifically, the prompt refresh set is defined as $\mathcal{S}_\mathcal{P}^{(t)} = \{i \in \mathcal{P} \mid t \bmod T_p = 0\}$. Between these refreshes, their cached features remain fixed to minimize redundant computation.

Response tokens. We define the refresh set $\mathcal{S}_\ell^{(t)}$ for layer ℓ using a hybrid strategy that combines periodic resets with adaptive drift monitoring. First, a **periodic set** ensures regular global updates: $\mathcal{S}_{\mathrm{period}}^{(t)} = \{i \in \mathcal{R} \mid t \bmod T_r = 0\}$. Second, to capture significant feature variations between intervals, we compute a **drift score** $d_{\ell,i}^{(t)} = 1 - \cos(V_{\ell,i}^{(t)}, V_{\ell,i}^{(t-1)})$. We then identify the subset of tokens with the most substantial changes: $\mathcal{S}_{\mathrm{shift},\ell}^{(t)} = \mathrm{TopK}_{i\in\mathcal{R}}(d_{\ell,i}^{(t)}, \lfloor \rho|\mathcal{R}| \rfloor)$. Finally, the effective refresh set for layer ℓ is the union of these two components: $\mathcal{S}_\ell^{(t)} = \mathcal{S}_{\mathrm{period}}^{(t)} \cup \mathcal{S}_{\mathrm{shift},\ell}^{(t)}$.

3.2 Observations

Our analysis of the dLLM's attention architecture reveals a non-uniform distribution of computational redundancy. This presents a hierarchical structure of optimization opportunities across the layer, head, and token levels.

Layer-Level. We begin at the highest architectural level by quantifying the importance of each layer via its representational transformation (formalized as an **importance score** in Section 3.3). As visualized in Fig. 1 (b), our analysis uncovers two critical phenomena: a distinct **bimodal importance profile** and remarkable **cross-sample consistency**. The bimodal profile indicates that boundary layers undergo significant feature transformation (manifested as lighter colors), whereas middle layers remain relatively static. Furthermore, the vertical uniformity of this pattern across different samples demonstrates that this hierarchy is structurally inherent rather than input-dependent. These findings serve as the cornerstone of our allocation strategy: the bimodal profile necessitates a non-uniform, group-based budget allocation, while the cross-sample consistency validates the feasibility of an efficient, static offline profiling approach.

Head-Level. Building on the layer-wise hierarchy, we examine the behavior of individual attention heads. We observe significant functional heterogeneity, where heads exhibit widely varying degrees of dependency on the prompt context. To illustrate this spectrum, Fig. 1 (c) highlights two contrasting examples representing the extremes: **Information Heads**, which rely heavily on the prompt for long-range information retrieval, and **Structure Heads**, which exhibit negligible attention to the prompt, instead directing their focus primarily toward the mask tokens to organize the local syntactic framework. This functional heterogeneity motivates our fine-grained strategy: a head-wise budget allocation that assigns resources based on each head's reliance on the context.

Token-Level. Finally, to identify which specific tokens to retain within the allocated budgets, we investigate token-level attention patterns. A crucial dichotomy emerges (see Fig. 1 (a)): while non-mask tokens (prompt and decoded tokens) exhibit a strong **locality bias** to encode local context, the attention originating from mask queries is **highly sparse and long-range**. These mask queries function as a task-driven mechanism for global information retrieval. Critically, we observe that this sparse attention pattern remains **remarkably consistent** across generation steps. This insight provides the definitive signal for our framework: the attention scores from mask queries serve as a robust and stable metric for determining prompt token importance, universally applicable across different heads.

3.3 The *MaskKV* Framework

Based on our empirical findings, we propose *MaskKV*, a framework that reframes KV cache pruning as a two-stage process: first, it establishes a universal importance ranking for all tokens, and second, it applies a hierarchical budget to this ranking to perform eviction. The core execution logic of our framework is detailed in appendix A.9.

Stage 1: Universal Token Importance Ranking. We observe that, unlike prompt queries, which exhibit local bias, mask queries perform sparse long-range retrieval with strong cross-step consistency. Consequently, we propose **Mask-Voting**, a one-shot method that leverages the reliable, task-aligned signal from mask queries at the initial inference step.

To capture the global context dependency, we first compute the full attention score matrix A between all mask queries (Q_{mask}) and the entire sequence of keys (K_{full}), including both prompt and mask tokens:

$$A = \text{softmax}\left(\frac{Q_{\text{mask}} K_{\text{full}}^T}{\sqrt{d_k}}\right) \in \mathbb{R}^{n_m \times (n_p + n_m)} \tag{6}$$

We derive the importance score vector $I \in \mathbb{R}^{n_p}$ by aggregating the attention scores for the prompt indices:

$$I_j = \sum_{i=1}^{n_m} A_{ij} \quad \text{for } j \in \{1, \ldots, n_p\} \tag{7}$$

The output I provides a universal, budget-agnostic ranking of all prompt tokens, quantifying their intrinsic importance to the generative task regardless of specific capacity constraints.

Stage 2: Hierarchical Budgeted Eviction. The second stage translates the global ranking from Stage 1 into strict cache constraints using a top-down allocation policy. Echoing the hierarchical redundancy revealed in Section 3.2, we distribute the total budget first across layers and then across heads to maximize representational capability.

Offline Layer-Level Allocation. To determine the optimal budget distribution, we first establish a quantitative metric for layer importance. We define the importance score $S^{(l)}$ of the l-th layer by measuring the cosine dissimilarity between its input and output hidden states:

$$S^{(l)} = 1 - \frac{1}{T} \sum_{i=1}^{T} \text{cos\_sim}\left(h_{in,i}^{(l)}, h_{out,i}^{(l)}\right) \tag{8}$$

This metric captures the magnitude of representation updates contributed by each layer.

Our strategy is driven by two key empirical observations regarding $S^{(l)}$ (detailed in Section 3.2):

- **Cross-Sample Consistency:** The importance profile remains stable across different inputs. This motivates us to adopt an **offline profiling** strategy (Algorithm 2), where budgets are determined *a priori* on a calibration set. This avoids the limitation of online methods, which require full cache instantiation for measurement and thus fail to reduce peak memory.
- **Bimodal Profile:** Boundary layers exhibit high importance while middle layers are relatively quiescent. To leverage this structural prior, we employ a **group-smoothed allocation** policy that mitigates layer-wise noise.

Given a target average budget B, we reserve a fixed base $k_{\text{base}}^{\text{layer}} = \lfloor \beta \cdot B \rfloor$ for stability. The remaining capacity forms a shared pool $C_{\text{pool}} = (B - k_{\text{base}}^{\text{layer}}) \cdot L$. We then compute a **group-smoothed score** $\hat{S}^{(l)}$ by assigning the average importance of the corresponding group (Boundary or Middle) to each layer: $\hat{S}^{(l)} = \text{mean}(\{S^{(k)} \mid k \in \text{Group}(l)\})$. The final budget B_l is distributed based on this smoothed profile:

$$B_l = k_{\text{base}}^{\text{layer}} + \underbrace{\left\lfloor C_{\text{pool}} \cdot \frac{\hat{S}^{(l)}}{\sum_{j=1}^{L} \hat{S}^{(j)}} \right\rfloor}_{\text{Importance-Driven Reallocation}} \tag{9}$$

This formulation ensures identical budgets within each group, enforcing the bimodal prior while adhering to the global capacity constraint.

Online Head-Level Allocation. To address the functional heterogeneity of heads (Section 3.2), we employ a dynamic, sample-aware **Prompt Preference** allocation (Stage 2 of Algorithm 3).

Let B_l be the target average budget per head. We define the preference ρ_h as the ratio of a head's prompt attention mass $\mathcal{A}_{\text{prompt}}^{(h)}$ to its total capacity $\mathcal{A}_{\text{total}}^{(h)}$:

$$\rho_h = \frac{\mathcal{A}_{\text{prompt}}^{(h)}}{\mathcal{A}_{\text{total}}^{(h)}} = \frac{\sum_{j \in \text{Prompt}} I_{h,j}}{N_m} \tag{10}$$

Since the total capacity $\mathcal{A}_{\text{total}}^{(h)}$ is constant (equal to the mask query count N_m), normalizing ρ_h simplifies to normalizing the prompt attention mass:

$$P_h^{(l)} = \frac{\mathcal{A}_{\text{prompt}}^{(h)}}{\sum_{h'=1}^{N_h} \mathcal{A}_{\text{prompt}}^{(h')}} \tag{11}$$

We then apply a guarantee-plus-surplus policy. Reserving a base $k_{\text{base}} = \lfloor \alpha B_l \rfloor$, the remaining pool $C_{\text{pool}}^{(l)} = (B_l - k_{\text{base}}) N_h$ is redistributed based on preference:

$$k_{l,h} = k_{\text{base}} + \lfloor C_{\text{pool}}^{(l)} \cdot P_h^{(l)} \rfloor \tag{12}$$

A visualization of prompt attention mass across heads is provided in Figure 9. Final eviction is performed by selecting the top-$k_{l,h}$ tokens based on the global ranking I.

4 Experiment

4.1 Experiment Settings

Implementation Details. We evaluate the effectiveness of our method on two representative dLLMs characterized by **distinct training and inference paradigms**: LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025). All experiments were conducted on 8 × NVIDIA A100 80GB GPUs. Additional details are provided in Appendix A.2.

Method	Qs	HQ	PRe	Sam
LLaDA	16.96	14.68	97.44	40.51
TPS (↑)	$2.85_{1.0\times}$	$1.07_{1.0\times}$	$1.04_{1.0\times}$	$1.62_{1.0\times}$
Mem. (↓)	$17.54_{1.0\times}$	$21.09_{1.0\times}$	$21.18_{1.0\times}$	$19.21_{1.0\times}$
+ dCache	15.26	13.87	97.44	41.86
TPS (↑)	$12.11_{4.2\times}$	$5.73_{5.4\times}$	$5.64_{5.4\times}$	$8.10_{5.0\times}$
Mem. (↓)	$21.31_{1.2\times}$	$29.98_{1.4\times}$	$30.23_{1.4\times}$	$25.65_{1.3\times}$
+ Fast	15.45	14.32	98.25	41.04
TPS (↑)	$13.16_{4.6\times}$	$7.54_{7.0\times}$	$7.46_{7.2\times}$	$10.11_{6.2\times}$
Mem. (↓)	$21.24_{1.2\times}$	$30.57_{1.4\times}$	$30.81_{1.5\times}$	$25.75_{1.3\times}$
+ Sparse	14.17	13.72	94.96	**41.90**
TPS (↑)	$13.54_{4.8\times}$	$6.65_{6.2\times}$	$6.60_{6.3\times}$	$9.21_{5.7\times}$
Mem. (↓)	$18.51_{1.1\times}$	$23.26_{1.1\times}$	$23.38_{1.1\times}$	$20.87_{1.1\times}$
+ MaskKV	**22.71**	**14.81**	**98.92**	40.48
TPS (↑)	$\mathbf{13.63}_{4.8\times}$	$\mathbf{10.51}_{9.8\times}$	$\mathbf{10.66}_{10.3\times}$	$\mathbf{12.73}_{7.9\times}$
Mem. (↓)	$\mathbf{16.06}_{0.9\times}$	$\mathbf{16.78}_{0.8\times}$	$\mathbf{16.81}_{0.8\times}$	$\mathbf{16.51}_{0.9\times}$

Table 1: Performance between dLLM-Cache, Fast-dLLM, Sparse-dLLM, MaskKV on LLaDA-Instruct. Dataset abbreviations: **Qs** (Qasper), **HQ** (HotpotQA), **PRe** (PRe), and **Sam** (Samsum).

4.2 Main Results

Preserving Accuracy under Constrained Cache As shown in Tab. 2, our method consistently surpasses prior cache-eviction strategies on specific budgets. Under the extreme 32 KV budget, it outperforms the best competing baseline by 7.02 points on LLaDA-8B. Notably, our method can even surpass the full-context dLLM-Cache baselines, which we analyze in detail in Appendix A.3.

Stable Performance across Varied Budgets. As shown in Fig. 3, under a 256 KV budget, our method preserves 94.33% of the dLLM-Cache performance on **LLaDA** and 98.66% on **Dream**, achieving the best overall results. This advantage is consistent across all KV budgets and remains robust even in extremely low-budget regimes where autoregressive models typically fail (Xiao et al., 2023). We attribute this robustness to bidirectional attention, which enables more informative KV representations and thus supports aggressive pruning while maintaining generation quality.

Comparison with dLLM-Specific Optimizations. Tab. 1 demonstrates that *MaskKV* establishes a superior Pareto frontier compared to state-of-the-art dLLM-tailored strategies. Unlike *dLLM-Cache* and

Table 2: LongBench results for LLaDA-8B and Dream-7B with specific KV cache budgets (B=32 and B=128). Best result in each column within a budget section is in **bold**.

Method	Single-Doc. QA		Multi-Doc. QA			Summarization			Few-shot Learning			Synthetic		Code		Ave. Score
	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PRe	Lcc	RB-P		
LLaDA-8B-Instruct																
Full KV Cache																
dLLM w/o Cache	16.96	31.31	14.68	17.60	11.48	29.24	21.93	27.58	65.20	47.98	40.51	98.17	65.69	59.57	39.14	
dLLM w/ Cache	15.26	29.62	13.87	17.17	10.44	29.75	22.06	26.68	66.00	44.94	41.86	97.44	66.07	59.34	38.61	
B=32																
SnapKV	9.10	17.49	**17.38**	16.45	8.06	**9.92**	12.21	13.95	39.25	54.32	16.41	55.00	39.90	29.08	24.18	
PyramidKV	9.90	12.12	14.46	14.62	8.18	9.10	8.83	12.36	26.42	53.26	14.51	28.00	39.18	27.66	19.90	
SqueezeAttention	11.49	14.72	16.66	15.42	8.15	9.28	10.49	14.86	43.50	53.67	15.73	52.00	33.60	25.56	23.22	
AdaKV	11.15	16.21	16.69	**16.98**	7.48	9.10	10.44	14.23	39.17	**55.69**	19.00	59.50	37.21	26.40	24.23	
MaskKV (Ours)	**14.61**	**24.45**	17.05	15.68	**12.50**	9.54	**14.33**	**16.43**	**40.42**	54.64	**29.28**	**90.33**	**56.08**	**42.21**	**31.25**	
B=128																
SnapKV	17.42	26.65	15.88	17.44	7.99	11.50	11.69	18.89	50.83	55.60	20.72	80.00	54.90	39.33	30.63	
PyramidKV	15.71	25.20	16.22	16.20	8.47	10.09	11.06	17.22	39.92	55.03	23.49	83.25	54.00	40.50	29.74	
SqueezeAttention	14.46	17.64	**18.12**	**17.97**	8.00	13.46	10.69	19.46	52.67	54.71	17.04	71.00	48.30	32.51	28.29	
AdaKV	19.42	24.68	17.06	17.57	8.82	11.85	9.51	18.49	49.92	**57.90**	21.03	78.00	54.64	36.27	30.37	
MaskKV (Ours)	**20.21**	**29.84**	15.78	16.65	**11.83**	**13.60**	**17.67**	**20.78**	**57.00**	46.06	**37.28**	**98.17**	**61.61**	**51.86**	**35.60**	
Dream-v0-Instruct-7B																
Full KV Cache																
dLLM w/o Cache	28.17	36.23	27.65	32.43	11.83	5.04	14.29	5.95	73.00	89.25	37.84	16.92	38.91	45.08	33.04	
dLLM w/ Cache	26.55	39.86	27.66	32.09	11.12	4.40	13.89	5.51	73.50	89.59	36.07	12.05	39.88	45.57	32.70	
B=32																
SnapKV	17.75	26.82	22.39	**27.91**	7.53	**2.65**	**12.58**	1.95	28.25	66.14	23.67	17.50	22.75	23.87	21.55	
PyramidKV	14.55	24.51	22.41	15.27	6.90	2.62	11.89	2.15	28.00	57.55	24.17	11.50	22.02	22.22	18.98	
SqueezeAttention	17.62	30.44	20.07	26.15	7.42	2.62	11.83	2.19	26.25	72.17	24.52	17.00	20.91	23.41	22.00	
AdaKV	17.09	25.42	23.64	24.77	7.55	2.58	12.46	1.89	27.75	69.77	25.21	18.83	21.67	24.98	21.69	
MaskKV (Ours)	**18.53**	**33.76**	**32.92**	24.53	**11.02**	2.59	11.98	**2.25**	**33.25**	**86.47**	**27.55**	**20.00**	**24.89**	**27.40**	**25.51**	
B=128																
SnapKV	**22.36**	39.02	30.28	32.27	11.51	2.72	**13.20**	2.72	38.00	87.17	28.69	**26.05**	**32.55**	36.51	28.79	
PyramidKV	17.31	36.59	25.21	24.38	10.99	2.77	12.75	2.90	39.75	84.83	29.71	24.00	30.26	30.33	26.56	
SqueezeAttention	19.36	36.19	28.35	29.79	10.25	2.81	13.11	2.64	33.00	85.99	29.25	25.00	27.11	31.92	26.77	
AdaKV	21.97	**39.68**	33.24	31.57	11.38	2.69	13.15	2.80	38.75	88.51	30.43	24.25	31.46	**37.28**	29.08	
MaskKV (Ours)	21.49	39.23	**38.25**	**33.99**	**14.76**	**2.86**	12.48	**3.19**	**53.50**	**88.69**	**30.97**	21.92	30.71	34.79	**30.49**	

Fast-dLLM which incur heavy memory overhead ($1.2\times \sim 1.5\times$), *MaskKV* is the unique method achieving true memory reduction ($\mathbf{0.8}\times \sim \mathbf{0.9}\times$) while delivering substantial speed gains, such as a $\mathbf{10.3}\times$ acceleration on PRe (vs. $7.2\times$ for *Fast-dLLM*). Moreover, on the Qasper dataset, *MaskKV* attains a score of **22.71**, significantly outperforming the sparsification-based *Sparse-dLLM* (14.17), confirming that our method accurately identifies and preserves critical evidence.

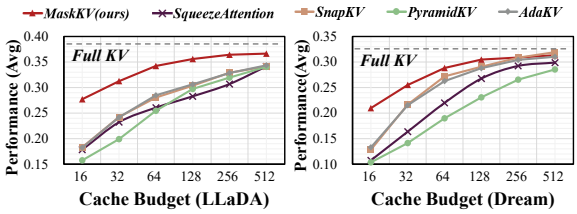

Figure 3: Average LongBench performance across varying KV cache sizes and KV cache eviction methods.

Efficiency in Speed and Memory. Our method markedly improves throughput and memory efficiency for long-context inference. To eliminate engineering bottlenecks present in the original implementation, we incorporate two optimizations: *Prompt-State Exclusion* and *Mask-Only Projection* (details in Appendix A.2). **These adjustments address implementation inefficiency without altering the model's output logic, and are orthogonal to the core cache eviction mechanism, without compromising comparison fairness** (see Appendix A.2). With these techniques, the memory footprint of *MaskKV* is significantly reduced. At a 32K context, it achieves $\mathbf{31}\times$ faster decoding and **65%** lower peak memory than LLaDA, supporting up to $\mathbf{8}\times$ longer context on an RTX 4090 GPU. Ablation results separating the impact of eviction from these optimizations are provided in Tab. 5.

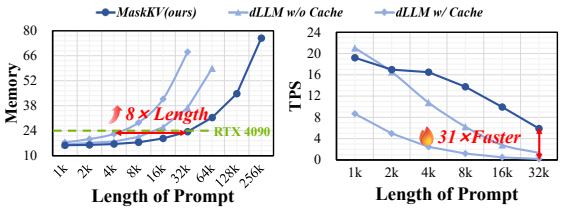

Figure 4: Analysis on latency and memory reduction.

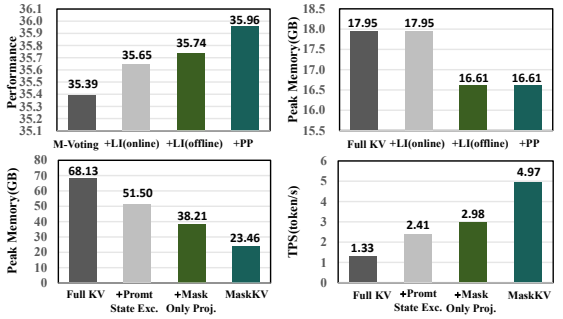

Figure 5: Ablation analysis of effectiveness and efficiency. The top row reports effectiveness results on LongBench. The bottom row reports efficiency results under a 32K-token context.

4.3 Ablation Study

Mask Voting and Budget Allocation. Our Mask Voting consistently outperforms other token selection methods by directly leveraging mask queries, effectively mitigating local bias and identifying influential tokens (Tab. 13). Built upon Mask-Voting (M-Voting), the proposed *Layer-Importance (LI)* layer budget allocation and *Prompt-Preference (PP)* head budget allocation further yield consistent gains over competing approaches (Tab. 12, Fig. 5), demonstrating that each component contributes positively to the final performance and efficiency. These improvements incur minimal overhead: offline budget allocation requires a fixed preprocessing cost of approximately 12 seconds on a single A100 GPU, while online budget allocation and cache eviction account for only 1.3% of the total inference latency, making the proposed design practical for deployment.

Ablation of Efficiency Gains. Fig. 5 dissects the efficiency improvements under a 32K-token context. We progressively evaluate the impact of each component, starting from the Full KV baseline. As shown in Fig. 5, while Prompt-State Exclusion and Mask-Only Projection help eliminate implementation overheads, **the most substantial efficiency leap stems from our cache eviction strategy**. Building upon this optimized foundation,

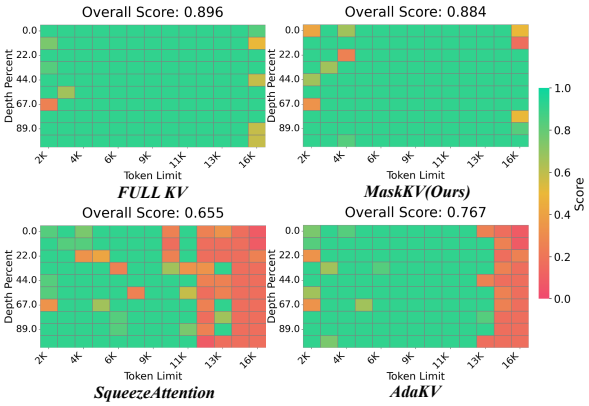

Figure 6: A visualization from the Needle-in-a-Haystack test. See Figure 7 for full results.

MaskKV further compresses the memory footprint and drastically boosts TPS. These results confirm that while system-level fixes are beneficial, the core eviction mechanism is the primary driver of the superior long-context efficiency.

5 Discussion

Needle-in-a-Haystack. To better understand the model's ability to retrieve fine-grained information hidden within long contexts, we conduct a preliminary study using the "needle-in-a-haystack" setup (see Fig. 6). Our method shows stronger robustness to increasing prompt length, effectively maintaining information retrieval performance.

Robustness of Calibration. We evaluate calibration robustness from both data scale and domain perspectives. Varying the size of the calibration set results in comparable downstream task performance, indicating that the proposed method is not sensitive to calibration scale (Tab. 7). We further construct calibration sets from different LongBench task categories and observe consistently similar performance across domains (Tab. 8). These results suggest that our method generalizes well beyond the calibration domain.

6 Conclusion

We explored dLLM characteristics and introduced MaskKV, a training-free framework enabling fine-grained cache eviction via Mask Voting and adaptive layer–head budget allocation. Experiments show that MaskKV reduces the KV cache to 256 tokens while retaining up to 94% of original performance, highlighting an efficient trade-off for long-context inference.

7 Limitations

The current study primarily investigates cache eviction in dLLMs. Given that diffusion-based paradigms are widely utilized in visual generation, the proposed *MaskKV* is theoretically compatible with certain generative frameworks such as MaskGIT or MAR, though its empirical effectiveness remains an open question. Due to the inherent differences in representation between linguistic tokens and visual features, direct cross-domain application may require further structural analysis and adaptation.

Acknowledgments

This project is sponsored by CCF-Tencent Rhino-Bird Funds.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and 1 others. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Xinhua Chen, Sitao Huang, Cong Guo, Chiyue Wei, Yintao He, Jianyi Zhang, Hai Li, Yiran Chen, and 1 others. 2025. Dpad: Efficient diffusion language models with suffix dropout. *arXiv preprint arXiv:2508.14148*.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. 2024. A simple and effective l_2 norm-based strategy for kv cache compression. *arXiv preprint arXiv:2406.11430*.

Alexander R Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R Radev. 2019. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. *arXiv preprint arXiv:1906.01749*.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. 2024. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550*.

Gemini. 2025. Gemini diffusion, our state-of-the-art, experimental text diffusion model.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. *arXiv preprint arXiv:1911.12237*.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. 2023. Longcoder: A long-range pre-trained language model for code completion. In *International Conference on Machine Learning*, pages 12098–12107. PMLR.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

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

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. *arXiv preprint arXiv:2104.02112*.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.

Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, and 1 others. 2025. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*.

Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328.

Jinsong Li, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Jiaqi Wang, and Dahua Lin. 2025a. Beyond fixed: Training-free variable-length denoising for diffusion large language models. *arXiv preprint arXiv:2508.00819*.

Tianyi Li, Mingda Chen, Bowei Guo, and Zhiqiang Shen. 2025b. A survey on diffusion language models. *arXiv preprint arXiv:2508.10875*.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2023. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*.

Xiaoran Liu, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. 2025a. Longllada: Unlocking long context capabilities in diffusion llms. *arXiv preprint arXiv:2506.14429*.

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

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

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

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

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. 2025. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. *arXiv preprint arXiv:2508.09192*.

Zihao Wang, Bin Cui, and Shaoduo Gan. 2024. Squeezeattention: 2d management of kv-cache in llm inference via layer-wise optimal budget. *arXiv preprint arXiv:2404.04793*.

Qingyan Wei, Yaojie Zhang, Zhiyuan Liu, Dongrui Liu, and Linfeng Zhang. 2025. Accelerating diffusion large language models with slowfast: The three golden principles. *arXiv preprint arXiv:2506.10848*.

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

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2024. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

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

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

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

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and 1 others. 2021. Qmsum: A new benchmark for query-based multi-domain meeting summarization. *arXiv preprint arXiv:2104.05938*.

A Appendix

A.1 Details of *MaskKV*

Calibration Set. To configure our offline layer allocation, we construct a calibration set derived from LongBench (Bai et al., 2023). Specifically, we randomly sample 10 instances from 6 diverse tasks, yielding a total of 60 samples. We compute the layer importance scores during the prefill phase for each sample and average them to derive the final importance profile for budget distribution.

Prompt-state Exclusion. In dLLM-Cache, features from both the prompt and response tokens (including keys, values, attention outputs, and MLP activations) are cached at each denoising step. However, we observe that only the key–value representations of prompt tokens contribute to the attention computation of mask tokens, while the prompt-side attention and MLP outputs have no downstream influence. We therefore exclude these redundant prompt features from caching and retain only their key–value pairs, which substantially reduces memory usage without affecting generation quality.

Mask-only Projection. In the official LLaDA implementation, after the final layer computation, the model projects all tokens (including both prompt and mask positions) into the vocabulary space to produce logits. This operation yields a large but unnecessary tensor, as the logits of prompt tokens are never used during decoding. We thus restrict the vocabulary projection to masked positions only and skip the prompt ones. This *mask-only projection* optimization removes redundant matrix multiplications and further reduces GPU memory consumption.

A.2 Implementation Details

Fairness in Evaluation. We implemented all methods, including MaskKV and all baselines, within a unified inference framework to ensure a rigorous apples-to-apples comparison. All engineering implementations and optimizations were kept identical across all methods, with the sole differences being the token scoring mechanisms and budget allocation strategies. Furthermore, to ensure the baselines were tuned fairly, we strictly adopted the optimal hyperparameters recommended in their original papers for all baseline methods. This setup guarantees that the performance differences are attributable to the methods' adaptability to dLLMs rather than engineering discrepancies.

Evaluation Metrics. We evaluate both the efficiency and quality of our method using quantitative metrics. Generation quality is assessed by the official task-specific metrics (see Tab. 6 for details) of LongBench, which measures model accuracy under cache eviction. Computation efficiency is reported in Tokens Per Second (TPS), reflecting the average number of tokens decoded per second. For memory efficiency, we track both the peak GPU memory during inference.

Baseline. We compare one token-selection scheme and three architectural budget-allocation policies under an identical cache budget, enabling a fair, apples-to-apples assessment of their effectiveness.
For the token selection strategy, we evaluate one strong KV-cache compression method as a baseline.

- **SnapKV** uses a small "observation window" at the end of the prompt to predict which parts of the entire context are most important. It analyzes the attention scores from this window in "voting" mechanism to identify and select these key parts.

For architectural budget allocation, we evaluate competitive approaches that distribute the budget across the model's various structural components.

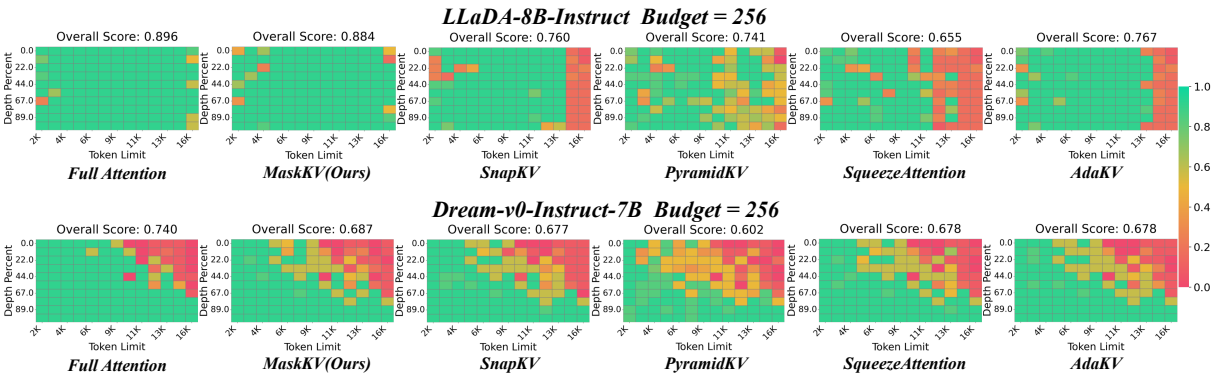

Figure 7: Performance comparison of different KV Cache compression techniques on LLaDA-8B and Dream-7B models in the "Needle-in-a-Haystack" test (Budget B=256). The heatmaps show retrieval accuracy at different context lengths (x-axis) and depths (y-axis), where greener colors indicate better performance.

- **PyramidKV** implements a static, non-uniform budget allocation where the cache capacity of each layer is a direct function of its depth. This function is engineered to be monotonically decreasing, granting the maximal budget to the lowest layers and progressively constricting it for higher layers that process more semantically aggregated representations.

- **SqueezeAttention** gauges layer importance by calculating the cosine similarity between the input and output of an attention block. Based on this score, it classifies layers into three tiers and assigns a minimal cache budget to the least important one.

- **Ada-KV** allocates its cache budget in a fine-grained, adaptive manner: it first assesses the relative importance of each key–value (KV) pair across all attention heads, then distributes the budget proportionally, granting a larger share of resources to KVs belonging to the most salient heads.

Parameters. Our experimental parameters are configured in accordance with prior research(Li et al., 2024; Cai et al., 2024; Wang et al., 2024; Feng et al., 2024). The specific settings are as follows:

- **Default Selection Method**: Unless otherwise specified, we adopt **SnapKV** as the foundational selection method for all budget allocation strategies. For SnapKV (Li et al., 2024) itself, the final window size is set to **32**, consistent with its application in the LongBench benchmark.

- **Pyramid-based Allocation**: We set the hyperparameter β, which directly controls the "steepness" of the allocation pyramid, to **20**, adhering to the default value proposed in the original paper (Cai et al., 2024).

- **SqueezeAttention**: We cluster the layers into three distinct groups. A **40%** budget is allocated to the least important group, a setting identified as optimal in its original study.

- **AdaKV**(Feng et al., 2024): We reserve a **20%** budget for uniform allocation. This measure is implemented to prevent the assignment of excessively small budgets to highly sparse attention heads.

Experiment settings. To ensure reproducibility, we outline our experimental settings. Unless otherwise specified, our default configuration sets the prompt refresh interval to 50, the response refresh interval to 5, the transfer ratio to 0.25, and the block length to 8. The step size is set equal to the generation length, which is specified for each task in Table 6. For the results presented in Fig. 3 and the ablation study in Tab. 12, the KV cache budget is set to 256. The experiment in Fig. 8 is conducted on the HotpotQA dataset with a budget of 32. For the NIAH baseline, we adopt the same configuration as that used in DuoAttention (Xiao et al., 2024).

Datasets. We conducted evaluations using LongBench (Bai et al., 2023). The LongBench benchmark (Bai et al., 2023) evaluates large language models across a diverse set of long-context tasks. The benchmark is structured into six key domains:

Table 3: Detailed LongBench Results for LLaDA-8B-Instruct. Best result in each column within a budget section is in **bold**.

Method	Single-Doc. QA		Multi-Doc. QA			Summarization			Few-shot Learning			Synthetic	Code		Ave. Score
	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PRe	Lcc	RB-P	
LLaDA-8B-Instruct															
Full KV Cache															
dLLM w/o Cache	16.96	31.31	14.68	17.60	11.48	29.24	21.93	27.58	65.20	47.98	40.51	98.17	65.69	59.57	39.14
dLLM w/ Cache	15.26	29.62	13.87	17.17	10.44	29.75	22.06	26.68	66.00	44.94	41.86	97.44	66.07	59.34	38.61
B=16															
SnapKV	10.66	12.28	12.96	13.18	4.91	7.59	9.85	11.80	21.25	49.98	14.37	31.50	30.75	24.17	18.23
PyramidKV	7.94	8.34	10.95	8.77	4.80	7.61	9.02	10.98	14.75	46.29	12.71	24.00	30.48	23.94	15.76
SqueezeAttention	8.27	10.49	14.28	14.19	5.47	7.27	7.92	11.98	31.25	42.39	13.83	30.25	29.49	22.73	17.84
AdaKV	9.48	11.97	12.13	**15.16**	5.62	7.55	**11.34**	11.90	25.00	49.16	14.24	31.58	28.63	22.81	18.33
MaskKV (Ours)	**16.11**	**19.81**	**18.24**	14.16	**11.40**	**7.86**	8.31	**13.69**	**28.50**	**52.32**	**23.29**	**87.00**	**50.09**	**37.32**	**27.72**
B=32															
SnapKV	9.10	17.49	**17.38**	16.45	8.06	**9.92**	12.21	13.95	39.25	54.32	16.41	55.00	39.90	29.08	24.18
PyramidKV	9.90	12.12	14.46	14.62	8.18	9.10	8.83	12.36	26.42	53.26	14.51	28.00	39.18	27.66	19.90
SqueezeAttention	11.49	14.72	16.66	15.42	8.15	9.28	10.49	14.86	43.50	53.67	15.73	52.00	33.60	25.56	23.22
AdaKV	11.15	16.21	16.69	**16.98**	7.48	9.10	10.44	14.23	39.17	**55.69**	19.00	59.50	37.21	26.40	24.23
MaskKV (Ours)	**14.61**	**24.45**	17.05	15.68	**12.50**	9.54	**14.33**	**16.43**	**40.42**	54.64	**29.28**	**90.33**	**56.08**	**42.21**	**31.25**
B=64															
SnapKV	13.77	23.25	17.12	**18.74**	8.74	10.55	10.78	16.67	46.83	56.95	19.79	73.00	48.94	33.75	28.49
PyramidKV	12.23	21.13	15.95	16.90	8.64	9.42	9.24	15.02	37.25	56.70	18.70	51.00	49.73	34.53	25.46
SqueezeAttention	11.30	17.71	17.68	18.32	9.49	11.20	11.60	17.61	44.79	54.58	16.51	66.00	38.64	29.51	26.07
AdaKV	13.21	22.39	**19.60**	17.56	8.25	10.50	9.25	17.25	47.17	**57.65**	20.15	71.50	47.88	30.44	28.06
MaskKV (Ours)	**18.48**	**27.42**	19.00	16.04	**10.12**	**10.93**	**17.25**	**18.88**	**55.08**	50.98	**33.47**	**95.08**	**59.71**	**46.80**	**34.23**
B=128															
SnapKV	17.42	26.65	15.88	17.44	7.99	11.50	11.69	18.89	50.83	55.60	20.72	80.00	54.90	39.33	30.63
PyramidKV	15.71	25.20	16.22	16.20	8.47	10.09	11.06	17.22	39.92	55.03	23.49	83.25	54.00	40.50	29.74
SqueezeAttention	14.46	17.64	**18.12**	**17.97**	8.00	13.46	10.69	19.46	52.67	54.71	17.04	71.00	48.30	32.51	28.29
AdaKV	19.42	24.68	17.06	17.57	8.82	11.85	9.51	18.49	49.92	**57.90**	21.03	78.00	54.64	36.27	30.37
MaskKV (Ours)	**20.21**	**29.84**	15.78	16.65	**11.83**	**13.60**	**17.67**	**20.78**	**57.00**	46.06	**37.28**	**98.17**	**61.61**	**51.86**	**35.60**
B=256															
SnapKV	19.74	27.33	16.62	**18.41**	**10.35**	12.12	12.03	21.00	47.63	53.29	26.53	92.92	58.93	42.95	32.85
PyramidKV	18.50	29.07	14.61	16.05	8.92	12.65	12.15	19.22	43.17	52.17	29.09	91.33	56.54	43.89	31.95
SqueezeAttention	19.35	23.79	16.69	16.35	8.78	13.75	11.24	21.43	51.50	**54.12**	21.48	77.25	57.08	36.91	30.69
AdaKV	19.52	27.73	16.14	17.36	9.89	12.12	11.41	20.13	51.25	53.88	25.87	96.25	58.30	40.78	32.90
MaskKV (Ours)	**22.71**	**29.72**	**14.81**	16.77	9.19	**16.69**	**20.18**	**23.14**	**62.25**	41.65	**40.48**	**98.92**	**61.23**	**52.20**	**36.42**
B=512															
SnapKV	19.98	30.03	12.89	16.93	**10.32**	15.66	13.21	23.36	49.17	52.09	32.71	98.00	61.24	45.37	34.35
PyramidKV	19.53	29.23	14.06	14.75	9.31	15.45	13.93	21.81	52.67	46.03	35.28	**100.00**	59.28	45.58	34.07
SqueezeAttention	**20.71**	26.45	**16.44**	**17.70**	8.04	14.69	12.23	23.54	55.92	50.13	29.29	96.75	61.08	45.01	34.14
AdaKV	19.65	**31.77**	13.47	16.10	9.97	14.53	13.41	22.34	50.25	52.06	33.31	96.00	60.95	44.27	34.15
MaskKV (Ours)	17.85	28.92	13.85	17.10	8.93	**18.58**	**20.29**	**25.05**	**64.25**	**41.62**	**41.04**	99.33	**62.46**	**53.78**	**36.65**

- **Single-Document QA:** Assesses a model's ability to extract answers from a single source document. This category utilizes datasets such as NarrativeQA (Kočiskỳ et al., 2018), Qasper (Dasigi et al., 2021), and MultiFieldQA (Bai et al., 2023), covering documents ranging from academic papers and legal files to encyclopedias.

- **Multi-Document QA:** Challenges models to synthesize information from multiple documents to formulate a coherent answer. It employs Wikipedia-based multi-hop QA datasets, including HotpotQA (Yang et al., 2018), 2WikiMultihopQA (Ho et al., 2020), and MuSiQue (Trivedi et al., 2022).

- **Summarization:** Tests a model's capacity for comprehensive understanding and condensation of long texts. The datasets for this task are GovReport (Huang et al., 2021), QMSum (Zhong et al., 2021), and the multi-document corpus MultiNews (Fabbri et al., 2019).

- **Few-shot Learning:** Measures a model's adaptability on a variety of tasks with limited examples. This includes classification

Table 4: Detailed LongBench Results for Dream-v0-Instruct-7B. Best result in each column within a budget section is in **bold**.

Method	Single-Doc. QA		Multi-Doc. QA			Summarization			Few-shot Learning			Synthetic	Code		Ave. Score
	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PRe	Lcc	RB-P	
Dream-v0-Instruct-7B															
Full KV Cache															
dLLM w/o Cache	28.17	36.23	27.65	32.43	11.83	5.04	14.29	5.95	73.00	89.25	37.84	16.92	38.91	45.08	33.04
dLLM w/ Cache	26.55	39.86	27.66	32.09	11.12	4.40	13.89	5.51	73.50	89.59	36.07	12.05	39.88	45.57	32.70
B=16															
SnapKV	**15.36**	18.35	11.98	9.92	5.27	2.56	10.88	1.91	26.50	27.14	19.42	3.56	17.28	16.08	13.30
PyramidKV	10.81	12.81	12.82	6.40	3.95	**2.57**	10.55	1.89	26.50	8.05	17.98	0.00	15.54	15.19	10.36
SqueezeAttention	12.39	16.36	11.21	6.25	3.95	2.54	10.60	1.87	26.50	10.26	18.35	3.06	13.66	13.60	10.76
AdaKV	15.31	15.14	12.18	8.50	6.44	2.54	10.79	1.82	26.50	21.56	20.08	5.11	16.91	17.10	12.86
MaskKV (Ours)	12.63	**25.72**	**23.87**	**20.99**	**8.53**	2.56	**11.15**	**2.02**	**27.50**	**79.07**	**24.30**	**13.50**	**20.57**	**21.40**	**20.99**
B=32															
SnapKV	17.75	26.82	22.39	**27.91**	7.53	**2.65**	**12.58**	1.95	28.25	66.14	23.67	17.50	22.75	23.87	21.55
PyramidKV	14.55	24.51	22.41	15.27	6.90	2.62	11.89	2.15	28.00	57.55	24.17	11.50	22.02	22.22	18.98
SqueezeAttention	17.62	30.44	20.07	26.15	7.42	2.62	11.83	2.19	26.25	72.17	24.52	17.00	20.91	23.41	22.00
AdaKV	17.09	25.42	23.64	24.77	7.55	2.58	12.46	1.89	27.75	69.77	25.21	18.83	21.67	24.98	21.69
MaskKV (Ours)	**18.53**	**33.76**	**32.92**	24.53	**11.02**	2.59	11.98	**2.25**	**33.25**	**86.47**	**27.55**	**20.00**	**24.89**	**27.40**	**25.51**
B=64															
SnapKV	19.61	34.91	31.26	28.17	10.51	2.65	12.81	2.28	34.25	81.68	27.56	**24.00**	28.07	30.02	26.27
PyramidKV	16.33	29.06	24.28	18.74	8.61	**2.70**	12.73	2.60	33.67	72.43	26.87	23.00	25.47	26.78	23.09
SqueezeAttention	17.62	30.44	23.42	26.15	8.75	2.62	12.58	2.19	26.25	72.17	24.52	17.00	20.91	23.41	22.00
AdaKV	20.98	36.31	33.10	29.06	9.55	2.59	**13.12**	2.29	35.25	85.24	28.50	**24.00**	28.09	**32.76**	27.20
MaskKV (Ours)	**22.73**	**37.98**	**36.47**	**31.03**	**12.26**	2.68	12.39	**2.77**	**45.00**	**87.36**	**29.98**	22.50	**29.11**	31.61	**28.85**
B=128															
SnapKV	**22.36**	39.02	30.28	32.27	11.51	2.72	**13.20**	2.72	38.00	87.17	28.69	**26.05**	**32.55**	36.51	28.79
PyramidKV	17.31	36.59	25.21	24.38	10.99	2.77	12.75	2.90	39.75	84.83	29.71	24.00	30.26	30.33	26.56
SqueezeAttention	19.36	36.19	28.35	29.79	10.25	2.81	13.11	2.64	33.00	85.99	29.25	25.00	27.11	31.92	26.77
AdaKV	21.97	**39.68**	33.24	31.57	11.38	2.69	13.15	2.80	38.75	88.51	30.43	24.25	31.46	**37.28**	29.08
MaskKV (Ours)	21.49	39.23	**38.25**	**33.99**	**14.76**	**2.86**	12.48	**3.19**	**53.50**	**88.69**	**30.97**	21.92	30.71	34.79	**30.49**
B=256															
SnapKV	23.01	40.81	35.52	33.92	12.08	2.89	13.06	3.33	44.75	89.57	32.16	21.65	**35.49**	37.61	30.42
PyramidKV	21.50	36.74	28.35	29.08	9.17	2.79	13.06	3.55	46.17	88.44	31.74	22.25	32.78	33.09	28.55
SqueezeAttention	21.82	41.20	35.80	33.23	11.36	2.83	13.11	2.56	37.00	**89.63**	30.79	**25.00**	30.29	34.69	29.24
AdaKV	24.00	**43.32**	35.86	33.75	11.97	2.90	**13.14**	3.41	48.50	87.70	32.66	19.96	35.40	**39.41**	30.86
MaskKV (Ours)	**24.13**	39.43	**36.47**	**31.93**	**12.66**	**2.93**	12.90	**3.74**	**58.00**	88.52	**32.93**	20.58	32.22	35.94	**30.88**
B=512															
SnapKV	24.27	40.23	35.32	34.87	11.65	2.91	13.32	4.15	51.00	**89.66**	33.09	16.29	37.22	40.47	31.03
PyramidKV	21.50	36.74	28.35	29.08	9.17	2.79	13.06	3.55	46.17	88.44	31.74	**22.25**	32.78	33.09	28.55
SqueezeAttention	22.17	40.15	35.98	**35.63**	11.32	2.89	13.19	3.91	45.00	86.95	32.37	17.08	33.04	38.75	29.89
AdaKV	23.86	**40.62**	**37.45**	34.66	12.58	3.05	**13.61**	3.81	57.75	89.15	33.50	17.42	**37.23**	**42.35**	**31.93**
MaskKV (Ours)	**25.90**	39.71	33.27	34.01	**13.68**	**3.10**	12.92	**4.26**	**60.75**	87.49	**33.78**	16.81	34.19	38.83	31.34

Table 5: Comparison of **dLLM-Cache** with its simplified variants.

Method	Memory (GB)
dLLM-Cache	68.13
+ Mask-Only Projection	53.74
+ Prompt State Exclusion	38.12
+ MaskKV(online)	38.12
+ MaskKV(offline)	**23.42**

with TREC (Li and Roth, 2002), conversational summarization with SAMSum (Gliwa et al., 2019), and reading comprehension with TriviaQA (Joshi et al., 2017).

- **Synthetic Tasks:** Purpose-built challenges designed to test specific abilities, such as counting unique passages (PassageCount (Bai et al., 2023)) or matching a summary to its source passage (PassageRetrieval-en (Raffel et al., 2020)).

- **Code Completion:** Evaluates a model's proficiency in generating code based on existing context. This is tested using the LCC (Guo et al., 2023) dataset for single-file contexts and

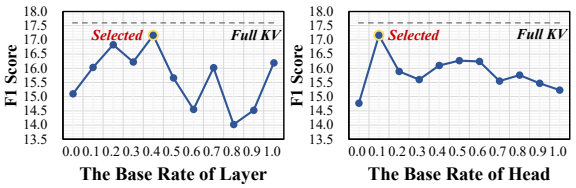

Figure 8: Impact of base rate on model performance.

the RepoBench-P (Liu et al., 2023) dataset for tasks requiring information aggregation across multiple files.

A.3 Case Study: Noise Reduction via Eviction

To investigate why *MaskKV* occasionally outperforms the full-context dLLM-Cache baseline, we present a qualitative analysis on the Qasper dataset in Tab. 9. Long-context tasks often involve documents containing multiple parallel topics or entities (e.g., multiple experiments described in sequential sections). For the Full KV baseline, retaining the entire context introduces a "noise pollution" problem: the attention mechanism may inadvertently attend to **distractor tokens**—semantically similar but task-irrelevant information—causing the model to generate verbose or hallucinated answers that conflate distinct details.

As shown in the case study, the document details three different datasets (IMDb, PTB, IWSLT) for three different tasks. When asked specifically about "sentiment classification", the Full KV model, overwhelmed by the presence of multiple "dataset" entities, retrieves all of them, resulting in a low F1 score due to poor precision. In contrast, *MaskKV* acts as an explicit **information filter**. The *MaskVoting* mechanism identifies that the prompt "sentiment classification" has strong attentional preference only for the IMDb section, while the PTB and IWSLT sections receive low importance scores. Consequently, these irrelevant contexts are evicted from the cache. This "forced focus" eliminates the possibility of distraction, guiding the model to generate a concise, ground-truth-aligned answer. This phenomenon explains our empirical observation: **cache eviction not only saves memory but can also enhance generation quality by denoising the context.**

A.4 Effect of Base Rates α and β

The base rate of head (α) and layer (β) represent uniform budget floors for attention heads and network layers, respectively. These base rates first guarantee each unit a minimal share, after which the remaining budget is redistributed according to estimated importance. Excessively large values drive allocations toward near-uniformity, diluting capacity for critical modules, whereas overly small values make the policy too aggressive and unstable. Empirical results (Fig. 8) show that setting $\alpha = 0.1$ and $\beta = 0.4$ provides the most stable accuracy and key-value (KV) budgets, while still concentrating resources where they matter most.

A.5 Comparison with Additional AR Baselines

To further validate the necessity of dLLM-specific eviction strategies, we compare *MaskKV* with two additional baselines designed for autoregressive models:

- **H2O** (Zhang et al., 2023): An eviction method that retains heavy-hitters based on accumulated attention scores.
- **KVZip**: A compression method that quantizes and merges KV pairs.

As shown in Tab. 10, both H2O and KVZip suffer from severe performance collapse, particularly on retrieval tasks like PRe (PassageRetrieval-en). **H2O** relies on the assumption of attention locality and heavy-hitters accumulation, which is often disrupted in the bidirectional attention of dLLMs. **KVZip**, while effective for ARMs, modifies the KV states in a way that is inconsistent with the iterative denoising procedure of diffusion models. In contrast, *MaskKV* leverages the mask tokens as natural probes to identify global importance, making it a more suitable and effective strategy for dLLMs.

A.6 Comparison of Voting Strategies

We analyze voting strategies by grouping masked tokens into front, middle, and back regions, each contributing a position-specific vote. Results (Tab. 11) show that later positions yield more accurate votes for identifying critical tokens, suggesting position-aware voting may further improve eviction effectiveness.

A.7 Visualization of Prompt Preference.

As shown in Fig. 9, we analyze the prompt preference distribution across different heads within the same layer. Some heads allocate substantial attention to the prompt, likely to extract task-relevant information, while others focus more on the masked

Table 6: Detailed information of the datasets in the LongBench benchmark.

Label	Task	Eval Metric	Avg Len	Gen Len	Language	Sample Num
NrtvQA	NarrativeQA	F1	18,409	128	EN	200
Qasper	Qasper	F1	3,619	128	EN	200
MF-en	MultiFieldQA-en	F1	4,559	64	EN	150
HotpotQA	HotpotQA	F1	9,151	32	EN	200
2WikiMQA	2WikiMultihopQA	F1	4,887	32	EN	200
Musique	MuSiQue	F1	11,214	32	EN	200
GovReport	GovReport	Rouge-L	8,734	512	EN	200
QMSum	QMSum	Rouge-L	10,614	512	EN	200
MultiNews	MultiNews	Rouge-L	2,113	512	EN	200
TREC	TREC	Accuracy	5,177	64	EN	200
TriviaQA	TriviaQA	F1	8,209	32	EN	200
SAMSum	SAMSum	Rouge-L	6,258	128	EN	200
PCount	PassageCount	Accuracy	11,141	32	EN	200
PRe	PassageRetrieval-en	Accuracy	9,289	32	EN	200
Lcc	LCC	Edit Sim	1,235	64	Python/C#/Java	500
RB-P	RepoBench-P	Edit Sim	4,206	64	Python/Java	500

Table 7: Task performance (%) on PassageRetrieval-en under different calibration set sizes.

Calibration Set Size	Performance (%)
10	98.75
20	99.42
40	99.17
80	99.42
160	98.50

Table 8: Task performance (%) on PassageRetrieval-en under calibration sets constructed from different task categories.

Calibration Task Type	Performance (%)
QA	99.00
Code	98.42
Summarization	98.33
Few-shot Learning	98.00

region to plan answer generation. Such analysis provides deeper insight into their internal decision behavior.

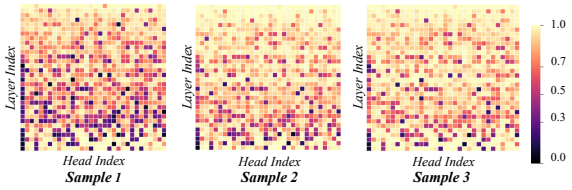

Figure 9: Visualization of Prompt Preference.

A.8 A Formal Proof on the Primacy of Mask Attention

A.8.1 Preliminaries and Notation

To ensure the rigor of the proof, we first define the symbols and notation used throughout this section.

- **Input Sequence:** The input sequence $X \in \mathbb{R}^{n \times d}$ consists of embeddings for n tokens, where d is the embedding dimension. The sequence X is partitioned into two parts:
  - **Prompt:** $X_p \in \mathbb{R}^{n_p \times d}$, with its set of token indices denoted as S_p.
  - **Mask:** $X_m \in \mathbb{R}^{n_m \times d}$, with its set of token indices denoted as S_m.

  The full sequence is a concatenation $X = [X_p, X_m]$, with a total length of $n = n_p + n_m$.
- **Transformer Layer:** A standard Transformer model consists of L identical layers stacked on top of each other. Let $h^{(l)} \in \mathbb{R}^{n \times d}$ denote the output hidden state of the l-th layer, where $l \in \{1, \ldots, L\}$. We define the initial embedding as the output of the 0-th layer, i.e., $h^{(0)} = X$.

Table 9: A case study from the Qasper dataset (LongBench) illustrating how cache eviction mitigates noise pollution. The document describes three distinct tasks (Sentiment Analysis, Language Modeling, Machine Translation) and their respective datasets. While the Full KV model is distracted by irrelevant datasets appearing in similar contexts, MaskKV effectively filters out these "distractor" spans, yielding a precise answer.

Role	Content
Question	What sentiment classification dataset is used?
Context	*(Excerpt from Introduction & Experiments)* ... We describe QRNN variants tailored to several natural language tasks, including **document-level sentiment classification**, **language modeling**, and **character-level machine translation**... ... *[Section: Sentiment Classification]* We evaluate the QRNN architecture on a popular document-level sentiment classification benchmark, the **IMDb movie review dataset** ... ... *[Section: Language Modeling]* We replicate the language modeling experiment ... using a standard preprocessed version of the **Penn Treebank (PTB)** ... ... *[Section: Machine Translation]* We evaluate ... on a challenging neural machine translation task, **IWSLT German–English** ...
Vanilla (Full KV)	The paper uses the IMDb movie review dataset for sentiment classification, the Penn Treebank (PTB) for language modeling, and the IWSLT German–English dataset for translation. ***Analysis:*** *The model fails to strictly follow the constraint "sentiment classification" and retrieves all datasets mentioned in the experiment section, leading to low precision.*
MaskKV (Ours)	The IMDb movie review dataset. ***Analysis:*** *By evicting the KV pairs associated with the "Language Modeling" and "Translation" sections (which received low attention from the mask queries), MaskKV eliminates the distractors, resulting in a concise and accurate match.*

- **Intra-Layer Computation:** The computation within layer l can be represented as a function, Block, which takes the output of the previous layer $h^{(l-1)}$ as input:

$$h^{(l)} = \text{Block}(h^{(l-1)})$$

To analyze the information flow, we can abstract the update process of each layer. The output of layer l is the sum of its input and an update term $\Delta^{(l)}$:

$$h^{(l)} = h^{(l-1)} + \Delta^{(l)}$$

where $\Delta^{(l)}$ represents the total update contributed by the sub-layers (MHA and FFN) of layer l.

A.8.2 Proposition

For any mask token $m \in S_m$, its final hidden state $h_m^{(L)}$, which directly determines the predictive logits, can be precisely expressed as the sum of its initial embedding $h_m^{(0)}$ and the cumulative updates from all L layers of the model. Within these updates, the Multi-Head Attention (MHA) mechanism serves as the sole channel for the mask token to incorporate information from the prompt tokens. Consequently, the attention scores originating from mask queries are the most direct and fundamental indicators of the importance of prompt information for the model's generative process.

A.8.3 Proof

The proof proceeds in three steps. First, we establish the central role of $h_m^{(L)}$ by considering the model's objective function. Second, we derive the compositional structure of $h_m^{(L)}$ through a recursive expansion. Finally, we analyze the components of this structure to demonstrate the unique role of the attention mechanism.

Step 1: The Inference Objective and Decisive Computations During inference, the objective of the model is to predict a sequence of tokens for the positions specified by the mask index set, S_m. This generative process begins with the computation of the final hidden states, $h^{(L)} \in \mathbb{R}^{n \times d}$, for the entire sequence. The language model head (LM Head), a linear projection matrix $W_{out} \in \mathbb{R}^{d \times |V|}$ (where $|V|$ is the vocabulary size), then maps these hidden states to logit vectors:

$$\text{Logits} = h^{(L)} \cdot W_{out}$$

A "softmax" function is subsequently applied to the logits at each position to yield a probability distribution over the vocabulary.

Critically, for the task at hand, our interest lies exclusively in the logits at the active mask positions (S_m), since all other tokens—whether part of the original prompt (S_p) or already unmasked in prior steps—are considered fixed context. Therefore, the generative process, whether it be greedy decoding or sampling, is performed exclusively on the probability distributions corresponding to the mask positions. This implies that the quantities of interest, which solely determine the generated output, are the final hidden states of the mask tokens, $\{h_m^{(L)} \mid m \in S_m\}$.

Table 10: Performance comparison with H2O and KVZip on LLaDA-8B. These methods exhibit significant performance degradation compared to MaskKV, highlighting the gap between AR and dLLM cache mechanisms.

Method	Budget	Qasper	2Wiki	SAMSum	PRe	LCC
KVZip	32	4.79	2.03	3.65	1.00	14.90
	128	12.19	10.43	3.64	4.48	20.18
H2O	32	2.37	5.25	4.59	1.08	37.51
	128	9.11	13.58	6.72	36.50	45.22
MaskKV (Ours)	32	**14.61**	**15.68**	**29.28**	**90.33**	**56.08**
	128	**20.21**	**16.65**	**37.28**	**98.17**	**61.61**

Table 11: Effect of mask token position on voting performance (**budget=256**). The best score in each column and the best overall average are highlighted in **bold**.

Mask Position	HotpotQA	2WikiMQA	Musique	TriviaQA	PRe	Average
first (front)	12.39	15.72	9.19	41.69	**99.75**	35.75
second (middle)	12.89	15.85	8.49	37.99	96.58	34.36
third (middle)	12.82	**16.82**	8.72	39.74	96.00	34.82
last (back)	**14.33**	15.97	**9.69**	**44.75**	97.50	**36.45**

Variant	Performance (Avg)
Mask-Voting	35.39
Layer Budget Allocation	
+SqueezeAttention	$35.62_{+0.23}$
+PyramidKV	$33.52_{-1.87}$
+Layer-Importance(online)	$\mathbf{35.65}_{+0.26}$
+Layer-Importance(offline)	$\mathbf{35.74}_{+0.35}$
Head Budget Allocation	
+AdaKV	$35.70_{+0.32}$
+Prompt-Preference	$\mathbf{35.96}_{+0.57}$
Layer + Head Allocation	
MaskKV (Ours)	$\mathbf{36.27}_{+0.88}$

Table 12: Ablation study of budget allocation.

Table 13: Comparison of token selection strategies on **gsm8k** with budget= 128. *Prompt-Voting* computes importance by using the full context to attend to itself (context-to-context), whereas *All-Voting* aggregates attention scores from all available queries.

Token Selection Strategy	Score (%)
SnapKV	64.90
Prompt-Voting	60.35
Mask-Voting	**68.08**
All-Voting	66.64

Step 2: Recursive Expansion of the Hidden State Based on the abstract update rule $h^{(l)} = h^{(l-1)} + \Delta^{(l)}$, we can perform a recursive expansion (a telescoping sum) for the final hidden state $h_m^{(L)}$ of any mask token m:

$$\begin{aligned} h_m^{(L)} &= h_m^{(L-1)} + \Delta_m^{(L)} \\ &= (h_m^{(L-2)} + \Delta_m^{(L-1)}) + \Delta_m^{(L)} \\ &= h_m^{(L-2)} + \Delta_m^{(L-1)} + \Delta_m^{(L)} \\ &\vdots \\ &= h_m^{(0)} + \sum_{l=1}^{L} \Delta_m^{(l)} \end{aligned}$$

This expansion is the mathematical centerpiece of our proof, showing that the final representation is an accumulation of updates upon its initial state.

Step 3: Analysis of the Update Components We now analyze the composition of the cumulative update term $\sum_{l=1}^{L} \Delta_m^{(l)}$. Each layer's update $\Delta_m^{(l)}$ consists of contributions from the MHA and FFN sub-layers: $\Delta_m^{(l)} = \text{MHA}_m^{(l)} + \text{FFN}_m^{(l)}$.

- **Contribution of the Feed-Forward Network (FFN):** The FFN is a position-wise transformation. Its computation for token m is independent of all other tokens $j \neq m$. Thus, the FFN can only process and non-linearly transform the information already present in h_m; it cannot introduce new information from the prompt.
- **Contribution of Multi-Head Attention (MHA):** The MHA mechanism is fundamentally different. The output for token m,

AttnOut_m, is a weighted sum of the Value vectors V_j of all tokens in the sequence:

$$\text{AttnOut}_m = \sum_{j=1}^{n} \alpha_{mj} V_j$$

where the attention weight $\alpha_{mj} = \text{softmax}\left(\frac{Q_m K_j^T}{\sqrt{d_k}}\right)$. The summation index j spans all tokens, including those in the prompt ($j \in S_p$). This demonstrates that MHA is the **exclusive** mechanism that allows for information exchange between different token positions. For a mask token $m \in S_m$, only through MHA can it interact with prompt tokens $j \in S_p$ to aggregate relevant information.

Combining these points, we see that within the final representation $h_m^{(L)} = h_m^{(0)} + \sum_{l=1}^{L}(\text{MHA}_m^{(l)} + \text{FFN}_m^{(l)})$, the MHA term is the sole channel through which information from the prompt can be incorporated into the mask token's representation.

A.8.4 Conclusion and Implication

In conclusion, our theoretical proof establishes the primacy of attention guided by mask queries within the generative paradigm of dLLMs. By demonstrating that this mechanism is the indispensable information bridge from context to target, our findings provide a robust theoretical foundation for novel inference strategies, such as the KV cache selection method proposed in this work.

A.9 Algorithm

Algorithm 1 MaskKV Main Inference Algorithm

Input: Prompt $\mathbf{c}$, initial masked sequence $\mathbf{y}^{(K)}$, steps K, intervals T_p, T_r, adaptive update ratio ρ
Input: Budget B, Base Ratios α, β
Output: Final prediction $\hat{\mathbf{y}}^{(0)}$

/* Stage 1: Offline Profiling & Layer Budget Allocation */
$S_{\text{offline}} \leftarrow$ LoadPrecomputedLayerImportance()
$\mathbf{B}_{layers} \leftarrow$ List()
$\mathbf{B}_{layers} \leftarrow$ AllocLayerBudget$(B, S_{\text{offline}}, \beta)$ ▷ Assign B_l based on importance
/* Stage 2: Initialization & Mask-Voting (Step $k = K$) */
$\mathcal{C}_p, \mathcal{C}_r, \mathcal{K}_{idx} \leftarrow$ InitializeAndProfile$(\mathbf{c}, \mathbf{y}^{(K)}, \mathbf{B}_{layers}, \alpha)$ ▷ See Algorithm 3
Generate prediction $\hat{\mathbf{y}}^{(0)}$; $\mathbf{y}^{(K-1)} \leftarrow \pi(\hat{\mathbf{y}}^{(0)}, \mathbf{y}^{(K)}, \mathbf{c}, K)$
/* Stage 3: Iterative Denoising Loop */
for $k = K - 1$ **down to** 1 **do**
    $\mathbf{x}_{in} \leftarrow [\mathbf{c}; \mathbf{y}^{(k)}]$
    for each layer l in the Transformer network **do**
        ref_p $\leftarrow (k \pmod{T_p} \equiv 0)$
        ref_r $\leftarrow (k \pmod{T_r} \equiv 0)$
        if ref_p **and** ref_r **then**
            $\mathbf{x}_{out}, \mathcal{C}_p, \mathcal{C}_r \leftarrow$ FullRefresh$(\mathbf{x}_{in}, l, \mathcal{C}_p, \mathcal{C}_r, \mathcal{K}_{idx}[l])$
        else if ref_p **and not** ref_r **then**
            $\mathbf{x}_{out}, \mathcal{C}_p, \mathcal{C}_r \leftarrow$ RefreshPromptOnly$(\mathbf{x}_{in}, l, \mathcal{C}_p, \mathcal{C}_r, \mathcal{K}_{idx}[l])$
        else if not ref_p **and** ref_r **then**
            $\mathbf{x}_{out}, \mathcal{C}_p, \mathcal{C}_r \leftarrow$ RefreshResponseOnly$(\mathbf{x}_{in}, l, \mathcal{C}_p, \mathcal{C}_r)$
        else
            $\mathbf{x}_{out}, \mathcal{C}_p, \mathcal{C}_r \leftarrow$ AdaptiveUpdate$(\mathbf{x}_{in}, l, \mathcal{C}_p, \mathcal{C}_r, \rho)$
        end if
        $\mathbf{x}_{in} \leftarrow \mathbf{x}_{out}$
    end for
    Generate prediction $\hat{\mathbf{y}}^{(0)}$; $\mathbf{y}^{(k-1)} \leftarrow \pi(\hat{\mathbf{y}}^{(0)}, \mathbf{y}^{(k)}, \mathbf{c}, k)$
end for
return final prediction $\hat{\mathbf{y}}^{(0)}$

Algorithm 2 AllocLayerBudget (Offline Bimodal Strategy)

Input: Target Average Budget B, Layer Importance Scores $S \in \mathbb{R}^L$, Base Ratio β
Output: Per-layer budget list $\mathbf{B}_{layers}$

```
/* 1. Base Allocation (Per Layer) */
```
$k_{\text{base}} \leftarrow \lfloor \beta \cdot B \rfloor$ ▷ Guaranteed budget for every layer
```
/* 2. Calculate Total Variable Pool */
```
$B_{\text{pool}} \leftarrow (B - k_{\text{base}}) \cdot L$
```
/* 3. Group Aggregation (Bimodal Profile) */
```
Define groups: G_{bound} (Boundary layers) and G_{mid} (Middle layers)
$S_{\text{bound}} \leftarrow \sum_{l \in G_{\text{bound}}} S[l]; \quad S_{\text{mid}} \leftarrow \sum_{l \in G_{\text{mid}}} S[l]$
```
/* 4. Distribute Pool to Groups */
```
$Pool_{\text{bound}} \leftarrow \lfloor B_{\text{pool}} \cdot \frac{S_{\text{bound}}}{S_{\text{bound}} + S_{\text{mid}}} \rfloor$
$Pool_{\text{mid}} \leftarrow B_{\text{pool}} - Pool_{\text{bound}}$
```
/* 5. Assign Final Budget to Each Layer */
```
for $l \leftarrow 1$ to L **do**
    if $l \in G_{\text{bound}}$ **then**
        $k_{\text{var}} \leftarrow \lfloor Pool_{\text{bound}} / |G_{\text{bound}}| \rfloor$
    else
        $k_{\text{var}} \leftarrow \lfloor Pool_{\text{mid}} / |G_{\text{mid}}| \rfloor$
    end if
    $\mathbf{B}_{layers}[l] \leftarrow k_{\text{base}} + k_{\text{var}}$
end for
return $\mathbf{B}_{layers}$

Algorithm 3 InitializeAndProfile (MaskKV Profiling)

Input: Prompt $\mathbf{c}$, sequence $\mathbf{y}^{(K)}$, layer budgets $\mathbf{B}_{layers}$, head ratio α
Output: Initial caches $\mathcal{C}_p, \mathcal{C}_r$, eviction indices $\mathcal{K}_{idx}$

Initialize empty $\mathcal{C}_p, \mathcal{C}_r, \mathcal{K}_{idx}$; $\mathbf{x}_{in} \leftarrow [\mathbf{c}; \mathbf{y}^{(K)}]$
for each layer $l \in \{1, \ldots, L\}$ **do**
  $\mathbf{x}_{norm} \leftarrow \text{LayerNorm}(\mathbf{x}_{in})$
  $\mathbf{Q}, \mathbf{K}, \mathbf{V} \leftarrow \text{Proj}(\mathbf{x}_{norm})$
  $\mathbf{Q}_m \leftarrow \mathbf{Q}_{|\mathbf{c}|:}$
  $\mathbf{K}_p \leftarrow \mathbf{K}_{:|\mathbf{c}|}$; $\mathbf{V}_p \leftarrow \mathbf{V}_{:|\mathbf{c}|}$
  /* 1. Mask-Voting: Identify Important Prompt Tokens */
  $\mathbf{A} \leftarrow \text{Softmax}(\frac{\mathbf{Q}_m \mathbf{K}_p^T}{\sqrt{D}}, \text{dim} = -1)$ ▷ Sum over mask queries
  $\mathbf{I} \leftarrow \mathbf{A}.\text{sum}(\text{dim} = -2)$
  $\mathbf{I} \leftarrow \text{GroupMean}(\mathbf{I})$ ▷ Average scores across GQA groups to align with KV heads
  /* 2. Hierarchical Budget Allocation */
  $B_l \leftarrow \mathbf{B}_{layers}[l]$ ▷ Average budget per head
  $\mathbf{P} \leftarrow \frac{\mathbf{I}.\text{sum}(\text{dim}=-1)}{\mathbf{I}.\text{sum}()}$ ▷ Normalize across heads to get prompt preference
  $k_{base} \leftarrow \lfloor \alpha \cdot B_l \rfloor$
  $B_{pool} \leftarrow (B_l - k_{base}) \cdot H_{kv}$
  $\mathbf{k} \leftarrow k_{base} + \lfloor B_{pool} \cdot \mathbf{P} \rfloor$ ▷ Distribute pool based on preference $\mathbf{P}$
  /* 3. Generate Eviction Indices */
  for $h \in \{1, \ldots, H\}$ **do**
    $\mathcal{K}_{idx}[l][h] \leftarrow \text{TopK}(\mathbf{I}[h], \mathbf{k}[h]).indices.sort()$
  end for
  /* 4. Cache Compression */
  $\mathbf{K}_p \leftarrow \text{Gather}(\mathbf{K}_p, \mathcal{K}_{idx}[l])$; $\mathbf{V}_p \leftarrow \text{Gather}(\mathbf{V}_p, \mathcal{K}_{idx}[l])$
  $\mathcal{C}_p[l].kv \leftarrow \{\mathbf{K}_p, \mathbf{V}_p\}$; $\mathcal{C}_r[l].kv \leftarrow \{\mathbf{K}_{|\mathbf{c}|:}, \mathbf{V}_{|\mathbf{c}|:}\}$
  /* Standard Forward Pass (Using Full KV) */
  $\mathbf{Attn} \leftarrow \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ ▷ Use full context for initial step
  $\mathbf{x}_{in} \leftarrow \mathbf{x}_{in} + \mathbf{Attn} + \text{FFN}(\ldots)$
end for
return $\mathcal{C}_p, \mathcal{C}_r, \mathcal{K}_{idx}$

Algorithm 4 MaskKV Case 1 - Full Refresh

Input: $\mathbf{x}_{in}$ (Full Sequence), $l, \mathcal{C}_p, \mathcal{C}_r, \mathcal{K}_{idx}$
Output: $\mathbf{x}_{out}, \mathcal{C}_p, \mathcal{C}_r$

$\mathbf{x}_{norm} \leftarrow \text{LayerNorm}(\mathbf{x}_{in})$
$\mathbf{Q}, \mathbf{K}, \mathbf{V} \leftarrow \text{Proj}(\mathbf{x}_{norm})$
/* 1. MaskKV Eviction & KV Caching */
$\mathbf{K}_p \leftarrow \text{Gather}(\mathbf{K}_{:|\mathbf{c}|}, \mathcal{K}_{idx})$; $\mathbf{V}_p \leftarrow \text{Gather}(\mathbf{V}_{:|\mathbf{c}|}, \mathcal{K}_{idx})$ ▷ Compress Prompt KV
$\mathcal{C}_p[l].\text{kv} \leftarrow \{\mathbf{K}_p, \mathbf{V}_p\}$
$\mathcal{C}_r[l].\text{kv} \leftarrow \{\mathbf{K}_{|\mathbf{c}|:}, \mathbf{V}_{|\mathbf{c}|:}\}$
/* 2. Compute & Cache Response Attention */
$\mathbf{Attn} \leftarrow \text{Attention}(\mathbf{Q}, [\mathbf{K}_p; \mathbf{K}_{|\mathbf{c}|:}], [\mathbf{V}_p; \mathbf{V}_{|\mathbf{c}|:}])$
$\mathcal{C}_r[l].\text{attn} \leftarrow \mathbf{Attn}_{|\mathbf{c}|:}$ ▷ Cache Response Features
$\mathbf{x}_{mid} \leftarrow \mathbf{x}_{in} + \mathbf{Attn}$
/* 3. Compute & Cache Response MLP */
$\mathbf{MLP} \leftarrow \text{FFN}(\text{LayerNorm}(\mathbf{x}_{mid}))$
$\mathcal{C}_r[l].\text{mlp} \leftarrow \mathbf{MLP}_{|\mathbf{c}|:}$ ▷ Cache Response Features
$\mathbf{x}_{out} \leftarrow \mathbf{x}_{mid} + \mathbf{MLP}$
return $\mathbf{x}_{out}, \mathcal{C}_p, \mathcal{C}_r$

Algorithm 5 MaskKV Case 2 - Refresh Prompt Only

Input: $\mathbf{x}_{in}, l, \mathcal{C}_p, \mathcal{C}_r, \mathcal{K}_{idx}$
Output: $\mathbf{x}_{out}, \mathcal{C}_p, \mathcal{C}_r$

$\mathbf{Q}_p, \mathbf{K}_p, \mathbf{V}_p \leftarrow \text{Proj}(\text{LayerNorm}(\mathbf{x}_{in,:|\mathbf{c}|}))$
/* 1. Update Prompt KV (Eviction) */
$\mathbf{K}_p \leftarrow \text{Gather}(\mathbf{K}_p, \mathcal{K}_{idx})$; $\mathbf{V}_p \leftarrow \text{Gather}(\mathbf{V}_p, \mathcal{K}_{idx})$
$\mathcal{C}_p[l].\text{kv} \leftarrow \{\mathbf{K}_p, \mathbf{V}_p\}$; $\{\mathbf{K}_r, \mathbf{V}_r\} \leftarrow \mathcal{C}_r[l].\text{kv}$
/* 2. Compute Prompt Attn & Reuse Response Attn */
$\mathbf{Attn}_p \leftarrow \text{Attention}(\mathbf{Q}_p, [\mathbf{K}_p; \mathbf{K}_r], [\mathbf{V}_p; \mathbf{V}_r])$
$\mathbf{x}_{mid} \leftarrow \mathbf{x}_{in} + [\mathbf{Attn}_p; \mathcal{C}_r[l].\text{attn}]$
/* 3. Compute Prompt MLP & Reuse Response MLP */
$\mathbf{MLP}_p \leftarrow \text{FFN}(\text{LayerNorm}(\mathbf{x}_{mid,:|\mathbf{c}|}))$
$\mathbf{x}_{out} \leftarrow \mathbf{x}_{mid} + [\mathbf{MLP}_p; \mathcal{C}_r[l].\text{mlp}]$
return $\mathbf{x}_{out}, \mathcal{C}_p, \mathcal{C}_r$

Algorithm 6 MaskKV Case 3 - Refresh Response Only

Input: $\mathbf{x}_{in}, l, \mathcal{C}_p, \mathcal{C}_r$
Output: $\mathbf{x}_{out}, \mathcal{C}_p, \mathcal{C}_r$

$\mathbf{Q}_r, \mathbf{K}_r, \mathbf{V}_r \leftarrow \text{Proj}(\text{LayerNorm}(\mathbf{x}_{in,|\mathbf{c}|:}))$
/* 1. Update Response KV & Reuse Prompt KV */
$\mathcal{C}_r[l].\text{kv} \leftarrow \{\mathbf{K}_r, \mathbf{V}_r\}$; $\{\mathbf{K}_p, \mathbf{V}_p\} \leftarrow \mathcal{C}_p[l].\text{kv}$
/* 2. Compute Response Attn & Store */
$\mathcal{C}_r[l].\text{attn} \leftarrow \text{Attention}(\mathbf{Q}_r, [\mathbf{K}_p; \mathbf{K}_r], [\mathbf{V}_p; \mathbf{V}_r])$
$\mathbf{x}_{mid} \leftarrow \mathbf{x}_{in} + [\mathbf{0}; \mathcal{C}_r[l].\text{attn}]$
/* 3. Compute Response MLP & Store */
$\mathcal{C}_r[l].\text{mlp} \leftarrow \text{FFN}(\text{LayerNorm}(\mathbf{x}_{mid,|\mathbf{c}|:}))$
$\mathbf{x}_{out} \leftarrow \mathbf{x}_{mid} + [\mathbf{0}; \mathcal{C}_r[l].\text{mlp}]$
return $\mathbf{x}_{out}, \mathcal{C}_p, \mathcal{C}_r$

Algorithm 7 MaskKV Case 4 - Adaptive Sparse Update (V-Verify)

Input: $\mathbf{x}_{in}, l, \mathcal{C}_p, \mathcal{C}_r, \rho$ (Update Ratio)
Output: $\mathbf{x}_{out}, \mathcal{C}_p, \mathcal{C}_r$

/* 1. Global V-Check (Compute V for all) */
$\mathbf{V}_r \leftarrow \text{Proj}_V(\text{LayerNorm}(\mathbf{x}_{in,|\mathbf{c}|:}))$
$\text{Sim} \leftarrow \text{CosineSim}(\mathbf{V}_r, \mathcal{C}_r[l].\text{kv}.\mathbf{V})$
/* 2. Select Indices to Refresh */
$k \leftarrow \lfloor \rho \cdot |\mathbf{y}| \rfloor$
$\mathcal{I} \leftarrow \text{TopK}(-\text{Sim}, k).indices$ ▷ Find tokens with largest drift
/* 3. Sparse KV Update (Only at $\mathcal{I}$) */
$\mathbf{x}_{selected} \leftarrow \text{Gather}(\mathbf{x}_{in,|\mathbf{c}|:}, \mathcal{I})$
$\mathbf{Q}_{\mathcal{I}}, \mathbf{K}_{\mathcal{I}} \leftarrow \text{Proj}_{QK}(\text{LayerNorm}(\mathbf{x}_{selected}))$
$\mathcal{C}_r[l].\text{kv}.\mathbf{K}.\text{scatter}(\mathcal{I}, \mathbf{K}_{\mathcal{I}})$; $\mathcal{C}_r[l].\text{kv}.\mathbf{V} \leftarrow \mathbf{V}_r$
/* 4. Sparse Attention (Only at $\mathcal{I}$) */
$\{\mathbf{K}_{full}, \mathbf{V}_{full}\} \leftarrow [\mathcal{C}_p.\text{kv}; \mathcal{C}_r.\text{kv}]$
$\mathbf{Attn}_{\mathcal{I}} \leftarrow \text{Attention}(\mathbf{Q}_{\mathcal{I}}, \mathbf{K}_{full}, \mathbf{V}_{full})$
$\mathcal{C}_r[l].\text{attn}.\text{scatter}(\mathcal{I}, \mathbf{Attn}_{\mathcal{I}})$ ▷ Update Feature Cache
/* 5. Sparse MLP (Only at $\mathcal{I}$) */
$\mathbf{x}_{mid,\mathcal{I}} \leftarrow \mathbf{x}_{selected} + \mathbf{Attn}_{\mathcal{I}}$
$\mathbf{MLP}_{\mathcal{I}} \leftarrow \text{FFN}(\text{LayerNorm}(\mathbf{x}_{mid,\mathcal{I}}))$
$\mathcal{C}_r[l].\text{mlp}.\text{scatter}(\mathcal{I}, \mathbf{MLP}_{\mathcal{I}})$ ▷ Update Feature Cache
/* 6. Output */
$\mathbf{x}_{out} \leftarrow \mathbf{x}_{in} + [\mathbf{0}; \mathcal{C}_r[l].\text{attn}] + [\mathbf{0}; \mathcal{C}_r[l].\text{mlp}]$
return $\mathbf{x}_{out}, \mathcal{C}_p, \mathcal{C}_r$