ATTENTION IS ALL YOU NEED FOR KV CACHE IN DIFFUSION LLMS

Quan Nguyen-Tri*
FPT AI Residency
Hanoi, Vietnam
`quannt40@fpt.com`

Mukul Ranjan* **& Zhiqiang Shen**
VILA Lab, MBZUAI
Abu Dhabi, UAE
{`mukul.ranjan,zhiqiang.shen`}`@mbzuai.ac.ae`

Project page: `https://vila-lab.github.io/elastic-cache-webpage/`

arXiv:2510.14973v1 [cs.CL] 16 Oct 2025

ABSTRACT

This work studies how to adaptively recompute key–value (KV) caches for diffusion large language models (DLMs) to maximize prediction accuracy while minimizing decoding latency. Prior methods' decoders recompute QKV for all tokens at every denoising step and layer, despite KV states changing little across most steps, especially in shallow layers, leading to substantial redundancy. We make three observations: (1) distant **MASK** tokens primarily act as a length-bias and can be cached block-wise beyond the active prediction window; (2) KV dynamics increase with depth, suggesting that selective refresh starting from deeper layers is sufficient; and (3) the most-attended token exhibits the smallest KV drift, providing a conservative lower bound on cache change for other tokens. Building on these, we propose **Elastic-Cache**, a training-free, architecture-agnostic strategy that jointly decides *when* to refresh (via an attention-aware drift test on the most-attended token) and *where* to refresh (via a depth-aware schedule that recomputes from a chosen layer onward while reusing shallow-layer caches and off-window MASK caches). Unlike fixed-period schemes, Elastic-Cache performs adaptive, layer-aware cache updates for diffusion LLMs, reducing redundant computation and accelerating decoding with negligible loss in generation quality. Experiments on LLaDA-Instruct, LLaDA-1.5, and LLaDA-V across mathematical reasoning and code generation tasks demonstrate consistent speedups: 8.7× on GSM8K (256 tokens), $45.1\times$ on longer sequences, and $4.8\times$ on HumanEval, while consistently maintaining higher accuracy than the baseline. Our method achieves significantly higher throughput (6.8× on GSM8K) than existing confidence-based approaches while preserving generation quality, enabling practical deployment of diffusion LLMs.

1 INTRODUCTION

Diffusion large language models (DLMs) (Li et al., 2025) have recently emerged as a compelling alternative to autoregressive Transformers (Radford et al., 2018; Achiam et al., 2023), yet their iterative denoising procedure makes inference particularly compute-intensive. In standard implementations, each decoding step recomputes queries, keys, and values (QKV) for every token at every layer, even though the underlying key–value (KV) states change only marginally across most steps. This all-tokens, all-layers recomputation incurs substantial latency and memory traffic, ultimately limiting practical deployment. Our goal in this study is to determine *how and when* to adaptively recompute the KV cache during decoding so as to maximize prediction quality while minimizing wall-clock latency.

A defining property of diffusion LLM decoding is the progressive unmasking of tokens under a length- and structure-aware attention pattern. This induces heterogeneous KV dynamics: shallow layers tend to stabilize quickly as they encode local lexical structure, whereas deeper layers continue to adjust global, semantic dependencies. We formalize this with a notion of KV drift: the step-to-step change in cached keys and values, and observe two consistent trends: (i) drift is small for most steps, and (ii) drift grows with layer depth. These trends suggest that indiscriminate recomputation is wasteful, and that targeted refreshes could preserve accuracy while slashing cost.

Prior acceleration methods for diffusion (and related) decoders typically refresh the KV cache on a fixed schedule, e.g., every k iterations without regard to instance difficulty, current attention patterns, or layerwise variability. Such fixed-period policies leave performance on the table: they recompute when nothing has changed and miss updates precisely when rapid semantic revisions occur. Moreover, by treating all layers uniformly, they over-service shallow layers whose representations have already converged, while under-servicing deeper layers where changes matter most. This motivates an adaptive, attention-aware alternative.

---

*Equal contribution

Our approach is built on three empirical observations. First, distant MASK tokens exert negligible influence on unmasking the current token and behave primarily as a length-bias prior; thus, their KV can be block-cached outside the active prediction window to avoid redundant work. Second, KV drift increases with depth, so refreshes should start at a learned boundary layer $\ell^\star$ and apply only to deeper layers, reusing shallow-layer caches. Third, the most-attended token at a step typically exhibits the smallest drift, providing a conservative lower bound on KV changes across the context. Monitoring this drift yields a reliable, low-overhead trigger for deciding whether a global refresh is warranted.

Based on these ideas, we propose **Elastic-Cache**, a training-free, architecture-agnostic strategy that couples *Attention-Aware KV Cache Update* with *Layer-Aware KV Cache Update*. The attention-aware module computes a lightweight drift statistic on the most-attended token; if the statistic exceeds a threshold, a refresh is triggered, otherwise cached KVs are reused. The layer-aware module then refreshes only layers $\ell \geq \ell^\star$, while shallow layers retain their caches, and off-window MASK tokens remain block-cached. Together, these mechanisms align recomputation with *where and when* the model's beliefs actually change, minimizing unnecessary QKV work.

In contrast to fixed-period baselines, our **Elastic-Cache** adapts to the input, step, and layer granularity together. It reduces compute by skipping recomputation during stable phases, focuses effort on deeper layers during semantic revisions, and leverages block-wise caching for distant MASK tokens. Conceptually, the method reframes KV management as an **attention-guided control problem**: attention estimates *which* tokens matter; drift detects *how much* the state has changed; and the layer boundary $\ell^\star$ encodes *where* updates pay off. This yields a practical pathway to low-latency diffusion LLM decoding without modifying training or the base architecture.

Our contributions of this work:

- We diagnose redundancy in diffusion LLM decoding and introduce KV drift as a principled signal for adaptive cache management.
- We propose **Elastic-Cache**, the first (to our best knowledge) adaptive, layer-aware KV refresh policy for diffusion LLMs that jointly decides *when* to recompute (attention-aware drift test) and *where* to recompute (depth-selective updates).
- We develop *block-wise MASK caching* to eliminate needless updates outside the prediction window. We provide comprehensive empirical experiments and ablations showing that our **Elastic-Cache** preserves generation quality while substantially reducing decoding latency across tasks and model scales.

2 PRELIMINARY

2.1 MASKED DIFFUSION MODELS

Masked Diffusion Models (MDMs), absorbing-state discrete diffusion, build on D3PM (Austin et al., 2021a) and its continuous-time variant (Campbell et al., 2022), replacing tokens with a special MASK along a forward process (Sahoo et al., 2024; Shi et al., 2024) at timestep t:

$$q_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0) = \prod_{i=1}^{L} q_{t|0}(x_t^i|x_0^i) = \prod_{i=1}^{L} \mathrm{Cat}(x_t^i; (1-t)\delta_{x_0^i} + t\delta_{\mathrm{MASK}}) \tag{1}$$

where $t \in [0,1]$ controls interpolation between the original data $\boldsymbol{x}_0$ (at $t=0$) and a fully masked sequence (at $t=1$), $\mathrm{Cat}(\cdot)$ denotes the categorical distribution. A parametric model $p_{\boldsymbol{\theta}}$ learns the reverse denoising; generation starts from all MASK and iteratively unmasks by sampling $p_{\boldsymbol{\theta}}(x_0^i|\boldsymbol{x}t)$. Recent theory (MDLM (Shi et al., 2024; Sahoo et al., 2024), RADD (Ou et al., 2024)) simplifies training from a variational bound to a reweighted cross-entropy over masked positions:

$$\mathcal{L}_{\mathrm{MDM}} = \int_0^1 \frac{1}{t} \mathbb{E}_{q_{t|0}(\boldsymbol{x}_t|\boldsymbol{x}_0)} \left[\sum_{i:x_t^i=\mathrm{MASK}} -\log p_{\boldsymbol{\theta}}(x_0^i|\boldsymbol{x}_t) \right] \mathrm{d}t \tag{2}$$

This formulation scales to LLMs as diffusion language models (DLMs), with LLaDA (Nie et al., 2025b) and Dream-7B (Ye et al., 2025) matching autoregressive performance while enabling parallel decoding and flexible infilling.

2.2 KEY-VALUE CACHE IN TRANSFORMERS

Transformer-based language models achieve computational efficiency during autoregressive generation through Key-Value (KV) caching (Pope et al., 2023). In causal attention, each layer projects the current hidden state $\mathbf{H}^t$ into query,

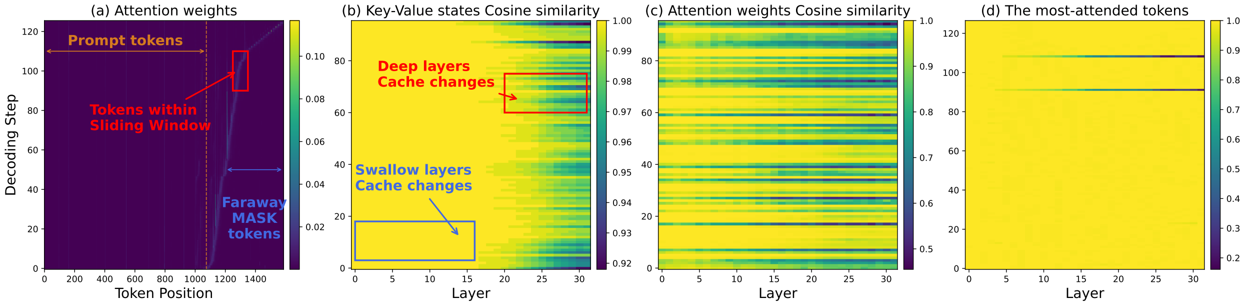


Figure 1: Visualization of our motivation. (a) MASK tokens located near each other receive high attention, while those situated far apart have minimal influence. (b) Over time, the representations in the KV states of cached tokens evolve, with deeper layers experiencing more substantial changes. (c) The changes in attention weights of most-attended tokens exhibit similar patterns to the changes in the KV states of all cached tokens. (d) The KV states of the most-attended tokens have the least changes.

key, and value representations using learned projection matrices $\mathbf{W_Q}, \mathbf{W_K}, \mathbf{W_V}$. At decoding step t, the attention computation for the current token follows:

$$\mathbf{A}_{[t]}^{t} = \text{softmax}\left(\frac{\mathbf{Q}_{[t]}^{t}(\mathbf{K}_{[1:t]}^{t})^{\top}}{\sqrt{d_k}}\right)\mathbf{V}_{[1:t]}^{t}, \qquad \text{KV cache: } \begin{cases} \mathbf{K}_{[1:t]}^{t} = \text{concat}(\mathbf{K}_{[1:t-1]}^{t-1}, \mathbf{K}_{[t]}^{t}), \\ \mathbf{V}_{[1:t]}^{t} = \text{concat}(\mathbf{V}_{[1:t-1]}^{t-1}, \mathbf{V}_{[t]}^{t}) \end{cases}. \tag{3}$$

To avoid redundant computation, previous key-value pairs are cached and reused. This caching strategy is effective because in causal attention, previously computed key-value pairs remain invariant throughout decoding ($\mathbf{K}_{[1:t-1]}^{t-1} = \mathbf{K}_{[1:t-1]}^{t}$), enabling efficient reuse without affecting model output.

KV-Cache in Bidirectional Attention. However, diffusion models employ bidirectional attention where all positions can attend to each other, breaking the invariance property of cached representations. As noted by dKV-Cache (Ma et al., 2025), token representations in diffusion models evolve dynamically during the iterative denoising process, making direct application of traditional KV-cache ineffective. The bidirectional dependencies cause previously computed key-value pairs to become stale as the sequence state changes, requiring careful redesign of caching strategies for diffusion language models.

3 METHODOLOGY

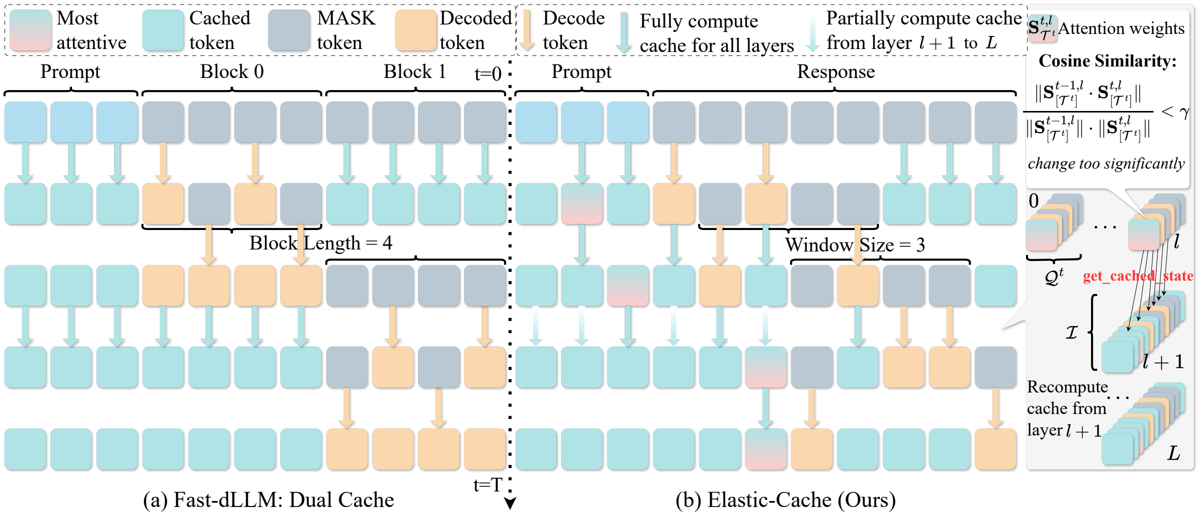
3.1 OUR FRAMEWORK OVERVIEW AND MOTIVATION

Diffusion LLMs differ from autoregressive decoders in that their key–value (KV) states evolve across denoising steps due to bidirectional dependencies. Our objective is to adaptively decide *when* and *where* to recompute the KV cache to preserve accuracy while minimizing latency. Baseline decoders recompute QKV for all tokens and layers at every step, despite *negligible* KV changes for most steps and especially in *shallow* layers (Fig. 1b); deeper layers exhibit larger drift. Rather than fixed-period refreshes (Wu et al., 2025; Ma et al., 2025; Liu et al., 2025), we propose **Elastic-Cache**, the first (to our knowledge) *adaptive, layer-aware* KV update policy for diffusion LLMs that jointly optimizes timing and location of recomputation.

Our design is driven by three observations. (1) Distant `MASK` tokens mainly act as a length prior and exert minimal influence on the current unmasking, we therefore block-cache their KV beyond the active prediction window (Fig. 1a). (2) KV drift grows with depth, refresh should start at a boundary layer and apply only to deeper layers (Fig. 1b). (3) The most-attended tokens typically shows the smallest KV change (Fig. 1d), giving a conservative lower bound for others, we use its drift as a lightweight trigger for refresh (Fig. 1c). Fig. 2 summarizes the pipeline. To the end, we proposed Elastic-Cache, a flexible method for key-value caching in diffusion large language models. Fig. 2 provides a visual representation of the overall pipeline of our proposed method.

3.2 SLIDING WINDOW DECODING AND KV CACHING

Formally, let $\mathcal{I} = \{1, 2, \ldots, N\}$ represent all positions. At decoding step t, let $\mathcal{D}^t$ denote newly decoded positions and $\mathcal{M}^t$ denote remaining masked positions, where $\mathcal{M}^{t-1} = \mathcal{M}^t \cup \mathcal{D}^t$. Denotes $\mathcal{D}^{<t} = \bigcup_i \{\mathcal{D}^i\}_{i=1}^{t}$ as the set of all


Most attentive
Cached token
MASK token
Decoded token
Decode token
Fully compute cache for all layers
Partially compute cache from layer $l+1$ to L
$\mathbf{S}^{t,l}_{\mathcal{T}^t}$ Attention weights
Cosine Similarity:
$\frac{\|\mathbf{S}^{t-1,l}_{[\mathcal{T}^t]} \cdot \mathbf{S}^{t,l}_{[\mathcal{T}^t]}\|}{\|\mathbf{S}^{t-1,l}_{[\mathcal{T}^t]}\| \cdot \|\mathbf{S}^{t,l}_{[\mathcal{T}^t]}\|} < \gamma$
change too significantly
Prompt
Block 0
Block 1
t=0
Prompt
Response
Block Length = 4
Window Size = 3
0
l
$\mathcal{Q}^t$
get_cached_state
$\mathcal{I}$
$l+1$
Recompute cache from layer $l+1$
L
t=T

(a) Fast-dLLM: Dual Cache (b) Elastic-Cache (Ours)

Figure 2: Illustration of the Key-Value cache method for diffusion LLMs. (a) The fast-dLLM (Wu et al., 2025) block-wise decoding method caches the Key-Value of all tokens outside the current block at each step. The KV cache is updated after completing a block of decoding. (b) Our proposed method, Elastic-Cache, caches the key-value of tokens outside a sliding window that flexibly moves through the sentence from left to right at each iteration. When the attention weights corresponding to the most-attended tokens (one for each layer) change significantly at a layer l, we start recomputing the KV cache from layer $l+1$ to the last layer.

decoded tokens up to time step t. Initially, at $t = 0$ we compute the attention for each layer l:

$$\mathbf{A}^{0,l}_{[\mathcal{I}]} = \mathrm{softmax}\left(\frac{\mathbf{Q}^{0,l}_{[\mathcal{I}]}(\mathbf{K}^{0,l}_{[\mathcal{I}]})^\top}{\sqrt{d_k}}\right)\mathbf{V}^{0,l}_{[\mathcal{I}]}, \qquad \text{initialize KV cache: } \begin{cases}\tilde{\mathbf{K}}^{0,l}_{[\mathcal{I}]} = \mathbf{K}^{0,l}_{[\mathcal{I}]} \\ \tilde{\mathbf{V}}^{0,l}_{[\mathcal{I}]} = \mathbf{V}^{0,l}_{[\mathcal{I}]}\end{cases}. \tag{4}$$

For each subsequence iteration t ranging from 1 to T, The model perform prediction for newly decoded position $\mathcal{D}^t$ and the remaining masked position $\mathcal{M}^t$. To enhance efficiency, we only perform predictions for masked positions that are closest to the left and form a sliding window of size β, denoted as $\mathcal{M}^t_\beta = \mathcal{M}^t_{[1:\beta]}$. We also have that $\mathcal{M}^{t-1}_\beta = \mathcal{M}^t_\beta \cup \mathcal{D}^t$. We've observed that masked positions within the sliding window tend to pay close attention to each other. Conversely, MASK tokens outside the sliding window receive minimal attention, allowing us to reuse their KV from the cache without significantly altering the prediction of the current masked position within the sliding window. We compute the attention only for the tokens in the sliding window $\mathcal{M}^{t-1}_\beta$ at step t as follows:

$$\mathbf{A}^{t,l}_{[\mathcal{M}^{t-1}_\beta]} = \mathrm{softmax}\left(\frac{\mathbf{Q}^{t,l}_{[\mathcal{M}^{t-1}_\beta]}(\tilde{\mathbf{K}}^{t,l}_{[\mathcal{I}]})^\top}{\sqrt{d_k}}\right)\tilde{\mathbf{V}}^{t,l}_{[\mathcal{I}]}, \qquad \text{update KV cache at } \mathcal{M}^{t-1}_\beta\text{: } \begin{cases}\tilde{\mathbf{K}}^{t,l}_{[\mathcal{M}^{t-1}_\beta]} = \mathbf{K}^{t,l}_{[\mathcal{M}^{t-1}_\beta]} \\ \tilde{\mathbf{V}}^{t,l}_{[\mathcal{M}^{t-1}_\beta]} = \mathbf{V}^{t,l}_{[\mathcal{M}^{t-1}_\beta]}\end{cases}. \tag{5}$$

Although the proposed sliding window decoding and KV cache share some similarities with the KV cache for block-wise decoding in Fast-dLLM (Wu et al., 2025) (as shown in the Fig. 2 for pipeline overview and comparison of both methods), we would like to emphasize the significant advancement of sliding window decoding. It ensures that tokens are close to each other and are predicted together, thereby minimizing the loss of cache for faraway MASK tokens. On the other hand, block-wise decoding may overlook those MASK tokens that are near the end of the block, leading to inefficient predictions due to overly aggressive caching of their surrounding context.

3.3 ATTENTION-AWARE KV CACHE UPDATE

The most important novelty of our proposed method is to automatically determine whether to update the KV cache to preserve accuracy while minimizing latency. Our method leverages the awareness of the model's attention weights to identify when the KV cache undergoes significant changes. At time step t and layer l, we determine the token that receives the most frequent attention from other tokens based on the attention weights corresponding to the current model's prediction for $\mathcal{M}^t_\beta$.

$$\mathcal{T}^{t,l} = \arg\max_{k \in \mathcal{D}^{<t}} \sum_{q \in \mathcal{M}^t_\beta} \mathbf{S}^{t,l}_{[q,k]}, \qquad \text{where: } \mathbf{S}^{t,l}_{[\mathcal{M}^t_\beta]} = \mathrm{softmax}\left(\frac{\mathbf{Q}^{t,l}_{[\mathcal{M}^t_\beta]}(\tilde{\mathbf{K}}^{t,l}_{[\mathcal{I}]})^\top}{\sqrt{d_k}}\right). \tag{6}$$

Algorithm 1 The Elastic-Cache Algorithm

```
Require: Trained model $p_{\boldsymbol{\theta}}$, Prompt $\mathbf{x}_{\text{prompt}}$, Sliding window size $\beta$, Update threshold $\gamma$, Generation length $N$.
Initialize: $\mathbf{x}^0 \leftarrow \{\mathbf{x}_{\text{prompt}}; \texttt{[MASK]}, \dots, \texttt{[MASK]}\}$;   $P \leftarrow \texttt{length}(\mathbf{x}_{\text{prompt}})$
$t \leftarrow 1$;   $\mathcal{D}^1 \leftarrow \{1, \dots, P\}$;   $\mathcal{M}^1 \leftarrow \{P+1, \dots, P+N\}$;   $\mathcal{T}^0 \leftarrow \emptyset$;   $\mathcal{I} = \mathcal{M}^0 = \mathcal{D}^1 \cup \mathcal{M}^1$
while $\mathcal{M}^t \neq \emptyset$ do
    $\mathcal{M}^t_\beta \leftarrow \mathcal{M}^t_{[:\beta]}$;   $\mathcal{Q}^t \leftarrow \mathcal{T}^{t-1} \cup \mathcal{M}^{t-1}_\beta$;   $\mathbf{H}^{t,0}_{[\mathcal{Q}^t]} \leftarrow \texttt{Embedding}(\mathbf{x}^t_{[\mathcal{Q}^t]})$;   $l^* \leftarrow \infty$
    for $l = 1, \dots, L$ do
        if $l > l^*$ then                                                    // Cache Update
            $\tilde{\mathbf{H}}^{t,l}_{[\mathcal{I}]}, \tilde{\mathbf{K}}^{t,l}_{[\mathcal{I}]}, \tilde{\mathbf{V}}^{t,l}_{[\mathcal{I}]} \leftarrow \texttt{cache\_update}(\mathcal{I})$;   $\mathbf{Q}^{t,l}_{[\mathcal{I}]}, \mathbf{K}^{t,l}_{[\mathcal{I}]}, \mathbf{V}^{t,l}_{[\mathcal{I}]} = \texttt{linear}(\mathbf{H}^{t,l}_{[\mathcal{I}]})$
            $\mathbf{H}^{t,l+1}_{[\mathcal{I}]}, \mathbf{S}^{t,l}_{[\mathcal{I}]} \leftarrow \texttt{attention\_layer}(\mathbf{Q}^{t,l}_{[\mathcal{I}]}, \mathbf{K}^{t,l}_{[\mathcal{I}]}, \mathbf{V}^{t,l}_{[\mathcal{I}]})$
        else                                                                 // Cache Reuse
            $\tilde{\mathbf{H}}^{t,l}_{[\mathcal{Q}^t]}, \tilde{\mathbf{K}}^{t,l}_{[\mathcal{Q}^t]}, \tilde{\mathbf{V}}^{t,l}_{[\mathcal{Q}^t]} \leftarrow \texttt{cache\_update}(\mathcal{Q}^t)$;   $\mathbf{Q}^{t,l}_{[\mathcal{Q}^t]}, \mathbf{K}^{t,l}_{[\mathcal{Q}^t]}, \mathbf{V}^{t,l}_{[\mathcal{Q}^t]} = \texttt{linear}(\mathbf{H}^{t,l}_{[\mathcal{Q}^t]})$
            $\mathbf{H}^{t,l+1}_{[\mathcal{Q}^t]}, \mathbf{S}^{t,l}_{[\mathcal{I}]} \leftarrow \texttt{attention\_layer}(\mathbf{Q}^{t,l}_{[\mathcal{Q}^t]}, \tilde{\mathbf{K}}^{t,l}_{[\mathcal{I}]}, \tilde{\mathbf{V}}^{t,l}_{[\mathcal{I}]})$
            $\sigma^{t,l} \leftarrow \texttt{cosine\_similarity}(\mathbf{S}^{t-1,l}_{[\mathcal{T}^{t-1}]}, \mathbf{S}^{t,l}_{[\mathcal{T}^{t-1}]})$
            if $\sigma^{t,k} < \gamma$ then                                   // Start update cache from layer $l+1$
                $l^* \leftarrow l$;   $\mathbf{H}^{t,l+1}_{[\mathcal{I}]} \leftarrow \texttt{get\_cached\_state}(\mathbf{H}^{t,l+1}_{[\mathcal{Q}^t]})$
            end if
        end if
        Get the most-attended token: $\mathcal{T}^{t,l} \leftarrow \arg\max_{k \in \mathcal{D}^{<t}} \sum_{q \in \mathcal{M}^t_\beta} \mathbf{S}^{t,l}_{[q,k]}$
    end for
    Decode new tokens: $\mathbf{x}^{t+1}, \mathcal{D}^{t+1} \leftarrow \texttt{decode}(\mathbf{x}^t, \mathcal{M}^t_\beta)$
    Update state: $\mathcal{M}^{t+1} \leftarrow \mathcal{M}^t \setminus \mathcal{D}^{t+1}$;   $\mathcal{T}^t = \bigcup_l \{\mathcal{T}^{t,l}\}_{l=1}^L$   $t \leftarrow t+1$      // State Update
end while
return $\mathbf{x}^{t-1}$
```

Here, we focus solely on the most-attended token among the current decoded tokens $\mathcal{D}^{<t}$. This is because the remaining MASK tokens either fall within the sliding window of predictions or have negligible influence on the unmasking tokens (Fig. 1a). We obtain one most-attended token per layer and compile the set of most-attended tokens, denoted as $\mathcal{T}^t = \bigcup_l \{\mathcal{T}^{t,l}\}_{l=1}^L$. In practice, the most-attended token for a layer often overlaps with tokens from other layers, resulting in a relatively limited number of most-attended tokens being available at any given time.

$\mathcal{T}^t$, besides being the tokens that have the most influence on the predictions' outcome, also signify the least changes among the cached decoded tokens (Fig. 1d). Therefore, we use $\mathcal{T}^t$ as a lightweight trigger for our cache update mechanism. Without updating all cached tokens, we only frequently update the most-attended tokens $\mathcal{T}^t$ to measure the degree of changes for all other cached tokens. Ideally, since $\mathcal{T}^t$ have the least change among the decoded tokens, we expect that when $\mathcal{T}^t$ change significantly, the rest of the decoded tokens will also change significantly. Therefore, we add $\mathcal{T}^{t-1}$ to the sliding window at step t: $\mathcal{T}^{t-1} \cup \mathcal{M}^{t-1}_\beta$. We then measure the changes in attention weights of $\mathcal{T}^t$ between the current and previous steps, t and $t-1$, using cosine similarity.

$$l^* = \begin{cases} l & \text{if: } \sigma^{t,l} < \gamma \\ \infty & \text{othewise} \end{cases}, \qquad \text{Cosine Similarity}(\mathbf{S}^{t-1,l}_{[\mathcal{T}^{t-1}]}, \mathbf{S}^{t,l}_{[\mathcal{T}^{t-1}]}) = \frac{\|\mathbf{S}^{t-1,l}_{[\mathcal{T}^{t-1}]} \cdot \mathbf{S}^{t,l}_{[\mathcal{T}^{t-1}]}\|}{\|\mathbf{S}^{t-1,l}_{[\mathcal{T}^{t-1}]}\| \cdot \|\mathbf{S}^{t,l}_{[\mathcal{T}^{t-1}]}\|} = \sigma^{t,l}. \tag{7}$$

The changes in attention $\mathbf{S}^{t,l}$ directly affect the output of the current layer or the input of the next layer $\mathbf{H}^{t,l+1}$. This implies that our cached values are diverging from the actual values, necessitating an update. When a layer l^* observes significant changes in attention weights $\sigma^{t,l} < \gamma$, we initiate the update of the KV cache for the subsequent layers, starting from $l^* + 1$ and continuing until the last layer L. To achieve this, we initialize the hidden states of all cached tokens with the states $\tilde{\mathbf{H}}^{t,l+1}_{[\mathcal{I}]}$, which have been saved and updated using patterns similar to $\tilde{\mathbf{K}}^{t,l+1}_{[\mathcal{I}]}$ and $\tilde{\mathbf{V}}^{t,l+1}_{[\mathcal{I}]}$.

$$\text{Update state: } \tilde{\mathbf{H}}^{t,l+1}_{[\mathcal{M}^{t-1}_\beta]} = \mathbf{H}^{t,l+1}_{[\mathcal{M}^{t-1}_\beta]}, \qquad \text{Initialize cache update: } \mathbf{Q}^{t,l+1}_{[\mathcal{I}]}, \mathbf{K}^{t,l+1}_{[\mathcal{I}]}, \mathbf{V}^{t,l+1}_{[\mathcal{I}]} = \text{linear}(\tilde{\mathbf{H}}^{t,l+1}_{[\mathcal{I}]}) \tag{8}$$

We then update and overwrite the KV cache using the same process as initially at $t = 0$, as described in Eq. 4. If none of the layers satisfy $\sigma^{t,l} < \gamma$, we continue to reuse our KV cache for future predictions.

We didn't directly compare the hidden state $\mathbf{H}^{t,l+1}$ and $\mathbf{H}^{t-1,l+1}$ because their changes depend on various network components. The error in measurement could be amplified by the divergence between the cached value and the actual value (including Key-Value states).

On the other hand, the changes in attention weights are closely linked to the source of the change in Key-Value states, which is the bidirectional attention mechanism in diffusion LLMs. Intuitively, the changes in attention weights become significant when new decoded tokens receive high attention and alter the attention output computed in the past when they were still masked. Consequently, the changes in attention weights exhibit very similar patterns to the changes in Key-Value states during decoding, as illustrated in Fig. 1b and Fig. 1c.

We use the hyper-parameter γ to set the trigger for automatic cache updates. As shown in Fig. 1c, the attention weights' cosine similarity landscapes influence this. A higher γ results in more frequent and extensive cache updates across multiple layers, while a lower γ triggers updates less frequently. This flexibility allows us to effectively manage the trade-off between accuracy and latency. Our overall algorithm is described in Algorithm 1.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Implementation Details. All our runs use a single NVIDIA A100 80GB. We evaluate **Elastic-Cache** on LLaDA-Instruct (Nie et al., 2025a), LLaDA-1.5 (Zhu et al., 2025), and multimodal LLaDA-V (You et al., 2025) across MBPP (Austin et al., 2021b), HumanEval (Chen et al., 2021), MATH (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), MathVista (Lu et al., 2023), and MathVerse (Zhang et al., 2024b). Default hyperparameters: attention threshold γ=0.9, parallel-decoding confidence ϵ=0.9, cache block size 32. For fair comparison, we re-run LLaDA (Nie et al., 2025a) and Fast-dLLM (Wu et al., 2025) under the same hardware/software.

Evaluation Framework and Metrics. We use `lm-eval-harness` (Gao et al., 2024). Throughput is tokens/sec averaged until emitting, matching Fast-dLLM's protocol (Wu et al., 2025). Accuracy metrics: GSM8K: 5-shot `flexible_extract` (Cobbe et al., 2021); MATH: 4-shot `math_verify` (`minerva_math`) (Hendrycks et al., 2021); HumanEval—0-shot with the Fast-dLLM post-processing (Chen et al., 2021; Wu et al., 2025); MBPP—3-shot `pass@1` (Austin et al., 2021b). For LLaDA-V, we adopt the official pipeline with `lmms-eval` (Zhang et al., 2024a; Li et al., 2024): MathVista: `gpt_eval_score` (Lu et al., 2023); MathVerse: `gpt_eval_score` on `mathverse_testmini_vision_dominant` (Zhang et al., 2024b).

Confidence-Aware Decoding. We employ confidence-aware decoding strategies from Fast-dLLM (Wu et al., 2025), which select only tokens with confidence scores exceeding a specified threshold (ϵ), instead of unmasking a fixed number of tokens per step, as in the baseline Diffusion LLM. This straightforward yet effective approach accelerates Diffusion LLM inference by enabling more tokens to be predicted concurrently at each iteration, contingent upon the model's performance. Consequently, we concentrate on comparing the acceleration achieved by the KV caching method under the same decoding strategies.

4.2 PERFORMANCE AND EFFICIENCY EVALUATION

Across Tables 1, 2, and 3, our proposed **Elastic-Cache** delivers substantial throughput gains for diffusion LLMs with minimal accuracy loss. By adaptively updating the cache only when necessary, it achieves a speedup of up to 45.1× over the standard baseline. While maintaining accuracy within 1$\sim$2% on MATH and HumanEval, it also achieves higher accuracy on GSM8K and MBPP. Compared to Fast-dLLM (Wu et al., 2025), Elastic-Cache consistently attains greater tokens/sec at better accuracy.

As presented in Table 1, on LLaDA-Instruct, **Elastic-Cache** reaches 90.1 t/s on GSM8K (512 tokens; 25.2× over baseline) at 77.71% accuracy, surpassing Fast-dLLM's 44.0 t/s @ 74.83%. On LLaDA-1.5 (Table 2), our approach yields even greater gains, including 45.1× on GSM8K-512, with an accuracy of 81.35% (baseline 81.35%). This observation indicates that Elastic-Cache performs better when the model's predictions are more accurate. The reason behind this could be the close relationship between our approach and attention scores. Intuitively, accurate predictions are associated with meaningful attention scores with fewer outliers, which makes our approach operate more smoothly.

We also observed that in most settings, Elastic-Cache provides higher throughput for longer generation lengths, whereas this is the opposite for Fast-dLLM (Wu et al., 2025), as it often experiences reduced throughput as the generation length increases. The advantages of our approach stem from the fixed-size sliding window and automatic cache update, which minimizes the dependency of throughput on the generation length.

In the multimodal setting (LLaDA-V; Table 3), **Elastic-Cache** raises MathVerse-256 throughput to 32.3 t/s from Fast-dLLM's 30.3 t/s while maintaining 29.19% accuracy, demonstrating robustness beyond text-only tasks. The significant improvement of Elastic-Cache compared to the baselines across various settings suggests that our method is broadly applicable and has high scalability potential.

Table 1: Comprehensive benchmark results on the LLaDA-Instruct suite. Each cell shows accuracy (top) and decoding throughput in tokens/sec with relative speedup to the LLaDA baseline (bottom, blue: t/s / orange: speedup). Highlighted cells denote the highest throughput and speedup per configuration. The highest accuracy is bolded.

			Confident-Aware Decoding		
Benchmark	**Gen Length**	**LLaDA**	**LLaDA**	**Fast-dLLM**	**Elastic-Cache**
GSM8K (5-shot)	256	78.01 7.3 (1.0×)	78.62 22.8 (3.1×)	77.94 53.7 (7.7×)	**78.24** 58.0 (8.2×)
	512	77.10 3.6 (1.0×)	77.33 18.6 (5.2×)	74.83 44.0 (12.3×)	**77.71** 90.1 (25.2×)
MATH (4-shot)	256	33.58 9.5 (1.0×)	33.28 25.8 (2.7×)	32.50 49.0 (5.1×)	**33.14** 48.7 (5.1×)
	512	37.20 7.1 (1.0×)	36.82 24.0 (3.4×)	35.70 52.8 (7.4×)	**36.60** 59.3 (7.9×)
HumanEval (0-shot)	256	40.85 33.3 (1.0×)	42.07 102.1 (3.1×)	37.20 99.8 (3.0×)	**40.24** 160.5 (4.8×)
	512	43.90 17.7 (1.0×)	43.29 51.6 (2.9×)	45.73 76.1 (4.3×)	**46.34** 100.7 (5.0×)
MBPP (3-shot)	256	29.80 6.5 (1.0×)	30.00 23.4 (3.6×)	25.40 45.1 (7.0×)	**32.2** 46.9 (7.3×)
	512	15.0 4.7 (1.0×)	15.0 20.8 (4.4×)	13.6 44.7 (9.5×)	**15.6** 63.0 (13.4×)

4.3 ABLATIONS

We ablate key choices: 1) Cache update threshold γ, 2) sliding window size β, and 3) prefill and generation length, to expose speed/accuracy trade-offs and justify defaults.

Cache Update Threshold (γ). Table 4 illustrates the sensitivity of our proposed method to the parameter γ. As γ is used to control the frequency of cache updates, a consistent decrease in γ leads to an increase in throughput. However, there is also a trend of decreasing accuracy as throughput increases. The trend is more consistent for the LLaDA-1.5 model, while for LLaDA, the accuracy at peak ($\gamma = 0.9$) is higher, but the throughput is lower.

Sliding Window Size (β). Fig. 3a shows that our accuracy is stable across various β and close to No-Cache until $\beta \approx 64$; beyond that LLaDA's tendency to emit EOS early degrades results (You et al., 2025). Throughput, however, is sensitive to β: larger windows enable more parallel prediction (fewer iterations, lower latency), but overly large β reduces cacheable MASK tokens, raising per-step compute and latency.

Sliding Window vs. Block-Wise. When switching Elastic-Cache to block-wise decoding (Fast-dLLM-style) (Fig. 3a), our accuracy is often similar to No-Cache, but short blocks hurt accuracy and throughput diverges. Our sliding window groups nearby MASK tokens that strongly attend to each other, whereas block-wise caching over-aggressively freezes distant MASKs, harming small-block predictions. Our Elastic-Cache's automatic cache refresh detects divergent tokens and updates them, preserving accuracy at the cost of some throughput.

Prefill and Generation Length. Table 5a and Table 5b provide insights into the impact of prefill length and generation length on the overall speedup. Notably, both Fast-dLLM and Elastic-Cache experience a decrease in throughput as the prefill length increases from 3-shot to 8-shot. However, Elastic-Cache exhibits a remarkable speedup and consistently high accuracy across different prefill lengths. Moreover, the throughput of Elastic-Cache increases with generation length, highlighting its unique scaling properties.

4.4 ANALYSIS

Cache update frequency. Fig. 3b and Fig. 3c illustrate the frequency of cache updates performed by Elastic-Cache under varying hyper-parameters γ and ϵ. The proposed method maintains a very low cache update frequency across different values of γ (Fig. 3b). In extreme cases, with $\gamma = 0.95$, the cache update frequency increases to only 20% compared to the baseline without a cache. Moreover, increasing the model's confidence and accuracy (with ϵ, Fig. 3c) enhances Elastic-Cache's effectiveness, and reduces the cache update frequency.

Tunable Speed–Accuracy Trade-off. The cache update threshold γ directly determines the balance (Table 4). An excessively high γ could lead to unnecessary cache updates, resulting in a decrease in speedup without any improvement in accuracy. Conversely, a smaller γ value could guarantee speedup while sacrificing accuracy. The optimal value for

Table 2: Comprehensive benchmark results on the LLaDA-1.5 suite. Each cell shows accuracy (top) and decoding throughput in tokens/sec with relative speedup to the LLaDA baseline (bottom, blue: t/s / orange: speedup). Highlighted cells denote the highest throughput and speedup per configuration.

Benchmark	Gen Length	LLaDA-1.5	Confident-Aware Decoding		
			LLaDA-1.5	**Fast-dLLM**	**Elastic-Cache**
GSM8K (5-shot)	256	80.36 6.7 (1.0×)	80.44 22.5 (3.3×)	80.59 51.2 (7.6×)	**81.50** 58.0 (8.7×)
	512	81.35 2.6 (1.0×)	81.88 17.2 (6.6×)	80.82 36.8 (14.1×)	**81.35** 117.2 (45.1×)
MATH (4-shot)	256	33.52 8.5 (1.0×)	33.60 22.3 (2.6×)	32.74 44.4 (5.2×)	**33.50** 51.0 (6.5×)
	512	35.63 5.0 (1.0×)	35.56 20.3 (4.0×)	33.68 44.4 (8.8×)	**35.36** 74.8 (14.9×)
HuamnEval (0-shot)	256	43.29 7.0 (1.0×)	42.68 17.5 (2.5×)	34.75 18.7 (2.7×)	**36.59** 20.9 (3.0×)
	512	40.85 3.2 (1.0×)	39.63 9.7 (3.1×)	36.59 15.4 (4.8×)	**37.80** 16.8 (5.3×)
MBPP (3-shot)	256	38.00 2.4 (1.0×)	38.00 14.2 (5.8×)	34.60 28.0 (11.6×)	**41.20** 32.7 (13.5×)
	512	38.20 1.0 (1.0×)	38.60 11.5 (11.5×)	36.20 17.8 (17.8×)	**39.00** 32.8 (32.8×)

Table 3: Performance and Speedup Comparison of LLaDA-V on MathVista and MathVerse. Each benchmark presents results from LLaDA-V (base) using Fast-dLLM, and our method.

Length	MathVista		MathVerse	
	Fast-dLLM	**Elastic-Cache (Ours)**	**Fast-dLLM**	**Elastic-Cache (Ours)**
256	55.9 28.7	55.9 29.7	26.78 30.3	29.19 32.3
512	54.1 23.7	55.8 24.1	25.5 28.1	29.19 30.8

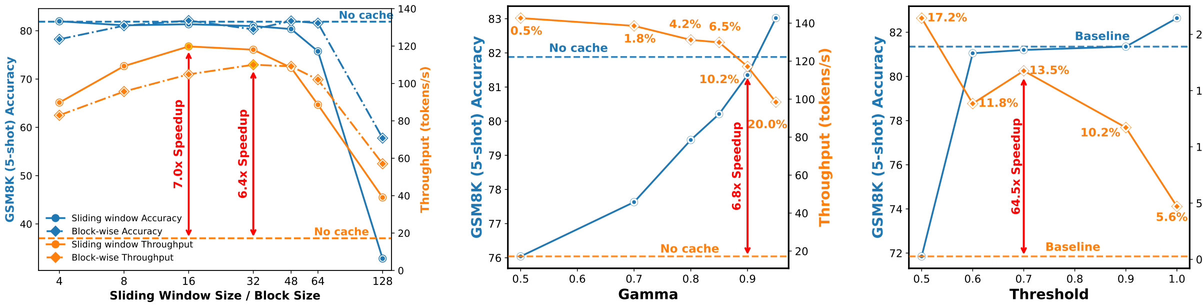
γ to maximize both accuracy and throughput depends on the model's prediction. Models with higher accuracy tend to have the best γ value, which is closer to 1.0 (Table 4).

Scaling Properties. Elastic-Cache scales greatly with the generation length and the power of the base model. Increasing the generation length slows down the baseline performance but speeds up Elastic-Cache (Tables 5b). Moreover, Elastic-Cache effectiveness is highly dependent on the accuracy of the model's predictions (Table 1, Table 2, Fig. 3c). This indicates that Elastic-Cache can effectively scale with the size of the model and the size of the training data, as LLMs generally improve when they scale up.

5 RELATED WORK

Diffusion Language Models. Classical diffusion models excel in continuous domains: images (Ho et al., 2020; Dhariwal & Nichol, 2021; Rombach et al., 2022), audio (Yang et al., 2023; Huang et al., 2023), and video (Xing et al., 2024; Ho et al., 2022a;b), building on the seminal formulation of Sohl-Dickstein et al. (2015). Adapting diffusion to discrete text has followed Markov/multinomial/continuous-time paths (Austin et al., 2021a; Hoogeboom et al., 2021b;a; Campbell et al., 2022; Sun et al., 2022), refined via score matching, ratio methods, and reparameterization (Meng et al., 2022; Lou & Ermon, 2023; Zheng et al., 2023), with recent work unifying these views (Sahoo et al., 2024; Shi et al., 2024; Ou et al., 2024; Zheng et al., 2024). Early NLP systems validated these ideas (He et al., 2022; Li et al., 2022; Gong et al., 2022) and explored semi-autoregression (Han et al., 2022). Masked diffusion approaching autoregressive quality (Sahoo et al., 2024) enabled scalable models (LLaDA) competitive with LLaMA (Nie et al., 2025a; 2024; Touvron et al., 2023a; Dubey et al., 2024), with further gains from AR adaptation and instruction tuning (Gong et al., 2024; Zhu et al., 2025; Ye et al., 2025). The paradigm now spans multimodal/structured domains (You et al., 2025; Yang et al., 2025; Yu et al., 2025; Wang et al., 2024a;b; Kitouni et al., 2023).

Acceleration Techniques for Large Language Models. KV caching underpins efficient transformer inference (Vaswani et al., 2017; Pope et al., 2023), complemented by GQA, RoPE, and modern LLM optimizations (Ainslie et al., 2023; Su et al., 2024; Touvron et al., 2023a;b; Dubey et al., 2024). Diffusion LLMs complicate caching due to bidirectional attention and evolving representations; dedicated methods include Fast-dLLM (Wu et al., 2025), dKV-



(a) Impact of Sliding window and block size (b) Cache update frequency. (c) Confident-aware decoding.

Figure 3: Ablation study and analysis of our proposed method. (a) Ablation study of our sliding window mechanism compared to block-wise decoding. (b) Analysis of cache update frequency under varying γ. The blue and orange lines represent accuracy and throughput, respectively. The numbers along the lines indicate the frequency of cache updates, assuming no baseline. (c) Analysis of cache update frequency under confident-aware decoding with varying ϵ.

Table 4: Impact of attention threshold on accuracy and speedup under GSM8K (5-Shot) for LLaDA and LLaDA1.5 with generation length of 512.

			Elastic-Cache (Ours)					
Model	**No Cache**	**Fast-dLLM**	$\gamma = 0.5$	$\gamma = 0.7$	$\gamma = 0.8$	$\gamma = 0.85$	$\gamma = 0.9$	$\gamma = 0.95$
LLaDA	77.10 3.6 (1.0×)	74.83 44.0 (12.2×)	71.57 109.9 (30.5×)	73.46 108.7 (30.2×)	74.30 103.9 (28.9×)	74.68 99.1 (27.5×)	77.71 91.5 (25.4×)	76.72 75.5 (21.0×)
LLaDA-1.5	81.35 2.6 (1.0×)	80.82 36.8 (14.2×)	76.04 142.7 (54.9×)	77.63 138.6 (53.3×)	79.45 131.2 (50.5×)	80.21 129.9 (50.0×)	81.35 117.2 (45.1×)	83.02 98.4 (37.8×)

Table 5: Comparison between Elastic-Cache and Fast-dLLM when varying few-shots and generation length.

(a) Impact of few-shots on Accuracy and Speedup Under GSM8K (generation length of 1024) for LLaDA.

Model.	**3-shot**	**5-shot**	**8-shot**
Fast-dLLM	73.77 28.5 (1.0×)	76.04 25.0 (1.0×)	75.36 20.8 (1.0×)
Elastic-Cache	75.13 185.3 (6.5×)	75.21 169.8 (6.8×)	75.28 143.9 (6.9×)

(b) Impact of generation length on Accuracy and Speedup Under GSM8K (5-Shot), $\gamma = 0.8$ for LLaDA.

Model.	**256**	**512**	**1024**
Fast-dLLM	77.94 53.7 (1.0×)	74.83 44.0 (1.0×)	76.04 25.0 (1.0×)
Elastic-Cache	78.24 58.0 (1.1×)	77.71 91.5 (2.1×)	75.21 169.8 (6.8×)

Cache (Ma et al., 2025), and DeepCache (Ma et al., 2024). Orthogonal accelerations exploit parallel/non-AR generation (Gu et al., 2017; Xiao et al., 2023), block-wise diffusion (Arriola et al., 2025), fast sampling (Chen et al., 2023), test-time scaling (Ramesh & Mardani, 2025), and consistency models (Kou et al., 2024). However, most rely on temporal heuristics or fixed thresholds, leaving attention patterns underused. **Our Perspective.** We close this gap with attention-aware and layer-aware caching for diffusion LLMs: tracking most-attended tokens and depth-varying KV dynamics to guide recomputation, complementary to interval-based (Ma et al., 2025) and confidence-based (Wu et al., 2025) policies and compatible with the broader acceleration toolkit (Ainslie et al., 2023; Su et al., 2024; Touvron et al., 2023a;b; Dubey et al., 2024; Gu et al., 2017; Xiao et al., 2023; Arriola et al., 2025; Chen et al., 2023; Ramesh & Mardani, 2025; Kou et al., 2024).

6 CONCLUSION

We presented **Elastic-Cache**, a training-free, architecture-agnostic policy that makes KV caching in diffusion LLMs adaptive along two axes: *when* to refresh (via an attention-aware drift test) and *where* to refresh (via a depth-selective update starting at a learned boundary layer). By block-caching distant MASK tokens, reusing shallow-layer caches, and refreshing only when the most-attended token indicates meaningful state change, Elastic-Cache removes large amounts of redundant QKV work. Across decoding steps, this yields substantial latency reductions with negligible impact on generation quality, addressing a key deployment bottleneck for diffusion decoders. Looking ahead, we plan to refine drift thresholds with learned predictors, formalize guarantees linking attention patterns to KV drift, and explore interplay with speculative decoding or other hardware-aware scheduling, extending the same principles to autoregressive LLMs and multimodal diffusion frameworks.

ETHICS STATEMENT

This work targets inference-time efficiency for diffusion LLMs and does not introduce new data collection or model training. All evaluations use publicly available datasets and third-party checkpoints under their original licenses, no personally identifiable information is processed. While faster decoding can lower the cost of generation and thus broaden access, it may also amplify misuse. We neither change safety filters nor attempt to bypass alignment constraints of the underlying models. We will document evaluation prompts and tasks, follow the usage policies of model providers, and encourage human oversight for downstream deployments, especially in high-stakes applications.

REPRODUCIBILITY STATEMENT

Elastic-Cache is training-free and defined by a small set of inference hyperparameters: the attention similarity threshold γ, block size and generation length. We will release code, configs, and scripts to reproduce all results: (i) reference implementations of Attention-Aware and Layer-Aware KV updates with ablation; (ii) exact prompts/datasets, metrics, and other criteria; and (iii) environment specs (CUDA/driver, framework versions) and hardware details (GPU type, batch sizes). We report wall-clock latency and accuracy metrics for each setting, and provide logs to our tables/figures from raw traces.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, 2023.

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models, 2025. URL `https://arxiv.org/abs/2503.09573`.

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

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

Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 35:28266–28279, 2022.

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.

Zixiang Chen, Huizhuo Yuan, Yongqian Li, Yiwen Kou, Junkai Zhang, and Quanquan Gu. Fast sampling via derandomization for discrete diffusion models. *arXiv preprint arXiv:2312.09193*, 2023.

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

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

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

Bolin Gao and Lacra Pavel. On the properties of the softmax function with application in game theory and reinforcement learning. *arXiv preprint arXiv:1704.00805*, 2017.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. URL `https://zenodo.org/records/12608602`.

Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and LingPeng Kong. Diffuseq: Sequence to sequence text generation with diffusion models. *arXiv preprint arXiv:2210.08933*, 2022.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, et al. Scaling diffusion language models via adaptation from autoregressive models. *arXiv preprint arXiv:2410.17891*, 2024.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*, 2017.

Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. Ssd-lm: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. *arXiv preprint arXiv:2210.17432*, 2022.

Zhengfu He, Tianxiang Sun, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. Diffusionbert: Improving generative masked language models with diffusion models. *arXiv preprint arXiv:2211.15029*, 2022.

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.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022a.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *Advances in neural information processing systems*, 35:8633–8646, 2022b.

Emiel Hoogeboom, Alexey A Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. *arXiv preprint arXiv:2110.02037*, 2021a.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in neural information processing systems*, 34:12454–12465, 2021b.

Rongjie Huang, Jiawei Huang, Dongchao Yang, Yi Ren, Luping Liu, Mingze Li, Zhenhui Ye, Jinglin Liu, Xiang Yin, and Zhou Zhao. Make-an-audio: Text-to-audio generation with prompt-enhanced diffusion models, 2023. URL `https://arxiv.org/abs/2301.12661`.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does bert learn about the structure of language? In *ACL 2019-57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Ouail Kitouni, Niklas Nolte, James Hensman, and Bhaskar Mitra. Disk: A diffusion model for structured knowledge. *arXiv preprint arXiv:2312.05253*, 2023.

Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. Cllms: Consistency large language models. *arXiv preprint arXiv:2403.00835*, 2024.

Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. Revealing the dark secrets of bert. *arXiv preprint arXiv:1908.08593*, 2019.

Bo Li, Peiyuan Zhang, Kaichen Zhang, Fanyi Pu, Xinrun Du, Yuhao Dong, Haotian Liu, Yuanhan Zhang, Ge Zhang, Chunyuan Li, and Ziwei Liu. Lmms-eval: Accelerating the development of large multimodal models, March 2024. URL `https://github.com/EvolvingLMMs-Lab/lmms-eval`.

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

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. *Advances in neural information processing systems*, 35:4328–4343, 2022.

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.

Aaron Lou and Stefano Ermon. Reflected diffusion models. In *International Conference on Machine Learning*, pp. 22675–22701. PMLR, 2023.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15762–15772, 2024.

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

Chenlin Meng, Kristy Choi, Jiaming Song, and Stefano Ermon. Concrete score matching: Generalized score matching for discrete data. *Advances in Neural Information Processing Systems*, 35:34532–34545, 2022.

Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. Scaling up masked diffusion models on text. *arXiv preprint arXiv:2410.18514*, 2024.

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*, 2025a.

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, 2025b. URL `https://arxiv.org/abs/2502.09992`.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *arXiv preprint arXiv:2406.03736*, 2024.

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.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Vignav Ramesh and Morteza Mardani. Test-time scaling of diffusion models via noise trajectory search. *arXiv preprint arXiv:2506.03164*, 2025.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works. *Transactions of the association for computational linguistics*, 8:842–866, 2021.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. URL `https://arxiv.org/abs/2112.10752`.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *arXiv preprint arXiv:2406.07524*, 2024.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K Titsias. Simplified and generalized masked diffusion for discrete data. *arXiv preprint arXiv:2406.04329*, 2024.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. PMLR, 2015.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Haoran Sun, Lijun Yu, Bo Dai, Dale Schuurmans, and Hanjun Dai. Score-based continuous-time discrete diffusion models. *arXiv preprint arXiv:2211.16750*, 2022.

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*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

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.

Xinyou Wang, Zaixiang Zheng, Fei Ye, Dongyu Xue, Shujian Huang, and Quanquan Gu. Diffusion language models are versatile protein learners. *arXiv preprint arXiv:2402.18567*, 2024a.

Xinyou Wang, Zaixiang Zheng, Fei Ye, Dongyu Xue, Shujian Huang, and Quanquan Gu. Dplm-2: A multimodal diffusion protein language model. *arXiv preprint arXiv:2410.13782*, 2024b.

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.

Yisheng Xiao, Lijun Wu, Junliang Guo, Juntao Li, Min Zhang, Tao Qin, and Tie yan Liu. A survey on non-autoregressive generation for neural machine translation and beyond, 2023. URL `https://arxiv.org/abs/2204.09269`.

Zhen Xing, Qijun Feng, Haoran Chen, Qi Dai, Han Hu, Hang Xu, Zuxuan Wu, and Yu-Gang Jiang. A survey on video diffusion models. *ACM Computing Surveys*, 57(2):1–42, 2024.

Dongchao Yang, Jianwei Yu, Helin Wang, Wen Wang, Chao Weng, Yuexian Zou, and Dong Yu. Diffsound: Discrete diffusion model for text-to-sound generation, 2023. URL `https://arxiv.org/abs/2207.09983`.

Ling Yang, Ye Tian, Bowen Li, Xinchen Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. Mmada: Multimodal large diffusion language models. *arXiv preprint arXiv:2505.15809*, 2025.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b, 2025. URL `https://hkunlp.github.io/blog/2025/dream`.

Zebin You, Shen Nie, Xiaolu Zhang, Jun Hu, Jun Zhou, Zhiwu Lu, Ji-Rong Wen, and Chongxuan Li. Llada-v: Large language diffusion models with visual instruction tuning. *arXiv preprint arXiv:2505.16933*, 2025.

Runpeng Yu, Xinyin Ma, and Xinchao Wang. Dimple: Discrete diffusion multimodal large language model with parallel decoding, 2025. URL `https://arxiv.org/abs/2505.16990`.

Kaichen Zhang, Bo Li, Peiyuan Zhang, Fanyi Pu, Joshua Adrian Cahyono, Kairui Hu, Shuai Liu, Yuanhan Zhang, Jingkang Yang, Chunyuan Li, and Ziwei Liu. Lmms-eval: Reality check on the evaluation of large multimodal models, 2024a. URL `https://arxiv.org/abs/2407.12772`.

Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Yu Qiao, et al. Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems? In *European Conference on Computer Vision*, pp. 169–186. Springer, 2024b.

Kaiwen Zheng, Yongxin Chen, Hanzi Mao, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. *arXiv preprint arXiv:2409.02908*, 2024.

Lin Zheng, Jianbo Yuan, Lei Yu, and Lingpeng Kong. A reparameterized discrete diffusion model for text generation. *ArXiv*, abs/2302.05737, 2023.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Llada 1.5: Variance-reduced preference optimization for large language diffusion models, 2025. URL `https://arxiv.org/abs/2505.19223`.

APPENDIX

CONTENTS

A Theoretical Validation for Elastic-Cache **15**

A.1 Notation and Setup 15

A.2 Background Lemmas and Assumptions 15

A.3 Layer-Wise KV Drift Monotonicity 16

A.4 Attention Concentration and Drift 19

A.5 Implications for Elastic-Cache 22

B Detailed Experiment Setup **22**

C Batch Implementation of Elastic-Cache **23**

D Use of Large Language Models **23**

E Sample Response **23**

A THEORETICAL VALIDATION FOR ELASTIC-CACHE

A.1 NOTATION AND SETUP

- L: number of transformer layers, indexed by $\ell \in \{1, \ldots, L\}$
- T: total denoising steps, indexed by $t \in \{0, \ldots, T\}$
- N: sequence length
- d: hidden dimension; d_k, d_v: key and value dimensions
- $\mathbf{H}_i^{t,\ell} \in \mathbb{R}^d$: hidden state of token i at step t, layer ℓ
- $\mathbf{K}_i^{t,\ell}, \mathbf{V}_i^{t,\ell} \in \mathbb{R}^{d_k}, \mathbb{R}^{d_v}$: key and value of token i
- $\mathbf{S}^{t,\ell} \in \mathbb{R}^{N \times N}$: attention weights at step t, layer ℓ
- $\mathcal{D}^{<t}$: decoded token positions up to step $t-1$
- $\mathcal{M}^t$: masked token positions at step t
- $\mathcal{M}_\beta^t$: sliding window of size β over masked positions
- $\alpha_k^{t,\ell} := \sum_{q \in \mathcal{M}_\beta^t} \mathbf{S}_{q,k}^{t,\ell}$: total attention token k receives
- $\Delta \mathbf{H}_i := \mathbf{H}_i^{t,\ell} - \mathbf{H}_i^{t-1,\ell}$: change in hidden state
- $\bar{\Delta}^{t,\ell} := \frac{1}{N} \sum_{i=1}^{N} \|\Delta \mathbf{H}_i^{t,\ell}\|_2$: average hidden state drift
- $\Delta_{\max} := \max_i \|\Delta \mathbf{H}_i\|_2$: maximum hidden state change
- $\Gamma^{t,\ell} := \alpha_{\mathcal{T}^{t,\ell}}^{t,\ell} - \max_{k \neq \mathcal{T}^{t,\ell}} \alpha_k^{t,\ell} \geq 0$: Attention Gap

A.2 BACKGROUND LEMMAS AND ASSUMPTIONS

Lemma A.1 (Lipschitz Continuity of Softmax)**.** *Based on the Proposition 2 in Gao & Pavel (2017), the softmax function* $\sigma : \mathbb{R}^n \to \Delta^{n-1}$ *defined by*

$$\sigma(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)} \tag{9}$$

is 1*-Lipschitz continuous with respect to the* ℓ_2 *norm:*

$$\|\sigma(\mathbf{z}) - \sigma(\mathbf{z}')\|_2 \leq \|\mathbf{z} - \mathbf{z}'\|_2, \quad \forall \mathbf{z}, \mathbf{z}' \in \mathbb{R}^n \tag{10}$$

Assumption A.2 (Bounded Representations)**.** At each layer ℓ and step t: $\left\|\mathbf{H}_i^{t,\ell}\right\|_2 \leq R_\ell$

Assumption A.3 (Lipschitz Network Components)**.** The projection matrices satisfy $\left\|\mathbf{W}_Q^\ell\right\|_2, \left\|\mathbf{W}_K^\ell\right\|_2, \left\|\mathbf{W}_V^\ell\right\|_2 \leq W_{\max}$. The feedforward network at layer ℓ is L_{FFN}-Lipschitz continuous.

Assumption A.4 (Progressive Unmasking)**.** At each step t, a non-empty subset $\mathcal{D}^t \subseteq \mathcal{M}^{t-1}$ is unmasked: $|\mathcal{D}^{<t}|$ increases and $\mathcal{M}^t = \mathcal{M}^{t-1} \setminus \mathcal{D}^t$.

Assumption A.5 (Layer-Wise Representation Dynamics)**.** There exists $\ell^* \in \{1, \ldots, L\}$ and functions $f_\ell(t) \to 0$ as $t \to T$ for $\ell \leq \ell^*$ such that:

- **Shallow layers** ($\ell \leq \ell^*$): The expected hidden state change for decoded tokens vanishes: For $\ell \leq \ell^*$:

$$\mathbb{E}[\left\|\mathbf{H}_i^{t,\ell} - \mathbf{H}_i^{t-1,\ell}\right\|_2 \mid i \in \mathcal{D}^{<t}] \leq f_\ell(t) \to 0$$

- **Deep layers** ($\ell > \ell^*$): The expected change remains bounded away from zero:

$$\liminf_{t \to T} \mathbb{E}[\left\|\mathbf{H}_i^{t,\ell} - \mathbf{H}_i^{t-1,\ell}\right\|_2 \mid i \in \mathcal{D}^{<t}] \geq c_\ell > 0$$

This reflects that early layers encode local lexical patterns that stabilize quickly, while deep layers encode semantic relationships that continue evolving (Kovaleva et al., 2019; Jawahar et al., 2019; Rogers et al., 2021). Our experiments validate this (Figure 1b).

Assumption A.6 (Attention Concentration)**.** The attention gap is a non-negligible fraction of total attention mass:

$$\Gamma^{t,\ell} \geq c \cdot |\mathcal{M}_\beta^t| \tag{11}$$

for some constant $c > 0$ independent of N, t, ℓ.

Definition A.7 (KV Drift)**.** The KV drift at layer ℓ, step t for token i is:

$$\Delta_i^{t,\ell} := \left\|\mathbf{K}_i^{t,\ell} - \mathbf{K}_i^{t-1,\ell}\right\|_2 + \left\|\mathbf{V}_i^{t,\ell} - \mathbf{V}_i^{t-1,\ell}\right\|_2 \tag{12}$$

Average drift over decoded tokens: $\Delta^{t,\ell} := \frac{1}{|\mathcal{D}^{<t}|}\sum_{i\in\mathcal{D}^{<t}} \Delta_i^{t,\ell}$

A.3 LAYER-WISE KV DRIFT MONOTONICITY

This theorem formalizes the observation that KV drift increases with layer depth, providing theoretical justification for our layer-aware cache refresh strategy that selectively recomputes deeper layers while reusing shallow-layer caches. Figure 1a empirically validates this monotonicity property.

Theorem A.8 (Layer-Wise KV Drift Monotonicity)**.** *Under Assumptions A.2–A.5, there exists a transition layer* $\ell^* \in \{1, \ldots, L\}$ *such that for sufficiently large* t *(when most tokens are decoded):*

$$\mathbb{E}_t[\Delta^{t,\ell}] \leq \mathbb{E}_t[\Delta^{t,\ell'}], \quad \forall \ell \leq \ell^* < \ell' \leq L \tag{13}$$

Proof. **Step 1: Relating KV Drift to Hidden State Drift.**

The key-value projections at layer ℓ are:

$$\mathbf{K}_i^{t,\ell} = W_K^\ell \mathbf{H}_i^{t,\ell} \tag{14}$$

$$\mathbf{V}_i^{t,\ell} = W_V^\ell \mathbf{H}_i^{t,\ell} \tag{15}$$

By the triangle inequality and Assumption A.3 ($\|W_K^\ell\|_2, \|W_V^\ell\|_2 \leq W_{\max}$):

$$\begin{aligned}\|\mathbf{K}_i^{t,\ell} - \mathbf{K}_i^{t-1,\ell}\|_2 &= \|W_K^\ell(\mathbf{H}_i^{t,\ell} - \mathbf{H}_i^{t-1,\ell})\|_2 \\ &\leq \|W_K^\ell\|_2 \|\mathbf{H}_i^{t,\ell} - \mathbf{H}_i^{t-1,\ell}\|_2 \\ &\leq W_{\max}\|\Delta\mathbf{H}_i^{t,\ell}\|_2 \end{aligned} \tag{16}$$

Similarly for values:

$$\|\mathbf{V}_i^{t,\ell} - \mathbf{V}_i^{t-1,\ell}\|_2 \leq W_{\max}\|\Delta\mathbf{H}_i^{t,\ell}\|_2 \tag{17}$$

Therefore:

$$\Delta_i^{t,\ell} = \|\mathbf{K}_i^{t,\ell} - \mathbf{K}_i^{t-1,\ell}\|_2 + \|\mathbf{V}_i^{t,\ell} - \mathbf{V}_i^{t-1,\ell}\|_2 \leq 2W_{\max}\|\Delta\mathbf{H}_i^{t,\ell}\|_2 \tag{18}$$

Step 2: Layer Recursion for Hidden States.

At layer ℓ, the transformer block computes:

$$\mathbf{H}_i^{t,\ell+1} = \mathbf{H}_i^{t,\ell} + \text{Attn}^\ell(\mathbf{Q}_i^{t,\ell}, \mathbf{K}^{t,\ell}, \mathbf{V}^{t,\ell}) + \text{FFN}^\ell(\mathbf{H}_i^{t,\ell} + \text{Attn}^\ell(\cdot)) \tag{19}$$

where the attention output is:

$$\text{Attn}^\ell(\mathbf{Q}_i^{t,\ell}, \mathbf{K}^{t,\ell}, \mathbf{V}^{t,\ell}) = \sum_{j=1}^N \mathbf{S}_{i,j}^{t,\ell}\mathbf{V}_j^{t,\ell} \tag{20}$$

The change in hidden state at layer $\ell + 1$ satisfies:

$$\begin{aligned}\|\Delta\mathbf{H}_i^{t,\ell+1}\|_2 &= \|\mathbf{H}_i^{t,\ell+1} - \mathbf{H}_i^{t-1,\ell+1}\|_2 \\ &\leq \|\Delta\mathbf{H}_i^{t,\ell}\|_2 + \|\text{Attn}^\ell(t) - \text{Attn}^\ell(t-1)\|_2 \\ &\quad + \|\text{FFN}^\ell(\text{input}^t) - \text{FFN}^\ell(\text{input}^{t-1})\|_2 \end{aligned} \tag{21}$$

By Assumption A.3, the FFN is L_{FFN}-Lipschitz:

$$\|\text{FFN}^\ell(\text{input}^t) - \text{FFN}^\ell(\text{input}^{t-1})\|_2 \leq L_{\text{FFN}}\|\text{input}^t - \text{input}^{t-1}\|_2 \tag{22}$$

The FFN input is $\mathbf{H}_i^{t,\ell} + \text{Attn}^\ell(\cdot)$, so:

$$\|\text{input}^t - \text{input}^{t-1}\|_2 \leq \|\Delta\mathbf{H}_i^{t,\ell}\|_2 + \|\text{Attn}^\ell(t) - \text{Attn}^\ell(t-1)\|_2 \tag{23}$$

Therefore:

$$\|\Delta\mathbf{H}_i^{t,\ell+1}\|_2 \leq (1 + L_{\text{FFN}})\|\Delta\mathbf{H}_i^{t,\ell}\|_2 + (1 + L_{\text{FFN}})\|\text{Attn}^\ell(t) - \text{Attn}^\ell(t-1)\|_2 \tag{24}$$

Step 3: Bounding Attention Output Change.

Denote $\Delta_{\text{attn}}^{t,\ell,i} := \|\text{Attn}^\ell(t) - \text{Attn}^\ell(t-1)\|_2$. We decompose:

$$\begin{aligned}
&\sum_{j=1}^{N} \mathbf{S}_{i,j}^{t,\ell}\mathbf{V}_j^{t,\ell} - \sum_{j=1}^{N} \mathbf{S}_{i,j}^{t-1,\ell}\mathbf{V}_j^{t-1,\ell} \\
&= \sum_{j=1}^{N} \mathbf{S}_{i,j}^{t,\ell}(\mathbf{V}_j^{t,\ell} - \mathbf{V}_j^{t-1,\ell}) + \sum_{j=1}^{N}(\mathbf{S}_{i,j}^{t,\ell} - \mathbf{S}_{i,j}^{t-1,\ell})\mathbf{V}_j^{t-1,\ell}
\end{aligned} \tag{25}$$

Taking norms and applying triangle inequality:

$$\Delta_{\text{attn}}^{t,\ell,i} \leq \sum_{j=1}^{N} \mathbf{S}_{i,j}^{t,\ell}\|\mathbf{V}_j^{t,\ell} - \mathbf{V}_j^{t-1,\ell}\|_2 + \sum_{j=1}^{N} |\mathbf{S}_{i,j}^{t,\ell} - \mathbf{S}_{i,j}^{t-1,\ell}|\|\mathbf{V}_j^{t-1,\ell}\|_2 \tag{26}$$

Step 3a: First term (value changes). Since $\sum_j \mathbf{S}_{i,j}^{t,\ell} = 1$ (attention weights sum to 1):

$$\begin{aligned}
\sum_{j=1}^{N} \mathbf{S}_{i,j}^{t,\ell}\|\mathbf{V}_j^{t,\ell} - \mathbf{V}_j^{t-1,\ell}\|_2 &\leq \sum_{j=1}^{N} \mathbf{S}_{i,j}^{t,\ell} W_{\max}\|\Delta\mathbf{H}_j^{t,\ell}\|_2 \quad \text{(by Assumption A.3)} \\
&= W_{\max}\mathbb{E}_{j\sim\mathbf{S}_{i,:}^{t,\ell}}[\|\Delta\mathbf{H}_j^{t,\ell}\|_2] \\
&\leq W_{\max}\bar{\Delta}^{t,\ell}
\end{aligned} \tag{27}$$

Step 3b: Second term (attention weight changes). By Cauchy-Schwarz: $\sum_j |a_j|b_j \leq (\sum_j |a_j|)\max_j b_j$

By Assumption A.2: $\|\mathbf{V}_j^{t-1,\ell}\|_2 \leq W_{\max}R_\ell$

Therefore:

$$\sum_{j=1}^{N} |\mathbf{S}_{i,j}^{t,\ell} - \mathbf{S}_{i,j}^{t-1,\ell}|\|\mathbf{V}_j^{t-1,\ell}\|_2 \leq W_{\max}R_\ell \sum_{j=1}^{N} |\mathbf{S}_{i,j}^{t,\ell} - \mathbf{S}_{i,j}^{t-1,\ell}| \tag{28}$$

By the inequality $\|\mathbf{v}\|_1 \leq \sqrt{n}\|\mathbf{v}\|_2$:

$$\sum_{j=1}^{N} |\mathbf{S}_{i,j}^{t,\ell} - \mathbf{S}_{i,j}^{t-1,\ell}| \leq \sqrt{N}\|\mathbf{S}_{i,:}^{t,\ell} - \mathbf{S}_{i,:}^{t-1,\ell}\|_2 \tag{29}$$

By Lemma A.1 (softmax is 1-Lipschitz in ℓ_2):

$$\|\mathbf{S}_{i,:}^{t,\ell} - \mathbf{S}_{i,:}^{t-1,\ell}\|_2 \leq \|\mathbf{z}_i^{t,\ell} - \mathbf{z}_i^{t-1,\ell}\|_2 \tag{30}$$

where $\mathbf{z}_i^{t,\ell} = (z_{i,1}^{t,\ell}, \ldots, z_{i,N}^{t,\ell})$ with $z_{i,j}^{t,\ell} = \frac{1}{\sqrt{d_k}}\mathbf{Q}_i^{t,\ell} \cdot \mathbf{K}_j^{t,\ell}$.

Step 3c: Bounding logit changes. For each component:

$$\begin{aligned} z_{i,j}^{t,\ell} - z_{i,j}^{t-1,\ell} &= \frac{1}{\sqrt{d_k}}[\mathbf{Q}_i^{t,\ell} \cdot \mathbf{K}_j^{t,\ell} - \mathbf{Q}_i^{t-1,\ell} \cdot \mathbf{K}_j^{t-1,\ell}] \\ &= \frac{1}{\sqrt{d_k}}[\mathbf{Q}_i^{t,\ell} \cdot (\mathbf{K}_j^{t,\ell} - \mathbf{K}_j^{t-1,\ell}) + (\mathbf{Q}_i^{t,\ell} - \mathbf{Q}_i^{t-1,\ell}) \cdot \mathbf{K}_j^{t-1,\ell}] \end{aligned} \tag{31}$$

By Cauchy-Schwarz and the bounds from Assumptions A.2–A.3:

$$\begin{aligned} |z_{i,j}^{t,\ell} - z_{i,j}^{t-1,\ell}| &\leq \frac{1}{\sqrt{d_k}}[W_{\max} R_\ell \cdot W_{\max} \|\Delta \mathbf{H}_j^{t,\ell}\|_2 + W_{\max} \|\Delta \mathbf{H}_i^{t,\ell}\|_2 \cdot W_{\max} R_\ell] \\ &= \frac{W_{\max}^2 R_\ell}{\sqrt{d_k}}[\|\Delta \mathbf{H}_i^{t,\ell}\|_2 + \|\Delta \mathbf{H}_j^{t,\ell}\|_2] \\ &\leq \frac{2W_{\max}^2 R_\ell}{\sqrt{d_k}} \max_k \|\Delta \mathbf{H}_k^{t,\ell}\|_2 \end{aligned} \tag{32}$$

Taking ℓ_2 norm of the logit vector:

$$\begin{aligned} \|\mathbf{z}_i^{t,\ell} - \mathbf{z}_i^{t-1,\ell}\|_2^2 &= \sum_{j=1}^{N} |z_{i,j}^{t,\ell} - z_{i,j}^{t-1,\ell}|^2 \\ &\leq N \left(\frac{2W_{\max}^2 R_\ell}{\sqrt{d_k}} \right)^2 (\max_k \|\Delta \mathbf{H}_k^{t,\ell}\|_2)^2 \end{aligned} \tag{33}$$

Therefore:

$$\|\mathbf{z}_i^{t,\ell} - \mathbf{z}_i^{t-1,\ell}\|_2 \leq \frac{2W_{\max}^2 R_\ell \sqrt{N}}{\sqrt{d_k}} \max_k \|\Delta \mathbf{H}_k^{t,\ell}\|_2 \tag{34}$$

For typical sequences where $\max_k \|\Delta \mathbf{H}_k^{t,\ell}\|_2 = O(\bar{\Delta}^{t,\ell})$:

$$\|\mathbf{z}_i^{t,\ell} - \mathbf{z}_i^{t-1,\ell}\|_2 \leq \frac{2W_{\max}^2 R_\ell \sqrt{N}}{\sqrt{d_k}} \bar{\Delta}^{t,\ell} \tag{35}$$

Step 3d: Combining. Combining the bounds from Steps 3a-3c:

$$\begin{aligned} \Delta_{\text{attn}}^{t,\ell,i} &\leq W_{\max} \bar{\Delta}^{t,\ell} + W_{\max} R_\ell \sqrt{N} \cdot \frac{2W_{\max}^2 R_\ell \sqrt{N}}{\sqrt{d_k}} \bar{\Delta}^{t,\ell} \\ &= W_{\max} \bar{\Delta}^{t,\ell} \left(1 + \frac{2W_{\max}^2 R_\ell^2 N}{\sqrt{d_k}} \right) \end{aligned} \tag{36}$$

Define:

$$C_{\text{attn}}(\ell) := \frac{2W_{\max}^2 R_\ell^2 N}{\sqrt{d_k}} = O\left(\frac{W_{\max}^2 R_\ell^2 N}{\sqrt{d_k}} \right) \tag{37}$$

Then:

$$\Delta_{\text{attn}}^{t,\ell,i} \leq W_{\max}(1 + C_{\text{attn}}(\ell)) \bar{\Delta}^{t,\ell} \tag{38}$$

Step 4: Recursive Bound on Hidden State Drift.

Substituting equation 38 into equation 24:

$$\|\Delta \mathbf{H}_i^{t,\ell+1}\|_2 \leq (1 + L_{\text{FFN}}) \|\Delta \mathbf{H}_i^{t,\ell}\|_2 + (1 + L_{\text{FFN}}) W_{\max}(1 + C_{\text{attn}}(\ell)) \bar{\Delta}^{t,\ell} \tag{39}$$

Taking averages over all tokens:

$$\bar{\Delta}^{t,\ell+1} \leq [(1 + L_{\text{FFN}}) + (1 + L_{\text{FFN}}) W_{\max}(1 + C_{\text{attn}}(\ell))] \bar{\Delta}^{t,\ell} \tag{40}$$

Define the layer-dependent amplification factor:

$$\lambda_\ell := (1 + L_{\text{FFN}})[1 + W_{\max}(1 + C_{\text{attn}}(\ell))] \tag{41}$$

Then:

$$\bar{\Delta}^{t,\ell+1} \le \lambda_\ell \bar{\Delta}^{t,\ell} \tag{42}$$

Step 5: Layer-wise Accumulation by Induction.

By induction on ℓ:

$$\bar{\Delta}^{t,\ell} \le \bar{\Delta}^{t,1} \prod_{k=1}^{\ell-1} \lambda_k \tag{43}$$

Since $\lambda_\ell > 1$, drift accumulates multiplicatively across layers.

Step 6: Applying Layer-Wise Specialization.

By Assumption A.5:

- **Shallow layers** ($\ell \le \ell^*$): $\bar{\Delta}^{t,\ell} \le f_\ell(t) \to 0$ as $t \to T$
- **Deep layers** ($\ell > \ell^*$): $\liminf_{t\to T} \bar{\Delta}^{t,\ell} \ge c_\ell > 0$

By equation 18:

$$\mathbb{E}[\Delta^{t,\ell}] = \mathbb{E}\left[\frac{1}{|\mathcal{D}^{<t}|} \sum_{i\in\mathcal{D}^{<t}} \Delta_i^{t,\ell}\right] \le 2W_{\max}\bar{\Delta}^{t,\ell} \tag{44}$$

Therefore, for sufficiently large t and any $\ell \le \ell^* < \ell'$:

$$\mathbb{E}[\Delta^{t,\ell}] \le 2W_{\max} f_\ell(t) \to 0 \tag{45}$$

$$\mathbb{E}[\Delta^{t,\ell'}] \ge 2W_{\max} c_{\ell'} > 0 \tag{46}$$

This establishes:

$$\mathbb{E}[\Delta^{t,\ell}] < \mathbb{E}[\Delta^{t,\ell'}], \quad \forall \ell \le \ell^* < \ell' \tag{47}$$

□

A.4 ATTENTION CONCENTRATION AND DRIFT

Theorem A.9 (Attention Concentration and Drift)**.** *Let* $\mathcal{T}^{t,\ell} = \arg\max_{k\in\mathcal{D}^{<t}} \sum_{q\in\mathcal{M}_\beta^t} \mathbf{S}_{q,k}^{t,\ell}$ *be the most-attended token at layer* ℓ*, step* t*. Under Assumptions A.2–A.3, the most-attended token has drift bounded by:*

$$\Delta_{\mathcal{T}^{t,\ell}}^{t,\ell} \le \bar{\Delta}^{t,\ell} + \epsilon_t \tag{48}$$

where $\bar{\Delta}^{t,\ell} = \frac{1}{|\mathcal{D}^{<t}|}\sum_{i\in\mathcal{D}^{<t}} \Delta_i^{t,\ell}$ *is the average drift and* $\epsilon_t = O\left(\frac{\sqrt{d_k}}{R_\ell\sqrt{N}}\right)$.

Proof. **Step 1: Bounding Attention Weight Changes.**

We derive how attention weights $\mathbf{S}_{q,k}^{t,\ell}$ change when hidden states change.

Step 1a: Logit change. The attention logits are $z_{q,k} = \frac{1}{\sqrt{d_k}}\mathbf{Q}_q \cdot \mathbf{K}_k$ where:

$$\mathbf{Q}_q = W_Q \mathbf{H}_q, \quad \mathbf{K}_k = W_K \mathbf{H}_k \tag{49}$$

The change in logits between steps t and $t-1$ is:

$$z_{q,k}^{t,\ell} - z_{q,k}^{t-1,\ell} = \frac{1}{\sqrt{d_k}}[\mathbf{Q}_q^{t,\ell} \cdot \mathbf{K}_k^{t,\ell} - \mathbf{Q}_q^{t-1,\ell} \cdot \mathbf{K}_k^{t-1,\ell}] \tag{50}$$

Using the identity $ab - a'b' = a(b - b') + (a - a')b'$:

$$= \frac{1}{\sqrt{d_k}}[\mathbf{Q}_q^{t,\ell} \cdot (\mathbf{K}_k^{t,\ell} - \mathbf{K}_k^{t-1,\ell}) + (\mathbf{Q}_q^{t,\ell} - \mathbf{Q}_q^{t-1,\ell}) \cdot \mathbf{K}_k^{t-1,\ell}] \tag{51}$$

Step 1b: Apply Cauchy-Schwarz inequality. Taking absolute value and applying Cauchy-Schwarz:

$$\begin{aligned} |z_{q,k}^{t,\ell} - z_{q,k}^{t-1,\ell}| \le \frac{1}{\sqrt{d_k}}[&\|\mathbf{Q}_q^{t,\ell}\|_2 \|\mathbf{K}_k^{t,\ell} - \mathbf{K}_k^{t-1,\ell}\|_2 \\ &+ \|\mathbf{Q}_q^{t,\ell} - \mathbf{Q}_q^{t-1,\ell}\|_2 \|\mathbf{K}_k^{t-1,\ell}\|_2] \end{aligned} \tag{52}$$

Step 1c: Bound projection norms. By Assumption A.2: $\|\mathbf{H}_i^{t,\ell}\|_2 \le R_\ell$ for all i, t.

By Assumption A.3: $\|W_Q\|_2, \|W_K\|_2 \le W_{\max}$.

Therefore:

$$\|\mathbf{Q}_q^{t,\ell}\|_2 \le \|W_Q\|_2 \|\mathbf{H}_q^{t,\ell}\|_2 \le W_{\max} R_\ell \tag{53}$$

$$\|\mathbf{K}_k^{t,\ell}\|_2 \le \|W_K\|_2 \|\mathbf{H}_k^{t,\ell}\|_2 \le W_{\max} R_\ell \tag{54}$$

$$\|\mathbf{K}_k^{t,\ell} - \mathbf{K}_k^{t-1,\ell}\|_2 \le \|W_K\|_2 \|\mathbf{H}_k^{t,\ell} - \mathbf{H}_k^{t-1,\ell}\|_2 \le W_{\max} \|\Delta \mathbf{H}_k\|_2 \tag{55}$$

$$\|\mathbf{Q}_q^{t,\ell} - \mathbf{Q}_q^{t-1,\ell}\|_2 \le W_{\max} \|\Delta \mathbf{H}_q\|_2 \tag{56}$$

Substituting these bounds:

$$\begin{aligned} |z_{q,k}^{t,\ell} - z_{q,k}^{t-1,\ell}| &\le \frac{1}{\sqrt{d_k}}[W_{\max} R_\ell \cdot W_{\max} \|\Delta \mathbf{H}_k\|_2 + W_{\max} \|\Delta \mathbf{H}_q\|_2 \cdot W_{\max} R_\ell] \\ &= \frac{W_{\max}^2 R_\ell}{\sqrt{d_k}}[\|\Delta \mathbf{H}_k\|_2 + \|\Delta \mathbf{H}_q\|_2] \end{aligned} \tag{57}$$

Step 1d: Use maximum drift. Since $\|\Delta \mathbf{H}_i\|_2 \le \Delta_{\max}$ for all i:

$$|z_{q,k}^{t,\ell} - z_{q,k}^{t-1,\ell}| \le \frac{2W_{\max}^2 R_\ell}{\sqrt{d_k}} \Delta_{\max} \tag{58}$$

Step 1e: Compute ℓ_2 norm of logit vector. The logit vector for query q is $\mathbf{z}_q = (z_{q,1}, \dots, z_{q,N}) \in \mathbb{R}^N$.

By the previous bound applied to each component:

$$\begin{aligned} \|\mathbf{z}_q^{t,\ell} - \mathbf{z}_q^{t-1,\ell}\|_2^2 &= \sum_{k=1}^{N} |z_{q,k}^{t,\ell} - z_{q,k}^{t-1,\ell}|^2 \\ &\le \sum_{k=1}^{N} \left(\frac{2W_{\max}^2 R_\ell}{\sqrt{d_k}}\right)^2 \Delta_{\max}^2 \\ &= N \cdot \frac{4W_{\max}^4 R_\ell^2}{d_k} \Delta_{\max}^2 \end{aligned} \tag{59}$$

Taking square root:

$$\|\mathbf{z}_q^{t,\ell} - \mathbf{z}_q^{t-1,\ell}\|_2 \le \frac{2W_{\max}^2 R_\ell \sqrt{N}}{\sqrt{d_k}} \Delta_{\max} \tag{60}$$

Step 1f: Apply softmax Lipschitz property. By Lemma A.1 (softmax is 1-Lipschitz in ℓ_2 norm):

$$\|\mathbf{S}_{q,:}^{t,\ell} - \mathbf{S}_{q,:}^{t-1,\ell}\|_2 \le \|\mathbf{z}_q^{t,\ell} - \mathbf{z}_q^{t-1,\ell}\|_2 \le \frac{2W_{\max}^2 R_\ell \sqrt{N}}{\sqrt{d_k}} \Delta_{\max} \tag{61}$$

Step 1g: Convert to ℓ_∞ norm. Since $\|\mathbf{v}\|_\infty \le \|\mathbf{v}\|_2$ for any vector $\mathbf{v}$:

$$\max_k |\mathbf{S}_{q,k}^{t,\ell} - \mathbf{S}_{q,k}^{t-1,\ell}| \le \frac{2W_{\max}^2 R_\ell \sqrt{N}}{\sqrt{d_k}} \Delta_{\max} \tag{62}$$

Step 2: Change in Total Attention Received.

For token k, the change in total attention received is:

$$\begin{aligned}|\alpha_k^{t,\ell} - \alpha_k^{t-1,\ell}| &= \left|\sum_{q \in \mathcal{M}_\beta^t} (\mathbf{S}_{q,k}^{t,\ell} - \mathbf{S}_{q,k}^{t-1,\ell})\right| \\ &\leq \sum_{q \in \mathcal{M}_\beta^t} |\mathbf{S}_{q,k}^{t,\ell} - \mathbf{S}_{q,k}^{t-1,\ell}| \quad \text{(triangle inequality)} \\ &\leq |\mathcal{M}_\beta^t| \cdot \max_q \max_k |\mathbf{S}_{q,k}^{t,\ell} - \mathbf{S}_{q,k}^{t-1,\ell}| \quad \text{(bound by max)} \end{aligned} \tag{63}$$

Using equation 62:

$$|\alpha_k^{t,\ell} - \alpha_k^{t-1,\ell}| \leq |\mathcal{M}_\beta^t| \cdot \frac{2W_{\max}^2 R_\ell \sqrt{N}}{\sqrt{d_k}} \Delta_{\max} \tag{64}$$

Step 3: Relating to KV Drift.

Recall that KV drift is $\Delta_i^{t,\ell} = \|\mathbf{K}_i^{t,\ell} - \mathbf{K}_i^{t-1,\ell}\|_2 + \|\mathbf{V}_i^{t,\ell} - \mathbf{V}_i^{t-1,\ell}\|_2$.

By Assumption A.3:

$$\Delta_i^{t,\ell} \leq W_{\max}\|\Delta\mathbf{H}_i\|_2 + W_{\max}\|\Delta\mathbf{H}_i\|_2 = 2W_{\max}\|\Delta\mathbf{H}_i\|_2 \tag{65}$$

Therefore: $\|\Delta\mathbf{H}_i\|_2 \geq \frac{\Delta_i^{t,\ell}}{2W_{\max}}$.

In particular: $\Delta_{\max} \geq \frac{\max_i \Delta_i^{t,\ell}}{2W_{\max}}$.

Substituting into equation 64:

$$|\alpha_k^{t,\ell} - \alpha_k^{t-1,\ell}| \leq |\mathcal{M}_\beta^t| \cdot \frac{2W_{\max}^2 R_\ell \sqrt{N}}{\sqrt{d_k}} \cdot \frac{\max_i \Delta_i^{t,\ell}}{2W_{\max}} = |\mathcal{M}_\beta^t| \cdot \frac{W_{\max} R_\ell \sqrt{N}}{\sqrt{d_k}} \max_i \Delta_i^{t,\ell} \tag{66}$$

Step 4: Stability Constraint and Excess Drift.

Suppose $\mathcal{T}^{t,\ell}$ has drift $\Delta_{\mathcal{T}^{t,\ell}}^{t,\ell} = \bar{\Delta}^{t,\ell} + \varepsilon$ where $\varepsilon > 0$ is excess drift beyond average.

Then:

$$|\alpha_{\mathcal{T}^{t,\ell}}^{t,\ell} - \alpha_{\mathcal{T}^{t,\ell}}^{t-1,\ell}| \leq |\mathcal{M}_\beta^t| \cdot \frac{W_{\max} R_\ell \sqrt{N}}{\sqrt{d_k}} (\bar{\Delta}^{t,\ell} + \varepsilon) \tag{67}$$

While tokens with average drift have:

$$|\alpha_k^{t,\ell} - \alpha_k^{t-1,\ell}| \leq |\mathcal{M}_\beta^t| \cdot \frac{W_{\max} R_\ell \sqrt{N}}{\sqrt{d_k}} \bar{\Delta}^{t,\ell} \tag{68}$$

The differential attention change is:

$$\Delta_{\text{differential}} = |\mathcal{M}_\beta^t| \cdot \frac{W_{\max} R_\ell \sqrt{N}}{\sqrt{d_k}} \varepsilon \tag{69}$$

For $\mathcal{T}^{t,\ell}$ to remain most-attended, the gap at step $t-1$ must absorb this differential:

$$\Gamma^{t-1,\ell} \geq \Delta_{\text{differential}} = |\mathcal{M}_\beta^t| \cdot \frac{W_{\max} R_\ell \sqrt{N}}{\sqrt{d_k}} \varepsilon \tag{70}$$

Step 5: Assuming Bounded Attention Gap.

Applying the assumption A.6:

$$c \cdot |\mathcal{M}_\beta^t| \geq |\mathcal{M}_\beta^t| \cdot \frac{W_{\max} R_\ell \sqrt{N}}{\sqrt{d_k}} \varepsilon \tag{71}$$

Canceling $|\mathcal{M}_\beta^t|$ (assuming $|\mathcal{M}_\beta^t| > 0$):

$$c \geq \frac{W_{\max} R_\ell \sqrt{N}}{\sqrt{d_k}} \varepsilon \tag{72}$$

Solving for ε:

$$\varepsilon \leq \frac{c\sqrt{d_k}}{W_{\max} R_\ell \sqrt{N}} = O\left(\frac{\sqrt{d_k}}{R_\ell \sqrt{N}}\right) \tag{73}$$

Therefore:

$$\Delta_{\mathcal{T}^{t,\ell}}^{t,\ell} \leq \bar{\Delta}^{t,\ell} + O\left(\frac{\sqrt{d_k}}{R_\ell \sqrt{N}}\right) \tag{74}$$

□

A.5 IMPLICATIONS FOR ELASTIC-CACHE

These results provide theoretical justification for our design:

- **Theorem A.8**: Deeper layers have larger KV drift, justifying layer-aware refresh starting from ℓ^*
- **Theorem A.9**: Most-attended tokens have minimal drift, validating their use as cache staleness indicators

B DETAILED EXPERIMENT SETUP

Implementation Details. We conduct all the experiments on a single NVIDIA A100 80GB GPU to ensure a consistent hardware environment. We evaluate our proposed method, **`Elastic-Cache`**, on three large scale DLMs: LLaDA-Instruct (Nie et al., 2025a), LLaDA-1.5 (Zhu et al., 2025), and the multimodal LLaDA-V (You et al., 2025). Our evaluation spans both language and multimodal reasoning tasks including MBPP (Austin et al., 2021b), HumanEval (Chen et al., 2021) for coding tasks, MATH (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021) for Maths related tasks and MathVista (Lu et al., 2023) MathVerse (Zhang et al., 2024b) for multimodal mathematical reasoning tasks. The major hyperparameters for Elastic-Cache, unless otherwise specified in ablation studies, are set to a attention threshold of $\gamma = 0.9$, a confidence threshold for parallel decoding of $\epsilon = 0.9$, and a cache block size of 32. To establish a rigorous and fair comparison for all baseline methods, were re-evaluate all the methods including the original diffusion model LLaDA Nie et al. (2025a) and Fast-dLLM (Wu et al., 2025). This process eliminates confounding variables from hardware or software discrepancies and ensures that all observed performance differences are attributable to the methods themselves.

Evaluation Framework and Metrics. Our evaluation protocol comprehensively assesses both inference efficiency and the preservation of model performance across a variety of tasks. For standardization and reproducibility, we conduct all task-specific evaluations using the `lm-eval-harness` library (Gao et al., 2024). We measure inference speed by throughput in tokens per second (t/s), which we calculate as the average number of tokens the model generates over the entire sequence until it produces an end-of-sequence (**`<eos>`**) token. We keep our calculation methodology consistent with that of Fast-dLLM (Wu et al., 2025) to ensure comparable speed benchmarks. We measure task-specific performance using established metrics appropriate for each benchmark: for GSM8K (Cobbe et al., 2021), we report 5-shot `flexible_extract` exact match accuracy; for the MATH dataset (Hendrycks et al., 2021), we report the 4-shot `math_verify` score using the `minerva_math` variant; for HumanEval (Chen et al., 2021), we evaluate 0-shot accuracy using a post-processing script consistent with the Fast-dLLM implementation to ensure fair comparison; and for MBPP (Austin et al., 2021b), we report the 3-shot `pass@1` metric. For multimodal evaluation on LLaDA-V (You et al., 2025), we utilize an evaluation suite adapted from its official implementation using the `lmms-eval` framework (Zhang et al., 2024a; Li et al., 2024) to test on the MathVista (Lu et al., 2023) and MathVerse (Zhang et al., 2024b) benchmarks. For MathVista, we report the `gpt_eval_score`, and for MathVerse, we report the `gpt_eval_score` on the `mathverse_testmini_vision_dominant` subset.

Hyper-parameters: The hyper-parameters used for Elastic-Cache are provided in Table 6. Specifically,

- For LLaDA and LLaDA-1.5, γ = 0.9 everywhere; β is mostly 16, except GSM8K (β = 32 at 256, 16 at 512) and HumanEval (β = 32 at both 256/512).
- For LLaDA-V (MathVista/MathVerse), γ = 0.7 and β = 16 for both 256 and 512 token lengths.
- All tasks are reported at generation lengths 256 and 512.

C BATCH IMPLEMENTATION OF ELASTIC-CACHE

In our experimental setup, we use a batch size of one for comparison purposes. However, in a real-world deployment scenario, requests can be grouped into batches to take advantage of parallelism and speed up decoding. In this section, we extend our implementation of Elastic-Cache to support batch decoding and compare it to baselines under different batch sizes.

Batch implementation of Elastic-Cache is very challenging because the query length varies during caching and cache updates. Specifically, the query positions at time t are represented as $\mathcal{Q}^t$, while caching is in progress, and $\mathcal{I}$ when cache updates are being performed. Since Elastic-Cache automatically triggers cache updates, each sample within a batch will have a different length due to the varying triggers for each sample. This poses a significant challenge to parallelism and efficiency of the method.

To address this problem, we propose a solution that involves rearranging the batch and concatenating all sentences within the batch into a single sentence. This approach enables us to compute the sentences in parallel, regardless of their current lengths. We then reimplement the multi-head attention function to compute attention on the concatenated sentence (Algorithm 2).

Algorithm 2 Batch attention computation

```
Require: Batch samples $\mathbf{Q}^{i,t,l}_{\mathcal{Q}^{i,t}}, \tilde{\mathbf{K}}^{i,t,l}_{\mathcal{I}}, \tilde{\mathbf{V}}^{i,t,l}_{\mathcal{I}}$;    Batch size $B$;
$\mathbf{Q}^{t,l} \leftarrow \texttt{Cat}([\mathbf{Q}^{1,t,l}_{\mathcal{Q}^{1,t}}, \dots, \mathbf{Q}^{B,t,l}_{\mathcal{Q}^{B,t}}])$
for $i = 1, \dots, B$ do
    $\mathbf{K}^{t,l} \leftarrow \texttt{Cat}([\mathbf{K}^{t,l}, \{\mathbf{K}^{i,t,l}_{\mathcal{I}}\} \times |\mathcal{Q}^{i,t}|]);$
    $\mathbf{V}^{t,l} \leftarrow \texttt{Cat}([\mathbf{V}^{t,l}, \{\mathbf{V}^{i,t,l}_{\mathcal{I}}\} \times |\mathcal{Q}^{i,t}|]);$
end for
$\mathbf{A}^{t,l} = \text{softmax}\left(\frac{\mathbf{Q}^{t,l}(\mathbf{K}^{t,l})^\top}{\sqrt{d_k}}\right)\mathbf{V}^{t,l}$
return $\mathbf{A}^{t,l}$
```

D USE OF LARGE LANGUAGE MODELS

An LLM was used to help solely polish the writing of the paper, while all method, idea and experiments were prepared and carried out entirely by the authors.

E SAMPLE RESPONSE

In the following, we present several examples of actual generation results produced by our approach under different parameter settings.

Table 6: The hyper-parameters of Elastic-Cache under various settings.

Model	Benchmark	Gen Length	β	γ
LLaDA	GSM8K (5-shot)	256	32	0.9
		512	16	0.9
	MATH (4-shot)	256	16	0.9
		512	16	0.9
	Humaneval (0-shot)	256	32	0.9
		512	32	0.9
	MBPP (3-shot)	256	16	0.9
		512	16	0.9
LLaDA-1.5	GSM8K (5-shot)	256	16	0.9
		512	16	0.9
	MATH (4-shot)	256	16	0.9
		512	16	0.9
	Humaneval (0-shot)	256	32	0.9
		512	32	0.9
	MBPP (3-shot)	256	16	0.9
		512	16	0.9
LLaDA-V	Mathvista	256	16	0.7
		512	16	0.7
	Mathverse	256	16	0.7
		512	16	0.7

Table 7: Example – Qualitative comparison for different γ, LLaDA-1.5

Prompt: *A store sells 3 types of fruit: apples at \$2 each, bananas at \$1 each, and oranges at \$3 each. If Sarah buys 4 apples, 6 bananas, and 2 oranges, how much does she spend in total?*

$\gamma = 0.5$	$\gamma = 0.7$	$\gamma = 0.95$
To determine how much Sarah spends in total, we need to calculate the cost of each type of fruit separately and then sum these amounts. 1. **Cost of apples:** - Sarah buys 4 apples. - Each apple costs \$2. - Total cost for apples = $(4 \times 2 = 8)$ dollars. 2. **Cost of bananas:** - Sarah buys 6 bananas. - Each banana costs \$1. - Total cost for bananas = $(6 \times 1 = 6)$ dollars. 3. **Cost of oranges:** - Sarah buys 2 oranges. - Each orange costs \$3. - Total cost for oranges = $(2 \times 3 = 6)$ dollars. Now, we sum the costs of all the fruits to find the total amount Sarah spends: $8 \text{ (apples)} + 6 \text{ (bananas)} + 6 \text{ (oranges)} = 20 \text{ dollars}$ Therefore, Sarah spends a total of 20 dollars. **Time: 3.1s**	To determine how much Sarah spends in total, we need to calculate the cost of each type of fruit separately and then sum these amounts. First, let's calculate the cost of the apples: - Sarah buys 4 apples. - Each apple costs \$2. - Therefore, the cost of apples is $(4 \times 2 = 8)$ dollars. Next, let's calculate the cost of the bananas: - Sarah buys 6 bananas. - Each banana costs \$1. - Therefore, the cost of bananas is $(6 \times 1 = 6)$ dollars. Finally, let's calculate the cost of the oranges: - Sarah buys 2 oranges. - Each orange costs \$3. - Therefore, the cost of oranges is $(2 \times 3 = 6)$ dollars. Now, we sum the costs of all the fruits to find the total amount Sarah spends: - Total cost = Cost of apples + Cost of bananas + Cost of oranges - Total cost = $(8 + 6 + 6 = 20)$ dollars So, Sarah spends a total of 20 dollars. **Time: 3.4s**	To determine how much Sarah spends in total, we need to calculate the cost of each type of fruit separately and then sum these amounts. First, let's calculate the cost of the apples: - Sarah buys 4 apples. - Each apple costs \$2. - Total cost for apples = $(4 \times 2 = 8)$ dollars. Next, let's calculate the cost of the bananas: - Sarah buys 6 bananas. - Each banana costs \$1. - Total cost for bananas = $(6 \times 1 = 6)$ dollars. Then, let's calculate the cost of the oranges: - Sarah buys 2 oranges. - Each orange costs \$3. - Total cost for oranges = $(2 \times 3 = 6)$ dollars. Now, we sum the costs of all the fruits to find the total amount spent: - Total cost = Cost of apples + Cost of bananas + Cost of oranges - Total cost = 8 + 6 + 6 - Total cost = 20 dollars Therefore, Sarah spends a total of 20 dollars. **Time: 3.5s**

Table 8: Example – Qualitative comparison for different γ, LLaDA-1.5

Prompt: *Tom has 24 marbles. He gives 1/3 of them to his sister and 1/4 of the remaining marbles to his brother. How many marbles does Tom have left?*

$\gamma = 0.5$	$\gamma = 0.7$	$\gamma = 0.95$
To determine how many marbles Tom has left, we will follow these steps: 1. Calculate the number of marbles Tom gives to his sister. 2. Subtract the marbles given to his sister from the original number of marbles. 3. Calculate the number of marbles Tom gives to his brother. 4. Subtract the marbles given to his brother from the remaining marbles. Let's start with the first step: **Step 1: Calculate the number of marbles Tom gives to his sister.** Tom gives $\frac{1}{3}$ of his 24 marbles to his sister. $\frac{1}{3} \times 24 = 8$ So, Tom gives 8 marbles to his sister. **Step 2: Subtract the marbles given to his sister from the original number of marbles.** $24 - 8 = 16$ So, after giving 8 marbles to his sister, Tom has 16 marbles left. **Step 3: Calculate the number of marbles Tom gives to his brother.** Tom gives $\frac{1}{4}$ of the remaining 16 marbles to his brother. $\frac{1}{4} \times 16 = 4$ So, Tom gives 4 marbles to his brother. **Step 4: Subtract the marbles given to his brother from the remaining marbles.** $16 - 4 = 12$ So, after giving 4 marbles to his brother, Tom has 12 marbles left. Therefore, the number of marbles Tom has left is $\boxed{12}$. **Time: 3.3s**	To determine how many marbles Tom has left, we will follow these steps: 1. Calculate the number of marbles Tom gives to his sister. 2. Subtract the marbles given to his sister from the original number of marbles. 3. Calculate the number of marbles Tom gives to his brother. 4. Subtract the marbles given to his brother from the remaining marbles. Let's start with the first step: **Step 1: Calculate the number of marbles Tom gives to his sister.** Tom gives $\frac{1}{3}$ of his 24 marbles to his sister. Marbles given to sister = $\frac{1}{3} \times 24 = 8$ **Step 2: Subtract the marbles given to his sister from the original number of marbles.** Remaining marbles after giving to sister = $24 - 8 = 16$ **Step 3: Calculate the number of marbles Tom gives to his brother.** Tom gives $\frac{1}{4}$ of the remaining 16 marbles to his brother. Marbles given to brother = $\frac{1}{4} \times 16 = 4$ **Step 4: Subtract the marbles given to his brother from the remaining marbles.** Remaining marbles after giving to brother = $16 - 4 = 12$ Therefore, the number of marbles Tom has left is $\boxed{12}$. **Time: 3.1s**	To determine how many marbles Tom has left, we will follow these steps: 1. Calculate the number of marbles Tom gives to his sister. 2. Subtract the marbles given to his sister from the total number of marbles. 3. Calculate the number of marbles Tom gives to his brother. 4. Subtract the marbles given to his brother from the remaining marbles. **Step 1: Calculate the number of marbles Tom gives to his sister.** Tom gives $\frac{1}{3}$ of his 24 marbles to his sister. Marbles given to sister = $\frac{1}{3} \times 24 = 8$ **Step 2: Subtract the marbles given to his sister from the total number of marbles.** Remaining marbles after giving to sister = $24 - 8 = 16$ **Step 3: Calculate the number of marbles Tom gives to his brother.** Tom gives $\frac{1}{4}$ of the remaining 16 marbles to his brother. Marbles given to brother = $\frac{1}{4} \times 16 = 4$ **Step 4: Subtract the marbles given to his brother from the remaining marbles.** Remaining marbles after giving to brother = 16 - 4 = 12 Therefore, the number of marbles Tom has left is $\boxed{12}$. **Time: 3.2s**

Table 9: Example – Qualitative comparison of responses under different γ.

Prompt: *Points A and B are on parabola* $y = 4x^2 + 7x - 1$, *and the origin is the midpoint of* $\overline{AB}$. *Find the square of the length of* $\overline{AB}$.

$\gamma = 0.5$	$\gamma = 0.7$	$\gamma = 0.9$
Let $A = (x_1, y_1)$ and $B = (x_2, y_2)$. Since the origin is the midpoint of $\overline{AB}$, we have: $\left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right) = (0,0).$ This implies: $x_1 + x_2 = 0 \quad \text{and} \quad y_1 + y_2 = 0.$ Since A and B lie on the parabola $y = 4x^2 + 7x - 1$, we have: $y_1+y_2 = (4x_1^2+7x_1-1)+(4x_2^2+7x_2-1$ Now, using the equations $x_1 + x_2 = 0$ and $y_1 + y_2 = 0$, we get: $4x_1^2 + 7x_1 - 1 + 4x_2^2 + 7x_2 - 1 = 0.$ This simplifies to: $4x_1^2 + 7x_1 + 4x_2^2 + 7x_2 - 2 = 0.$ Thus, the square of the length of $\overline{AB}$ is: $(x_1-x_2)^2+(y_1-y_2)^2 = 0^2+0^2 = 0.$ Therefore, the square of the length of $\overline{AB}$ is: $\boxed{0}.$ **Time: 8.1s**	Let $A = (x_1, y_1)$ and $B = (x_2, y_2)$. Since the origin is the midpoint of $\overline{AB}$, we have: $\left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right) = (0,0).$ This implies: $x_1 + x_2 = 0 \quad \text{and} \quad y_1 + y_2 = 0.$ Since A and B lie on the parabola $y = 4x^2 + 7x - 1$, we have: $y_1 = 4x_1^2 + 7x_1 - 1$ and $y_2 = 4x_2^2 + 7x_2 - 1.$ From $x_1 + x_2 = 0$, we get $x_1 = -x_2$. Substituting $x_1 = -x_2$ into the equations for y_1 and y_2: $y_1 = 4x_2^2 + 7x_2 - 1$ and $y_2 = 4x_2^2 + 7x_2 - 1.$ Since $y_1 + y_2 = 0$, we have: $y_1 = -y_2.$ Thus, $y_1 = -(4x_2^2+7x_2-1) = -4x_2^2-7x_2+1$ Similarly, $y_2 = 4x_2^2 + 7x_2 - 1.$ Therefore, $y_1 = -y_2 =$ $-(4x_2^2 + 7x_2 - 1) = -4x_2^2 - 7x_2 + 1.$ Thus, the square of the length of $\overline{AB}$ is given by: $\boxed{2}.$ **Time: 9.3s**	Let $A = (x_1, y_1)$ and $B = (x_2, y_2)$. Since the origin is the midpoint of $\overline{AB}$, we have: $\left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right) = (0,0).$ This implies: $x_1 + x_2 = 0 \quad \text{and} \quad y_1 + y_2 = 0.$ Since A and B lie on the parabola $y = 4x^2 + 7x - 1$, we have: $y_1 = 4x_1^2 + 7x_1 - 1$ and $y_2 = 4x_2^2 + 7x_2 - 1.$ Using $y_1 + y_2 = 0$, we get: $y_1 + y_2 = (4x_1^2+7x_1-1)+(4x_2^2+7x_2-1) = 0.$ Simplifying, we have: $4x_1^2 + 7x_1 + 4x_2^2 + 7x_2 - 2 = 0.$ Since $x_1 + x_2 = 0$, we get: $4x_1^2 + 7x_1 + 4x_2^2 + 7x_2 - 2 = 0.$ Now, we need to find the square of the length of $\overline{AB}$. The length of $\overline{AB}$ is given by: $AB = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$ Using the midpoint formula and the properties of the parabola, we can simplify the expression. However, since the problem asks for the square of the length, we can directly use the result. Thus, the square of the length of $\overline{AB}$ is: $\boxed{50}.$ **Time: 8.7s (Correct)**

Table 10: Coding Example – Close Elements

Prompt:

```
from typing import List

def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """␣Check␣if␣in␣given␣list␣of␣numbers,␣are␣any␣two␣numbers␣closer␣to␣each␣other␣
        than
␣␣␣␣given␣threshold.
␣␣␣␣>>>␣has_close_elements([1.0,␣2.0,␣3.0],␣0.5)
␣␣␣␣False
␣␣␣␣>>>␣has_close_elements([1.0,␣2.8,␣3.0,␣4.0,␣5.0,␣2.0],␣0.3)
␣␣␣␣True
␣␣␣␣"""
```

FAST-DLLM

```
sorted_numbers = sorted(numbers)
for i in range(len(sorted_numbers)):
    if abs(sorted_numbers[i] -
        sorted_numbers[i + 1]) <
        threshold:
         return True
return False
```

OURS

```
sorted_numbers = sorted(numbers)
for i in range(len(sorted_numbers) -
    1):
    if abs(sorted_numbers[i] -
        sorted_numbers[i + 1]) <
        threshold:
         return True
return False
```

Fast-DLLM generated incorrect solution

Table 11: Coding Example – Truncate Number

Prompt:

```
def truncate_number(number: float) -> float:
    """␣Given␣a␣positive␣floating␣point␣number,␣it␣can␣be␣decomposed␣into
␣␣␣␣and␣integer␣part␣(largest␣integer␣smaller␣than␣given␣number)␣and␣decimals
␣␣␣␣(leftover␣part␣always␣smaller␣than␣1).

␣␣␣␣Return␣the␣decimal␣part␣of␣the␣number.
␣␣␣␣>>>␣truncate_number(3.5)
␣␣␣␣0.5
␣␣␣␣"""
```

FAST-DLLM

```
if number < 0:
    raise ValueError("The␣number␣must
        ␣be␣positive.")
integer_part = int(number)
decimal_part = number - integer_part
return decimal_part
```

OURS

```
if number < 0:
    raise ValueError("The␣number␣must
        ␣be␣positive")
integer_part = int(number)
decimal_part = number - integer_part
return decimal_part
```

Both solutions are functionally equivalent with minor differences