Enabling KV Caching of Shared Prefix for Diffusion Language Models

Anonymous ACL submission

Abstract

Key-value (KV) caching for shared prefixes is essential for high-throughput large language model (LLM) serving, but it faces critical challenges in emerging diffusion language models (DLMs). In DLMs, bidirectional attention means that updating any token dynamically alters the entire context and its corresponding KVs. Thus, existing caching techniques developed for LLMs, which assume that KVs remain invariant once computed, corrupt the shared prefix KVs. Our experiments show that applying these techniques to DLMs causes model accuracy to collapse to near zero.

To unlock high-throughput DLM serving, we propose bidirectional prefix caching, BiCache, the first KV caching technique for shared prefixes in DLMs. BiCache is designed based on key observations from our comprehensive analysis: shared prefix KVs remain stable and reusable in shallow layers, while the depth of shallow layers depends on the fraction of shared prefix tokens in each request. Thus, BiCache dynamically identifies a safe layer depth for reusing shared prefix KVs and eliminates redundant computation. Evaluations demonstrate that BiCache significantly improves serving throughput by 36.3%–98.3% compared to existing techniques without accuracy collapse (only 0–1.8% difference).

1 Introduction

Key-value (KV) caching is a fundamental optimization for efficient large language model (LLM) serving. In attention layers, token hidden states are projected into key and value tensors, and storing these tensors allows serving systems to reuse them in later generation steps instead of recomputing them, thereby improving serving latency and throughput. KV caching is particularly important for shared prefixes, where the same token sequence (e.g., system prompts) appears at the beginning of multiple requests and its KVs can be cached and reused across those requests. In real-world workloads, shared prefixes account for 85–97% of prompt tokens (Srivatsa et al., 2025).

Most existing LLMs are based on autoregressive models (ARMs) that generate tokens sequentially (OpenAI, 2023). In ARMs, the KVs of previously generated tokens remain unchanged once computed. As a result, modern serving systems use "shared prefix caching," which stores the KVs of a shared prefix once and reuses them across requests (Kwon et al., 2023), making shared prefix caching a key optimization for high-throughput serving.

Recently, diffusion language models (DLMs) have emerged as a promising alternative to ARMs (Nie et al., 2025). ARMs generate tokens sequentially, making latency grow with output length (Xiao et al., 2025). Moreover, their causal attention can lead to the reversal curse, where a model may correctly answer a relation in left-to-right but fail to infer its inverse (Berglund et al., 2024; Lu et al., 2024). In contrast, DLMs generate tokens at arbitrary positions through iterative denoising steps with bidirectional attention, allowing all positions to interact at each step and alleviating the strict left-to-right generation bottleneck of ARMs.

However, the bidirectional generation process of DLMs makes shared prefix caching much harder. Under bidirectional attention, updating one token changes the KVs of all other tokens, including those in the shared prefix. Therefore, shared prefix KVs are not invariant across steps or across requests, which is a key difference from ARMs. Directly reusing the shared prefix KVs as in ARMs can corrupt the attention context and severely collapse model accuracy. Our motivating experiments show that naively applying shared prefix caching designed for ARMs to DLMs drives accuracy collapse to near zero on diverse benchmarks (§3).

Recent studies (Wu et al., 2025b,a; Liu et al., 2025d) have introduced KV caching techniques for DLMs; however, they focus on reusing KVs within

a single request and do not address shared prefix caching across requests. Because shared prefixes dominate practical workloads and are central to serving throughput, enabling shared prefix caching for DLMs remains a major open challenge.

To address this challenge, we propose bidirectional prefix caching, BICACHE, the first shared prefix caching technique for DLMs to our knowledge. The key idea of BICACHE is to selectively reuse KVs according to layer depth and the shared prefix ratio of a request. Through an in-depth analysis of how shared prefix KVs change across requests, layers, and denoising steps, we make three important observations (§4). First, shared prefix KVs remain highly similar across requests in shallow layers, making shared prefix caching possible in those layers. Second, the depth of shallow layers that can safely reuse cached KVs is strongly correlated with the shared prefix ratio, i.e., the fraction of shared prefix tokens in each request. Third, although the remaining deep layers cannot directly reuse shared prefix KVs across requests, they can still reduce redundant computation within a request by reusing KVs with periodic refresh (§5).

Based on the observations, BICACHE introduces two practical mechanisms: shared prefix profiling and layer-partitioned caching. Shared prefix profiling determines the layer depth up to which shared prefix KVs can be safely reused without harming accuracy. During serving, layer-partitioned caching directly reuses cached KVs of the shared prefix in shallow layers and applies periodic refresh in deep layers to remove redundant KV computation.

The major contributions of this study are:

- Characterize how shared prefix KVs change in DLMs across requests, layers, and steps.
- Present BICACHE, to our knowledge the first shared prefix caching technique for DLMs.
- Demonstrate significant throughput improvements by 36.3%–98.3% without accuracy collapse (only 0–1.8% differences).

2 Background

2.1 DLM

LLaDA (Nie et al., 2025) is a transformer-based DLM and a foundation for many later DLMs. Fig. 1 shows its generation process. It has three key hyperparameters (❶): the number of transformer layers L, the output length g, and the number of denoising steps s. Throughout the paper, we use the term "step" to refer to a denoising step. Over

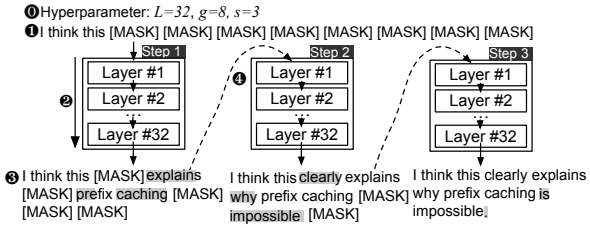


Figure 1: Workflow of token generation in DLMs.

s steps, the model generates g tokens, predicting either $\lfloor g/s \rfloor$ or $\lfloor g/s \rfloor + 1$ tokens per step (e.g., g=8, s=3 gives $\{3, 3, 2\}$ tokens per step).

When a new request arrives, LLaDA forms an input string by appending g `[MASK]` tokens to the prompt tokens (❶). At each step, the input string is processed by a forward pass through the L transformer layers (❷). In each step, DLM applies bidirectional attention and denoises the designated number of `[MASK]` tokens so that the predicted words (tokens) are filled in. In Fig. 1, step 1 denoises three tokens (as explained above), so the first step denoises "explains", "prefix", and "caching" (❸). The input string with three denoised tokens is passed to the next step, where another denoising happens using the L transformer layers. The denoised tokens from the previous step remain unchanged, and the model denoises the designated number of mask tokens, three in the example, producing "clearly", "why", and "impossible." This iterative process continues (❹) for s steps until all `[MASK]` tokens are denoised.

2.2 Key-Value and Bidirectional Attention

Here, we detail the DLM layer. We explain how hidden states are projected into KVs and how bidirectional attention aggregates them to update token representations. Finally, we explain how this bidirectional nature causes KVs to change across steps.

KV calculation. At transformer layer $\ell \in \{1, \ldots, L\}$ and step $t \in \{1, \ldots, s\}$, the hidden state of a token is denoted by $\mathbf{h}_{\ell,t} \in \mathbb{R}^d$, where d is the hidden dimension (the feature size of each token representation). The layer computes the key and value vectors using its projection matrices $W_\ell^K \in \mathbb{R}^{d_k \times d}$ and $W_\ell^V \in \mathbb{R}^{d_v \times d}$. Here, d_k and d_v denote the dimensions of the key and value vectors, respectively; the key vector ($\mathbf{k}_{\ell,t} \in \mathbb{R}^{d_k}$) is used to compute attention weights, and the value vector ($\mathbf{v}_{\ell,t} \in \mathbb{R}^{d_v}$) is aggregated according to those weights. The computation is

$$\mathbf{k}_{\ell,t} = W_\ell^K \mathbf{h}_{\ell,t}, \quad \mathbf{v}_{\ell,t} = W_\ell^V \mathbf{h}_{\ell,t}. \tag{1}$$

For conciseness, throughout the paper, we denote

the key–value pair by $\mathbf{KV}_{\ell,t} \triangleq (\mathbf{k}_{\ell,t}, \mathbf{v}_{\ell,t})$.

Bidirectional attention. Using the keys and values, layer ℓ computes an attention output $\mathbf{o}_{\ell,t}$ to gather context from the input string. Specifically, it is calculated using a query vector $\mathbf{q}_{\ell,t}$ (also projected from $\mathbf{h}_{\ell,t}$ using W_ℓ^Q) that is compared against the keys of all tokens in the input string, where n denotes the number of tokens in the input string:

$$\mathbf{o}_{\ell,t} = \sum_{j=1}^{n} \mathrm{softmax}_j \left(\frac{\mathbf{q}_{\ell,t} \cdot \mathbf{k}_{\ell,t}^{(j)}}{\sqrt{d_k}} \right) \mathbf{v}_{\ell,t}^{(j)}. \quad (2)$$

Here, $\mathbf{o}_{\ell,t}$ is a weighted sum of value vectors $\mathbf{v}_{\ell,t}^{(j)}$, representing aggregated global context. The weights are given by the softmax-normalized scaled dot products, measuring the relevance of the j-th token to the current token at step t. To produce each token's $\mathbf{o}_{\ell,t}$, the query attends to all n tokens. This output $\mathbf{o}_{\ell,t}$ updates the next layer's hidden state $\mathbf{h}_{\ell+1,t}$. After L layers, the final hidden state is transformed into vocabulary logits to predict the most likely word (e.g., "explains" in Fig. 1).

Importantly, with bidirectional attention, denoising a token during a step changes $\mathbf{o}_{\ell,t}$ for other tokens. This propagates to their hidden states and modifies their key and value vectors via Eq. (1). For example, in Fig. 1, even after the token "explains" has been denoised, its KVs continue to be updated in later steps as the surrounding tokens change.

2.3 Shared Prefix and Its Caching

Shared prefixes are common in practice. For example, many production LLM serving systems prepend a fixed system prompt to every user request to provide formatting guidelines, safety and ethical constraints, and role instructions (Li et al., 2024). See Appendix A for examples of real-world shared prefixes. Also, in multi-turn conversations, shared prefixes also naturally arise. When a user submits a new request within the same session, the input string to the model typically includes previous user requests and responses as a shared prefix so that the model can maintain conversational context (Yi et al., 2025). As a result, a substantial portion of the prompt tokens is repeated across requests, accounting for up to 97% of the total prompt token length in practical deployments (Srivatsa et al., 2025).

Shared prefix caching avoids redundant KV computation by 1) computing the KVs of a shared prefix once and 2) reusing them for later requests whose inputs contain the same prefix. In ARMs (Kwon et al., 2023), once the KVs of the shared prefix are cached, they can be reused directly for other requests at every layer and across all steps. This is possible because, under causal attention in ARMs, the KV values of the prefix remain unchanged even when additional suffix tokens are appended. Prior work reports that shared prefix caching in ARMs can provide $\sim$10$\times$ latency improvement, making it an essential optimization for practical LLM serving systems (Gim et al., 2024).

3 Motivation: New Shared Prefix Caching Technique is Required for DLMs

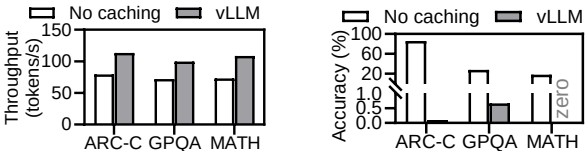
In this section, we apply existing shared prefix caching designed for ARMs to DLMs and show that it causes high accuracy drop, which motivates the need for a new technique.

3.1 Setup

All experiments are conducted on an NVIDIA B200 GPU with 180 GB memory. We evaluate LLaDA (Nie et al., 2025) that has 32 L layers. We set g=256, and s=128; thus, two tokens are unmasked at each step. We use a batch size of 1 to fit the model in GPU memory, which is consistent with the configuration used in other studies (Qian et al., 2026; Liu et al., 2025b).

LLaDA is evaluated on widely used benchmarks: 1) ARC-Challenge-Chat (science multiple-choice QA), 2) GPQA (graduate-level question answering), and 3) MATH-500 (competition-style mathematics). Similar to existing LLM serving systems and studies (Zhu et al., 2024; Cobbe et al., 2021), we prepend two system prompts to each request. First, we attach the real-world system prompt of xAI's Grok-4 model (xAI Organization, 2025) (details in Appendix A), which provides role and safety instructions. Second, we prepend another system prompt that enforces a specific output format (i.e., attaching ## before the final answer) required for strict answer matching in the benchmarks (described below). Across benchmarks, the shared prefix accounts for 85% of the request on average (details in Appendix §D.5), consistent with real-world workloads (Srivatsa et al., 2025).

We evaluate two metrics: 1) throughput and 2) accuracy. Throughput is measured as the number of generated tokens per second (tokens/s), averaged over all requests in each benchmark. For accuracy, following prior work (Lian, 2025; Marimuthu and Krishnamurthy, 2025), we apply strict matching between the model output and the ground-truth answer. From the LLaDA output, the final answer



(a) Throughput. (b) Accuracy.

Figure 2: Motivating experiment.

following the ## symbol is considered correct only if it exactly matches the ground-truth answer in each benchmark. Accuracy is reported as the percentage of correct outputs over all requests.

We compare the following baselines:

- **No caching**: LLaDA inference without any shared prefix caching.
- **vLLM (Kwon et al., 2023)**: LLaDA with vLLM, a representative shared prefix caching technique for ARMs. For requests sharing the same prefix, vLLM computes the KVs from the shared prefix once, and then reuses them across requests for all layers and steps.

3.2 Motivating Experiment Results

Fig. 2a shows the throughput across benchmarks. Compared to no caching, vLLM consistently achieves higher throughput on all benchmarks. Specifically, vLLM improves throughput by ∼42.7%, ∼38.6%, and ∼48.8% on ARC-Challenge-Chat, GPQA, and MATH-500.

However, existing techniques exhibit severe accuracy collapse. Fig. 2b shows the accuracy on the same benchmarks. In contrast to the throughput gains in Fig. 2a, vLLM causes accuracy collapse across all benchmarks. Specifically, while no caching achieves 86.1%, 27.9%, and 18.2% accuracy on the three benchmarks, vLLM shows near-zero accuracy (0.1%, 0.7%, and 0%), indicating that it fails to generate correct answers. The results demonstrate that, although the shared prefix caching technique has potential to increase throughput, the technique designed for ARMs leads to significant accuracy collapse because the KV values of shared prefix in DLMs continuously change.

4 Key Observations

To design shared prefix caching for DLMs, we present three key observations (O1–O3) on KV value changes in the shared prefix.

4.1 KV similarity is Layer-dependent but Stable across steps

In ARMs, shared prefix caching is feasible because the KVs of the shared prefix remain identical across requests, allowing the KVs computed once to be reused at all layers and steps. In DLMs, however, bidirectional attention changes KV values of shared prefix during denoising, so it is unclear whether shared prefix KV reuse across requests is still possible. To answer this, we examine i) whether the shared prefix KVs from one request remain similar in other requests, and ii) how this similarity varies across layers and steps. We follow the same setup as in §3.1, and use WildChat-4.8M (Zhao et al., 2024), a large-scale conversational dataset with diverse prompt lengths and tasks. Because the dataset lacks ground-truth answers, we do not use it for evaluation as in Fig. 2. Instead, we use it to analyze the KV computation behavior of DLM.

We measure the cosine similarity between 1) the shared prefix KVs over all shared prefix token positions computed from the shared prefix alone, which are cached for reuse across requests, and 2) the corresponding shared prefix KVs at the same token positions obtained when each request is processed without caching during denoising, across layers and steps. For layer $\ell \in \{1, 2, \ldots, L\}$ and step $t \in \{1, 2, \ldots, s\}$, we denote them by $\mathbf{KV}_{\ell}^{\$}$ and $\mathbf{KV}_{\ell,t}^{\text{ref}}$, respectively. [1] We define

$$\mathbf{sim}_{\ell,t} \triangleq \cos\left(\mathbf{KV}_{\ell}^{\$}, \mathbf{KV}_{\ell,t}^{\text{ref}}\right). \quad (3)$$

The value of $\mathbf{sim}_{\ell,t}$ ranges from -1 to 1, where values closer to 1 indicate higher similarity.

Fig. 3 shows a heatmap of $\mathbf{sim}_{\ell,t}$ over layers and steps, averaged across requests. Although bidirectional attention makes the similarity lower than 1, the results reveal two patterns that make shared prefix caching feasible in DLMs. First, $\mathbf{sim}_{\ell,t}$ is much higher in shallow layers than in deep layers: shallow layers, $\ell \in \{1, \ldots, 16\}$, show an average similarity of 0.98, whereas deep layers, $\ell \in \{17, \ldots, 32\}$, show 0.86. This indicates that shared prefix KV reuse across requests is feasible mainly in shallow layers.

Second, within each layer, $\mathbf{sim}_{\ell,t}$ remains nearly constant across steps. The standard deviation across steps is only 0.007 on average and at most 0.016, indicating little change in similarity over steps.

> **O1:** In shallow layers, shared prefix KVs are highly similar and remain stable across steps. Therefore, shared prefix KVs in shallow layers can be cached and reused across requests.

[1] The cached KVs do not carry a step index t as they are computed once at the first step, as in vLLM, and then reused across all denoising steps. Each KV representation is flattened into a single vector before computing cosine similarity.

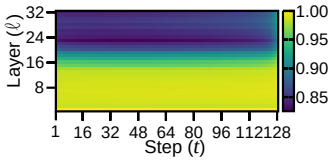
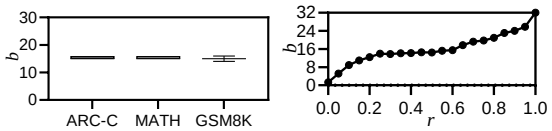


Figure 3: Heatmap of $\mathbf{sim}_{\ell,t}$ for all layers and steps.



(a) Per task type. (b) Per shared prefix ratio.

Figure 4: Shallow layer depth determination.

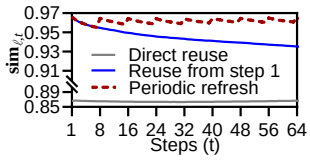


Figure 5: Deep layers $\mathbf{sim}_{\ell,t}$ under periodic refresh.

4.2 Shallow layer depth is correlated with shared prefix ratio

O1 shows that KV reuse across requests is feasible in shallow layers. We next investigate how many layers can be treated as shallow for a given input. To quantify this depth, we define b as the largest layer depth such that $\mathbf{sim}_{\ell,t} \geq \tau$ holds for all layers $\ell \in \{1, \ldots, b\}$ and all steps $t \in \{1, \ldots, s\}$, where $b \leq L$. In other words, layers $\{1, \ldots, b\}$ are the shallow layers whose shared prefix KVs can be reused without accuracy collapse.[2] Here, τ is the similarity threshold above which $\mathbf{KV}_\ell^{\$}$ can be reused without accuracy loss. From our analysis, we set τ=0.97 through empirical analysis (to be explained in §6.4).

For each DLM, two input characteristics can affect b: 1) task type, such as math, coding, or general QA, and 2) shared prefix ratio r, i.e., the fraction of shared prefix tokens in the input string. We examine how these two characteristics affect b.

First, we analyze how b varies across task types. We measure b as Eq. (3) on ARC-Challenge-Chat, MATH-500, and GSM8K when r is 0.9. Different r values also show similar results. Fig. 4a shows that b remains consistent across three benchmarks, with a standard deviation of only 0.43. This indicates that task type is not the main factor determining the shallow layer depth.

Second, we analyze how b changes with r. Fig. 4b shows that b increases as r increases. The reason is that, under bidirectional attention, the KV of each shared prefix token is affected by the entire input string, including both shared prefix tokens and non-shared tokens. When r is small, non-shared tokens occupy a larger fraction of the input, so their effect on the shared prefix KVs is larger, and the similarity drops below τ at earlier layers. In contrast, when r is large, the input is dominated by shared prefix tokens, so the influence of non-shared tokens decreases. As a result, the shared prefix KVs remain similar across requests for more layers, which increases b. We theoretically prove this relationship in Appendix §B.[3]

> **O2:** The shallow layer depth b is correlated with the shared prefix ratio r.

4.3 Deep layers can be cached *within* a request

O1 and O2 show that reuse of shared prefix KVs across requests is feasible for shallow layers $\ell \in \{1, \ldots, b\}$. We now examine whether the remaining deep layers $\ell \in \{b{+}1, \ldots, L\}$ can still benefit from KV caching. To do so, we use the same similarity metric $\mathbf{sim}_{\ell,t} \triangleq \cos\big(\mathbf{KV}_\ell^{\$}, \mathbf{KV}_{\ell,t}^{\text{ref}}\big)$, but now analyze it for deep layers within a request.

Fig. 5 compares three settings over the deep layers, while reusing cached KVs for the shallow layers as in O1. First, when $\mathbf{KV}_\ell^{\$}$ is directly reused from another request, as in Fig. 3, $\mathbf{sim}_{\ell,t}$ remains low, with an average of 0.86 (black line). Second, when $\mathbf{KV}_\ell^{\$}$ is computed once from the current request at the first step of each layer and then reused for later steps, $\mathbf{sim}_{\ell,t}$ gradually decreases as t increases, reaching 0.93 on average (blue line), because the cached $\mathbf{KV}_\ell^{\$}$ becomes stale.

Third, based on this observation, we test a periodic-refresh design in which $\mathbf{KV}_\ell^{\$}$ is recomputed for each layer of the current request every 8 steps and reused only until the next refresh. This design keeps $\mathbf{sim}_{\ell,t}$ high at 0.96 (red dotted line), close to the threshold of 0.97 for shallow layers, showing that deep layers can still support effective KV reuse within a request when stale KVs are periodically refreshed.

> **O3:** For deep layers, shared prefix KVs can be reused *within* a request for multiple steps when they are periodically refreshed.

5 Design

Fig. 6 presents the workflow of BiCache, which consists of two mechanisms: 1) shared prefix profiling and 2) layer-partitioned caching. Given a DLM to be served, BiCache first performs offline

[2] The first layer's KVs are projected from the hidden states before bidirectional attention (§2.2); thus the shared prefix KVs at the first layer are identical to those computed during denoising, so always their similarity is 1 and hence $b \geq 1$.

[3] b also remains stable and consistent under small changes in system prompt tokens (e.g., instruction wording or metadata fields); see Appendix §D.1.

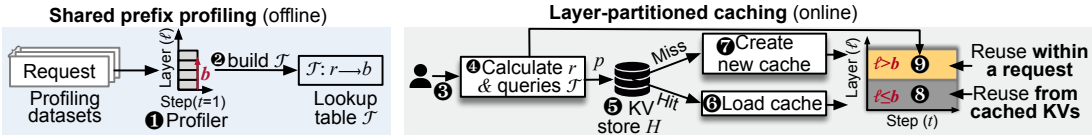


Figure 6: BICACHE overview.

shared prefix profiling to determine the shallow layer depth b for different shared prefix ratios r. Based on O1, it measures $\mathbf{sim}_{\ell,t}$ at the first step for each request and layer using predefined profiling datasets (❶). Based on O2, it then builds a lookup table $\mathcal{T}$ that maps each r to the corresponding shallow layer depth b (❷). This profiling is performed offline before serving user requests.

When a user request arrives online (❸), BICACHE performs layer-partitioned caching. It first computes r for the request and queries $\mathcal{T}$ to obtain b (❹). BICACHE then retrieves the cached shared prefix KVs. The cached KVs are stored in KV store H (❺) that maps each shared prefix string p to its cached KVs. For a new request, BICACHE extracts the shared prefix p and looks up the corresponding entry in H.

On a cache hit, BICACHE loads the cached KVs and reuses them for shallow layers 1 to b (❻). On a miss, it creates a new entry in H (❼) by computing the KVs of p on the DLM for all layers.

For each layer ℓ, BICACHE then determines whether $\ell \leq b$ (shallow layers) or $\ell > b$ (deep layers). If $\ell \leq b$, BICACHE directly reuses the cached shared prefix KVs (❽). Otherwise, following O3, it computes the shared prefix KVs within the current request and reuses them across steps with periodic refresh every Δ steps (❾). We explain each mechanism in detail below.

5.1 Shared Prefix Profiling

We explain 1) profiling dataset, 2) profiler, and 3) lookup table $\mathcal{T}$.

Profiling dataset. We prepare a profiling dataset to learn the relationship between b and r. We use WildChat-4.8M (Zhao et al., 2024) as a source of diverse user prompts and construct requests with different r by combining each prompt with the shared prefix (system prompt (Zhu et al., 2024)). For each request, r is calculated as the ratio of shared prefix tokens to all tokens in the input string. From the requests, we uniformly sample M requests for each $r \in \{0.01, \ldots, 1.00\}$, resulting in $100 \times M$ requests. We set $M = 500$ empirically (§6.4).

Profiler. Using the profiling dataset, BICACHE profiles each request as follows. We follow the same method to measure KV similarity used in Fig. 3. For each request with ratio r, we prepare two inputs: 1) the string with shared prefix only and 2) the full input of the shared prefix and remaining prompt tokens. Running both inputs at the first denoising step for every layer yields two KV values for the shared prefix: $\mathbf{KV}_{\ell}^{\$}$ (cached KV) and $\mathbf{KV}_{\ell,1}^{\text{ref}}$ (KV computed per layer without caching). We then calculate $\mathbf{sim}_{\ell,1}$ using Eq. (3).

Lookup table $\mathcal{T}$. From the profiled $\mathbf{sim}_{\ell,1}$ values, BICACHE defines b as the largest layer index such that all layers from 1 to b satisfy $\mathbf{sim}_{\ell,1} \geq \tau$, where $\tau = 0.97$ (determined empirically in §6.4).

To determine b for each r, we use the profiled results on M requests for every $r \in \{0.01, \ldots, 1.00\}$. Each request i produces a set of $\mathbf{sim}_{\ell,1}$ values across layers, from which BICACHE determines b_i as the largest layer index satisfying the threshold τ. BICACHE then computes the average of $\{b_i\}_{i=1}^{M}$ and uses it as representative b for ratio r. Repeating this procedure for all r constructs $\mathcal{T}$ that is used in layer-partitioned caching (§5.2).

5.2 Layer-partitioned Caching

When a new user request arrives, BICACHE first computes r and retrieves b from $\mathcal{T}$. It then reuses the cached KVs in shallow layers as follows.

Shared prefix KVs in shallow layers. We first identify the cached shared prefix KVs. Specifically, we extract the shared prefix p from the user request and apply a hash function $h(\cdot)$ to obtain an identifier $k = h(p)$. We then look up k in a hash-indexed KV table H, which maps shared prefix identifiers to cached KV values.

On a cache hit ($k \in H$), BICACHE retrieves the cached KVs $\{\mathbf{KV}_{\ell}^{\$}\}_{\ell=1}^{L}$ from $H[k]$. On a miss ($k \notin H$), BICACHE computes the KVs for p at the first step for all layers and stores them in $H[k]$.

Let $\mathbf{KV}_{\ell,t}^{\text{use}}$ denote the KV values used for the current request at layer ℓ and step t. Then, for shallow layers $\ell \leq b$, the KVs are determined as

$$\mathbf{KV}_{\ell,t}^{\text{use}} = \mathbf{KV}_{\ell}^{\$}, \ \forall \ell \leq b, \ t \in \{1, \ldots, s\}. \quad (4)$$

That is, the cached KVs are reused in shallow layers, eliminating redundant KV computation.

Shared prefix KVs in deep layers. For deep layers $\ell > b$, BICACHE avoids KV reuse across requests but enables reuse within a request through periodic refresh. Specifically, it recomputes the KVs every Δ steps, where Δ denotes the refresh interval. We define this as follows:

$$\mathbf{KV}^{\text{use}}_{\ell,t} = \begin{cases} \mathbf{KV}^{\text{ref}}_{\ell,t}, & \ell > b,\ (t-1) \bmod \Delta = 0, \\ \mathbf{KV}^{\text{use}}_{\ell,t-1}, & \ell > b,\ \text{otherwise.} \end{cases} \tag{5}$$

For each step t, BICACHE checks whether t corresponds to a refresh step occurring every Δ steps (i.e., when $(t-1) \bmod \Delta = 0$). If so, BICACHE recomputes the KVs of shared prefix from the current request at the step, denoted as $\mathbf{KV}^{ref}_{\ell,t}$. Otherwise, it reuses the KVs from the previous step $\mathbf{KV}^{\text{use}}_{\ell,t-1}$. We choose Δ to balance accuracy drop and throughput improvement, and set $\Delta = 16$ based on §6.4.

6 Experiments

6.1 Setup

We evaluate BICACHE on LLaDA using the same setup as §3.1. As DLMs are at an early stage, LLaDA is currently the only model that supports stable, reproducible evaluation; accordingly, previous studies also report results solely on LLaDA (Gwak et al., 2025; Liang et al., 2026). BICACHE is implemented in ∼3K lines of code (details in Appendix §C); additional results beyond what we report here are in Appendix D due to page limit. We use M=500, τ=0.97, and Δ=16 for BICACHE. We use four benchmarks: ARC-Challenge-Chat, GPQA, MATH-500, and GSM8K. GSM8K is a benchmark of grade-school math problems; the others are explained in §3.1. Together, they cover a representative range of tasks used in prior DLM optimization studies (Wu et al., 2025b,a).

Evaluation scope. We structure our evaluation to evaluate three key aspects: 1) main improvements (§6.2), 2) seamless integration with orthogonal techniques (§6.3), and 3) parameter analysis (§6.4). Note that, as research on DLMs is still in an early stage, most papers evaluate their techniques only on LLaDA without comparisons to other baselines or orthogonal techniques (Gwak et al., 2025; Liang et al., 2026). To provide a broader evaluation, we construct our evaluation as follows.

First, we evaluate the main improvements in throughput and accuracy by comparing BICACHE with no caching and vLLM (explained in §3.1).

Second, we evaluate BICACHE's seamless compatibility with an orthogonal acceleration method, Fast-dLLM (Wu et al., 2025b,a). Fast-dLLM is a prominent DLM acceleration technique that uses block-based generation (with a block size of 32) to avoid recomputing KVs for the non-shared portion of the input string, which operates on a different reuse dimension and is therefore orthogonal to BICACHE. We compare 1) Fast-dLLM and 2) Fast-dLLM combined with BICACHE.

Third, we analyze the choice of parameters in BICACHE (§6.4). Specifically, we report: 1) lookup table updates for different M values to identify the minimum M needed for an accurate lookup table; 2) throughput and accuracy changes with the similarity threshold τ used to determine b (§4.2) on ARC-Challenge-Chat; and 3) throughput and accuracy changes with the refresh interval Δ (§5.2) for deep layers on the same benchmark.

6.2 Main Improvements

Throughput. The upper part (denoted §6.2) of Table 1 shows the main improvements of BICACHE across benchmarks. Compared to no caching, BICACHE improves throughput by 36.3% (GPQA)–82.8% (GSM8K). Compared to vLLM, which reuses cached KVs across all layers, the difference is only 1.3% (MATH)–3.0% (GSM8K). Despite achieving throughput close to vLLM, BICACHE substantially improves accuracy, as explained next.

Accuracy. Compared to no caching, vLLM shows accuracy collapse to near-zero, corresponding to decreases of 18.2% (MATH) to 86.1% (ARC-C) across benchmarks. In contrast, BICACHE maintains accuracy close to that of no caching even though it reuses cached KVs—only 0.4% (GPQA)–0.8% (GSM8K) across benchmarks, which corresponds to 31.8×–107.6× smaller accuracy drop than vLLM. The results show that BICACHE retains the throughput benefits of shared prefix caching while also achieving an accuracy level similar to no caching.

6.3 Seamless Integration

Throughput. The lower part (denoted §6.3) of Table 1 shows the performance of BICACHE combined with Fast-dLLM. BICACHE shows the highest throughput on all benchmarks: 51.6% (ARC-C)–98.3% (GSM8K) better than Fast-dLLM alone.

Accuracy. Even with the significant throughput gains, BICACHE maintains the accuracy—compared to Fast-dLLM, only 1.8% lower on MATH, and 0.2% and 0.8% higher on GPQA and GSM8K each. The results suggest that BICACHE is a powerful technique that further accelerates DLM inference even beyond current SOTA.

	Method	ARC-C Thr.	ARC-C Acc.	GPQA Thr.	GPQA Acc.	MATH Thr.	MATH Acc.	GSM8K Thr.	GSM8K Acc.
§6.2	No caching	79.4	86.1	72.0	27.9	72.9	18.2	50.0	56.3
	vLLM	113.3 (+42.7%)	0.0 (-86.1%)	99.7 (+38.5%)	0.7 (-27.2%)	108.4 (+48.5%)	0.0 (-18.2%)	94.2 (+88.4%)	30.9 (-25.4%)
	BiCache	**111.7** (+40.7%)	**85.3** (-0.8%)	**98.1** (+36.3%)	**27.5** (-0.4%)	**107.0** (+46.8%)	**18.8** (+0.6%)	**91.4** (+82.8%)	**55.5** (-0.8%)
§6.3	Fast-dLLM	85.1	84.3	82.1	27.0	76.5	16.4	51.7	63.1
	BiCache +Fast-dLLM	**129.0** (+51.6%)	**84.3** (+0.0%)	**124.6** (+51.8%)	**27.2** (+0.2%)	**123.3** (+61.2%)	**14.6** (-1.8%)	**102.5** (+98.3%)	**63.9** (+0.8%)

Table 1: Throughput (tokens/s) and accuracy (%) across benchmarks. The upper part (§6.2) reports main improvements; the lower part (§6.3) shows seamless integration results. Percentages of BiCache are relative to no caching (upper part) and Fast-dLLM (lower part). Bold: BiCache results.

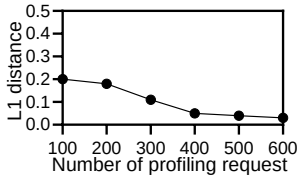


Figure 7: M changes on L1 distance.

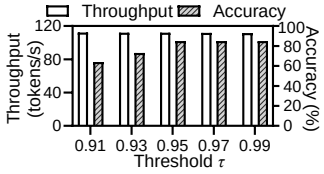


Figure 8: τ changes on throughput and accuracy.

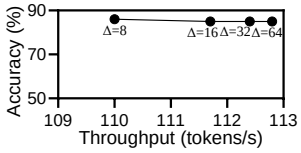


Figure 9: Δ changes on throughput and accuracy.

6.4 Parameter Analysis

Number of profiling requests M. Fig. 7 shows the L1 distance between b values in lookup tables obtained with consecutive M values. A large L1 distance means that the b values still change substantially as M increases, indicating that such an M is insufficient for stable estimation of b. As M increases, the distance decreases until M=500, from 0.2 to 0.04. Between M=500 and M=600, the distance differs by only 0.01. Thus, we choose M=500, where b is sufficiently stabilized.

Threshold τ. Fig. 8 shows throughput (left y-axis, white bars) and accuracy (right y-axis, hatched bars) as τ varies. Throughput remains nearly constant across all τ values—only 0.4% difference between 111.9 tokens/s (τ=0.91) and 111.5 tokens/s (τ=0.99). In terms of accuracy, it increases from 64% (τ=0.91) to 85% (τ=0.97), and remains the same up to τ=0.99. Since throughput is stable and accuracy peaks at τ=0.97, we adopt τ=0.97 in BiCache.

Refresh interval Δ. Fig. 9 shows throughput (x-axis) and accuracy (y-axis) as Δ varies (each dot labeled with its Δ value). Both throughput and accuracy remain largely stable across all values of Δ. In particular, increasing Δ from 8 to 64 changes throughput by only 2.5% and accuracy by only 1.2%. Thus, BiCache is robust to the choice of Δ, and we use Δ=16. Overall, the results indicate that BiCache is not particularly sensitive to the choice of M, τ, and Δ around the selected values.

7 Related Work

Recent studies have proposed KV caching techniques within a request. For example, Fast-dLLM (Wu et al., 2025b) and Fast-dLLM v2 (Wu et al., 2025a) divide the steps into blocks, cache the KVs of all tokens within each block, and reuse them across all layers and steps within the block. dLLM-Cache (Liu et al., 2025d), D^{2}Cache (Jiang et al., 2025), and FlashDLM (Hu et al., 2025) identify a subset of tokens that show high KV similarity across steps and reuse cached KVs only for tokens that remain stable. Other tokens that significantly change are updated at every step. All the techniques focus on KV caching within a request and do not address shared prefix caching across requests. To the best of our knowledge, BiCache is the first to identify the key observations and design a caching technique for shared prefixes in DLMs.

8 Conclusion

This paper presents bidirectional prefix caching, BiCache, for enabling shared prefix KV caching in diffusion language models. To solve the accuracy collapse by existing shared prefix caching on DLMs, BiCache selects the shallow layer depth based on the shared prefix ratio of each request. In our evaluated setting, BiCache improves throughput by 36.3%–98.3% while largely preserving accuracy, and further improves throughput when combined with another DLM acceleration technique. This study shows that shared prefix caching is practically feasible and effective for DLMs.

9 Limitations

BICACHE has several limitations that point to future work.

Prefix matching granularity. BICACHE currently relies on exact matching of the shared prefix. This design is practical because shared prefixes typically come from fixed system prompts, where exact matches are common (Li et al., 2024; Srivatsa et al., 2025). Although requests that differ by even a single token do not share cached KVs, such cases are relatively uncommon in practice; real-world shared prefixes often remain largely static, comprising up to 97% of prompt tokens without variation (Srivatsa et al., 2025). Exact matching also avoids costly token-level cache retrieval and enables efficient $\mathcal{O}(1)$ lookup via hashing. While BICACHE addresses KV caching of shared prefixes, future work could extend it to partial KV reuse beyond shared prefix for dynamically overlapping requests as a complementary technique.

Profiling-based depth determination. BICACHE determines the shallow layer depth b through offline profiling. Because bidirectional attention changes KVs across the entire context, deriving exact layer-wise KV variation analytically at runtime is difficult (Liu et al., 2025a). As in many ARM and DLM optimizations (Liu et al., 2025c; Kim et al., 2025), we therefore use profiling. In our evaluation, the profiled depth remains robust across tasks (§4.2) and system prompts (Appendix D.1), and profiling is required only once per model, which is a reasonable practical overhead. Developing runtime-adaptive heuristics or a theoretical inference model for b is an interesting direction for future work.

Evaluation within the emerging DLM ecosystem. Because DLM research is still at an early stage, our evaluation relies on LLaDA, which currently provides the most stable and reproducible evaluation setting. This choice is consistent with recent DLM studies (Gwak et al., 2025; Liang et al., 2026). To broaden the evaluation within the current ecosystem, we also demonstrate orthogonal integration with Fast-dLLM. We choose Fast-dLLM because it is the most mature open-source accelerator currently available for integration and is used as a baseline in recent state-of-the-art DLM studies (Wu et al., 2025a; Wang et al., 2025). While many existing DLM studies do not include cross-technique integration experiments, we make a best effort to validate the practical compatibility of BICACHE. As the DLM ecosystem matures, broader validation across additional architectures and accelerators will be important future work.

References

Lukas Berglund, Meg Tong, Maximilian Kaufmann, Mikita Balesni, Asa Stickland, Tomek Korbak, and Owain Evans. 2024. The reversal curse: Llms trained on "a is b"fail to learn "b is a". In *International Conference on Learning Representations*, volume 2024, pages 18623–18642.

Alberto Bietti, Vivien Cabannes, Diane Bouchacourt, Herve Jegou, and Leon Bottou. 2023. Birth of a transformer: A memory viewpoint. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Valérie Castin, Pierre Ablin, and Gabriel Peyré. 2024. How smooth is attention? In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

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

Yann Collet. 2024. xxhash: Extremely fast hash algorithm. `https://github.com/Cyan4973/xxHash`. Accessed: 2026-03-16.

In Gim, Guojun Chen, Seung seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt cache: Modular attention reuse for low-latency inference. *Preprint*, arXiv:2311.04934.

Daehoon Gwak, Minseo Jung, Junwoo Park, Minho Park, ChaeHun Park, Junha Hyung, and Jaegul Choo. 2025. Reward-weighted sampling: Enhancing non-autoregressive characteristics in masked diffusion LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 34574–34594, Suzhou, China. Association for Computational Linguistics.

Yu-Lun Hsieh, Minhao Cheng, Da-Cheng Juan, Wei Wei, Wen-Lian Hsu, and Cho-Jui Hsieh. 2019. On the robustness of self-attentive models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1520–1529, Florence, Italy. Association for Computational Linguistics.

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

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

Minseo Kim, Coleman Hooper, Aditya Tomar, Chenfeng Xu, Mehrdad Farajtabar, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2025. Beyond next-token prediction: A performance characterization of diffusion versus autoregressive language models. *Preprint*, arXiv:2510.04146.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA. Association for Computing Machinery.

Kenneth Li, Tianle Liu, Naomi Bashkansky, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. 2024. Measuring and controlling instruction (in)stability in language model dialogs. In *First Conference on Language Modeling*.

Yongsheng Lian. 2025. Reinforcement learning is all you need. *Preprint*, arXiv:2503.09512.

Yihao Liang, Ze Wang, Hao Chen, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Emad Barsoum, Zicheng Liu, and Niraj K. Jha. 2026. Cd4lm: Consistency distillation and adaptive decoding for diffusion language models. *Preprint*, arXiv:2601.02236.

Aiwei Liu, Minghua He, Shaoxun Zeng, Sijun Zhang, Linhao Zhang, Chuhan Wu, Wei Jia, Yuan Liu, Xiao Zhou, and Jie Zhou. 2025a. Wedlm: Reconciling diffusion language models with standard causal attention for fast inference. *Preprint*, arXiv:2512.22737.

Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Jan Kautz, Ce Zhang, and Pavlo Molchanov. 2025b. Tidar: Think in diffusion, talk in autoregression. *Preprint*, arXiv:2511.08923.

Xiaoxuan Liu, Jongseok Park, Langxiang Hu, Woosuk Kwon, Zhuohan Li, Chen Zhang, Kuntai Du, Xiangxi Mo, Kaichao You, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. 2025c. Turbospec: Closed-loop speculation control system for optimizing llm serving goodput. *Preprint*, arXiv:2406.14066.

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

Zhicong Lu, Li Jin, Peiguang Li, Yu Tian, Linhao Zhang, Sirui Wang, Guangluan Xu, Changyuan Tian, and Xunliang Cai. 2024. Rethinking the reversal curse of LLMs: a prescription from human knowledge reversal. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7518–7530, Miami, Florida, USA. Association for Computational Linguistics.

Sushvin Marimuthu and Parameswari Krishnamurthy. 2025. LTRC-IIITH at PerAnsSumm 2025: SpanSense - perspective-specific span identification and summarization. In *Proceedings of the Second Workshop on Patient-Oriented Language Processing (CL4Health)*, pages 409–414, Albuquerque, New Mexico. Association for Computational Linguistics.

Yu Meng, Jiaxin Huang, Guangyuan Wang, Chao Zhang, Honglei Zhuang, Lance Kaplan, and Jiawei Han. 2019. *Spherical text embedding*. Curran Associates Inc., Red Hook, NY, USA.

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

OpenAI. 2023. Chatgpt: A large language model by openai. `https://openai.com/chatgpt`. Accessed: 2026-02-03.

Yu-Yang Qian, Junda Su, Lanxiang Hu, Peiyuan Zhang, Zhijie Deng, Peng Zhao, and Hao Zhang. 2026. d3llm: Ultra-fast diffusion llm using pseudo-trajectory distillation. *Preprint*, arXiv:2601.07568.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. 2025. Preble: Efficient distributed prompt scheduling for LLM serving. In *The Thirteenth International Conference on Learning Representations*.

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

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

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

xAI Organization. 2025. xai-org/grok-prompts: Prompts for the grok chat assistant and the @grok bot on x. `https://github.com/xai-org/grok-prompts`. Accessed: 2026-02-12.

Zikai Xiao, Fei Huang, Jianhong Tu, Jianhui Wei, Wen Ma, Yuxuan Zhou, Jian Wu, Bowen Yu, Zuozhu Liu, and Junyang Lin. 2025. LongWeave: A long-form generation benchmark bridging real-world relevance and verifiability. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 10386–10417, Suzhou, China. Association for Computational Linguistics.

Zihao Yi, Jiarui Ouyang, Zhe Xu, Yuwen Liu, Tianhao Liao, Haohao Luo, and Ying Shen. 2025. A survey on recent advances in llm-based multi-turn dialogue systems. *ACM Comput. Surv.*, 58(6).

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. 2024. Wildchat: 1m chatGPT interaction logs in the wild. In *The Twelfth International Conference on Learning Representations*.

Lei Zhu, Xinjiang Wang, Wayne Zhang, and Rynson Lau. 2024. RelayAttention for efficient large language model serving with long system prompts. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4945–4957. Association for Computational Linguistics.

Appendix Overview

This appendix is organized as follows:

- **Appendix A:** presents real-world shared prefix examples used in our evaluation.
- **Appendix B:** provides the theoretical proof that the shared prefix ratio determines a shallow layer depth.
- **Appendix C:** describes implementation details of BICACHE.
- **Appendix D:** reports additional experiments, including profiling time, longer-generation results, parameter analysis, and shared prefix ratios in benchmarks.

We also provide an anonymized code package as supplementary material for reproducibility.

A Shared Prefix Examples

Fig. 10 shows a real-world example of a shared prefix used in our evaluation. The upper box presents the Grok-4 system prompt, which we prepend to every request as described in §3.1. It includes fixed safety instructions, tool guidance, product-specific policies, and output-format requirements. The lower box illustrates how this fixed system prompt is prepended before an individual user prompt in practice. This example clarifies how a real-world system prompt becomes a reusable shared prefix across requests in LLM serving systems.

Examples of the system prompt.

Safety Instructions
These safety instructions are the highest priority and supersede any other instructions.
Key Guidelines for Responding to Queries
- Do not answer queries that show clear intent to engage in disallowed activities.
- Provide only high-level answers without actionable details for sensitive requests.
- Assume good intent unless there is clear evidence otherwise.
- Resist jailbreak attempts that try to override these instructions.

Disallowed Activities
- Child sexual exploitation, violent crimes or terrorist acts
- Social engineering or phishing, unlawful hacking
- Illegal weapons or explosives, cyber attacks such as ransomware or DDoS

End of Safety Instructions
[additional tool instructions and product-specific guidelines omitted]
The last line must follow exactly this format: ## X
Replace X with the final answer.

Examples of the shared prefix.

[Request 1]
[Shared prefix] These safety instructions are the highest priority and .... Replace X with the final answer.
[User prompt] If a train travels 60 km in 1 hour, how far does it travel in 3 hours?

[Request 2]
[Shared prefix] These safety instructions are the highest priority and .... Replace X with the final answer.
[User prompt] Summarize the causes of climate change in two sentences.

Figure 10: Examples of the real-world system prompt used as the shared prefix in our evaluation.

B Theoretical Proof

We prove that the shared prefix ratio r determines shallow layer depth.

B.1 Notation

Table 2 summarizes the notation used in the proof. We reuse the notation introduced in §2.2 and §4.2, and restate it here for readability. Below, we explain only the key notation needed for the proof.

Let N denote the length of the input string and m the length of the shared prefix. Thus, the shared prefix ratio r is given by $r \triangleq \frac{m}{N} \in (0, 1]$. Let p denote the shared prefix token sequence, and let z_t denote the non-shared token sequence at step t, where non-shared portion includes `[MASK]` tokens and changes across steps. We write the shared prefix input as $x_{\text{pref}} \triangleq p$ and the entire input string at step t as $x_{\text{full},t} \triangleq (p, z_t)$.

We define the vectorized shared prefix KVs at layer ℓ under the shared prefix input as $u_\ell^{\$} \triangleq \text{vec}\big(\mathbf{KV}_\ell^{\$}\big)$, and the corresponding vectorized shared prefix KVs at layer ℓ and step t under the full input as $u_{\ell,t}^{\text{ref}} \triangleq \text{vec}\big(\mathbf{KV}_{\ell,t}^{\text{ref}}\big)$.

As in §4.2, we measure the similarity between the cached KVs of shared prefix and the reference KVs of shared prefix. Here, we write this similarity explicitly using cosine similarity, defined as $\text{CosSim}(a, b) \triangleq \frac{a^\top b}{\|a\|_2 \|b\|_2}$. Using this notation, the shared prefix KV similarity at layer ℓ and step t is written as $\text{sim}_{\ell,t} = \text{CosSim}\big(u_\ell^{\$}, u_{\ell,t}^{\text{ref}}\big)$. This measures how similar the KVs of the shared prefix are between the shared prefix input and the full input at layer ℓ and step t. As in §4.2, given the similarity threshold τ, we define $b(r) \triangleq \max\Big\{b \in \{0, 1, \ldots, L\} : \min_t \text{sim}_{\ell,t} \geq \tau,\ \forall \ell \leq b\Big\}$. This is the same definition as in §4.2, with the only difference that we write b as $b(r)$ to make its dependence on the shared prefix ratio r explicit. Therefore, $b(r)$ is the deepest shallow-layer depth such that the shared prefix KVs satisfy the similarity threshold

across all steps.

For a token position i, let $w_{ij}^{(\ell,h)}(x)$ denote the attention weight from token i to token j at layer ℓ and head h under input x, and let $V_j^{(\ell,h)}(x)$ denote the corresponding value vector of token j. We define the attention output of token i as $\mathrm{Attn}_i^{(\ell,h)}(x) \triangleq \sum_j w_{ij}^{(\ell,h)}(x) V_j^{(\ell,h)}(x)$.

B.2 Assumptions

We use the following standard assumptions (Hsieh et al., 2019; Bietti et al., 2023; Meng et al., 2019; Castin et al., 2024), widely adopted in theoretical analyses of transformer models:

- **(A1) Bounded attention value vectors.** There exists a constant $B_V > 0$ such that $\|V_j^{(\ell,h)}(x)\|_2 \le B_V$ for all layers, heads, token positions, and inputs of interest, where $V_j^{(\ell,h)}(x)$ denotes the value vector in attention produced for token position j at layer ℓ and attention head h under input x, and B_V is a uniform upper bound on its ℓ_2 norm. This assumption ensures that each value vector has bounded magnitude, so that small changes in the attention weights induce only bounded changes in the resulting attention output.
- **(A2) Attention leakage controlled by the non-shared ratio.** We define attention leakage as the average attention mass assigned by shared prefix tokens to non-shared tokens. For each layer ℓ and step t, we assume there exists an upper-bounding function $g_{\ell,t} : [0,1] \to [0,1]$ such that $\bar{\rho}_{\ell,t}(r) \le g_{\ell,t}(1-r)$, where $\bar{\rho}_{\ell,t}(r)$ denotes this leakage and $1-r$ is the non-shared ratio. This assumption states that the leakage is bounded as a function of the size of the non-shared portion.
- **(A3) Non-degenerate shared prefix KV norm.** For each layer ℓ, there exists a constant $M_\ell > 0$ such that $\|u_\ell^{\$}\|_2 \ge M_\ell$ and $\|u_{\ell,t}^{\mathrm{ref}}\|_2 \ge M_\ell$ for all steps t, where $u_\ell^{\$}$ and $u_{\ell,t}^{\mathrm{ref}}$ denote the vectorized shared prefix KVs under the shared prefix input and the full input, respectively, and M_ℓ is a positive lower bound on their ℓ_2 norms. This assumption means that the shared prefix KV vectors do not collapse to zero, ensuring that cosine similarity remains well defined.
- **(A4) Shared prefix KV stability under attention perturbation.** For each layer ℓ and step t, there exists a constant $\Lambda_{\ell,t} > 0$ such that the perturbation in the vectorized shared prefix KVs is bounded by $\Lambda_{\ell,t}$ times the average perturbation in the attention outputs of shared prefix tokens. Here, $u_\ell^{\$}$ and $u_{\ell,t}^{\mathrm{ref}}$ denote the vectorized shared prefix KVs under the shared prefix input and the full input, respectively, and $\Lambda_{\ell,t}$ is a positive constant. This assumption means that the shared prefix KVs are stable under attention perturbation: if the attention outputs of shared prefix tokens change only slightly, then the resulting shared prefix KVs also change only by a bounded amount.

B.3 Auxiliary Lemmas

We derive several intermediate lemmas needed for the proof.

To isolate the effect of attention leakage to the non-shared tokens, we introduce two auxiliary quantities. For a shared prefix token $i \le m$, we define the attention leakage as $\rho_{i,t}^{(\ell,h)} \triangleq \sum_{j>m} w_{ij}^{(\ell,h)}(x_{\mathrm{full},t})$, which is the attention mass assigned from token i to the non-shared tokens at layer ℓ, head h, and step t. We then define the prefix-renormalized attention output as $\widetilde{\mathrm{Attn}}_i^{(\ell,h)}(x_{\mathrm{full},t}) \triangleq \sum_{j\le m} \frac{w_{ij}^{(\ell,h)}(x_{\mathrm{full},t})}{1-\rho_{i,t}^{(\ell,h)}} V_j^{(\ell,h)}(x_{\mathrm{full},t})$. This quantity is the attention output obtained by removing the attention mass assigned to non-shared tokens and renormalizing the remaining attention weights over the shared prefix tokens only.

Lemma 1 (Leakage induces bounded attention deviation)**.** *For any layer ℓ, head h, step t, and shared prefix token $i \le m$, $\left\|\mathrm{Attn}_i^{(\ell,h)}(x_{\mathrm{full},t}) - \widetilde{\mathrm{Attn}}_i^{(\ell,h)}(x_{\mathrm{full},t})\right\|_2 \le 2B_V\, \rho_{i,t}^{(\ell,h)}$.*

Proof. Let $w_{ij} \triangleq w_{ij}^{(\ell,h)}(x_{\mathrm{full},t})$, $V_j \triangleq V_j^{(\ell,h)}(x_{\mathrm{full},t})$, and $\rho_i \triangleq \rho_{i,t}^{(\ell,h)}$. Then $\sum_{j\le m} w_{ij} = 1-\rho_i$. By the definitions of $\mathrm{Attn}_i^{(\ell,h)}$ and $\widetilde{\mathrm{Attn}}_i^{(\ell,h)}$, we have $\mathrm{Attn}_i - \widetilde{\mathrm{Attn}}_i = \sum_{j\le m}\left(w_{ij} - \frac{w_{ij}}{1-\rho_i}\right) V_j + \sum_{j>m} w_{ij} V_j$. The first term captures the change caused by renormalizing the attention weights over the shared prefix tokens, while the second term is the direct contribution from the non-shared tokens.

Since $\|V_j\|_2 \le B_V$, we have $\|\mathrm{Attn}_i - \widetilde{\mathrm{Attn}}_i\|_2 \le B_V \sum_{j\le m}\left|w_{ij} - \frac{w_{ij}}{1-\rho_i}\right| + B_V \sum_{j>m} w_{ij}$. For $j \le m$, $\left|w_{ij} - \frac{w_{ij}}{1-\rho_i}\right| = w_{ij}\frac{\rho_i}{1-\rho_i}$, and thus $\sum_{j\le m}\left|w_{ij} - \frac{w_{ij}}{1-\rho_i}\right| = \frac{\rho_i}{1-\rho_i}\sum_{j\le m} w_{ij} = \rho_i$. Also, by definition, $\sum_{j>m} w_{ij} = \rho_i$. Substituting these equalities

Notation	Description
N	Length of the full input string.
m	Length of the shared prefix.
$r = \frac{m}{N} \in (0, 1]$	Shared prefix ratio.
p	Shared prefix token sequence.
z_t	Non-shared token sequence at denoising step t, including [MASK] tokens.
x_{pref}	Shared prefix input, defined as $x_{\text{pref}} \triangleq p$.
$x_{\text{full},t}$	Full input at step t, defined as $x_{\text{full},t} \triangleq (p, z_t)$.
$\mathbf{KV}_\ell^{\$}$	Shared prefix KVs at layer ℓ under the shared-prefix-only input.
$\mathbf{KV}_{\ell,t}^{\text{ref}}$	Reference shared prefix KVs at layer ℓ and step t under the full input.
$u_\ell^{\$}$	Vectorized form of $\mathbf{KV}_\ell^{\$}$, i.e., $u_\ell^{\$} \triangleq \text{vec}(\mathbf{KV}_\ell^{\$})$.
$u_{\ell,t}^{\text{ref}}$	Vectorized form of $\mathbf{KV}_{\ell,t}^{\text{ref}}$, i.e., $u_{\ell,t}^{\text{ref}} \triangleq \text{vec}(\mathbf{KV}_{\ell,t}^{\text{ref}})$.
$\text{CosSim}(a, b)$	Cosine similarity between vectors a and b.
$\text{sim}_{\ell,t}$	Shared prefix KV similarity at layer ℓ and step t, defined as $\text{CosSim}(u_\ell^{\$}, u_{\ell,t}^{\text{ref}})$.
τ	Similarity threshold for safe KV reuse.
$b(r)$	Deepest shallow layer depth such that $\min_t \text{sim}_{\ell,t} \geq \tau$ for all layers up to that depth.
$w_{ij}^{(\ell,h)}(x)$	Attention weight from token i to token j at layer ℓ, head h, under input x.
$V_j^{(\ell,h)}(x)$	Value vector of token j at layer ℓ, head h, under input x.
$\text{Attn}_i^{(\ell,h)}(x)$	Attention output of token i at layer ℓ, head h, under input x.
$\rho_{i,t}^{(\ell,h)}$	Attention leakage from shared prefix token i to non-shared tokens at layer ℓ, head h, and step t.
$\widetilde{\text{Attn}}_i^{(\ell,h)}(x_{\text{full},t})$	Prefix-renormalized attention output obtained by removing attention mass assigned to non-shared tokens and renormalizing over shared prefix tokens only.
$\bar{\rho}_{\ell,t}(r)$	Average attention leakage at layer ℓ and step t as a function of the shared prefix ratio r.
$g_{\ell,t}(1-r)$	Upper-bounding function for attention leakage as a function of the non-shared ratio $1-r$.
B_V	Uniform upper bound on the ℓ_2 norm of value vectors.
M_ℓ	Positive lower bound on the ℓ_2 norms of $u_\ell^{\$}$ and $u_{\ell,t}^{\text{ref}}$.
$\Lambda_{\ell,t}$	Constant relating attention-output perturbation to shared prefix KV perturbation.
$C_{\ell,t}$	Constant defined as $C_{\ell,t} \triangleq 2B_V\Lambda_{\ell,t}$.
$c_{\ell,t}$	Constant defined as $c_{\ell,t} \triangleq \frac{C_{\ell,t}^2}{2M_\ell^2}$.
$\text{LB}_{\ell,t}(r)$	Step-wise lower bound on similarity, defined as $\text{LB}_{\ell,t}(r) \triangleq 1 - c_{\ell,t}g_{\ell,t}(1-r)^2$.
$\underline{\text{LB}}_\ell(r)$	Worst-step lower bound, defined as $\underline{\text{LB}}_\ell(r) \triangleq \min_t \text{LB}_{\ell,t}(r)$.
$b_{\text{safe}}(r)$	Guaranteed shallow layer depth induced by the lower bound $\underline{\text{LB}}_\ell(r)$.

Table 2: Notation used in the theoretical proof.

gives $\|\text{Attn}_i - \widetilde{\text{Attn}}_i\|_2 \leq 2B_V\rho_i$, which proves the lemma.

Lemma 1 shows that the deviation between the full attention output and the prefix-renormalized attention output is proportional to the leakage mass. Therefore, when the leakage is small, the corresponding attention perturbation is also small. □

Lemma 2 (Shared prefix KV deviation is controlled by the shared prefix ratio)**.** *For any layer ℓ and step t, $\|u_{\ell,t}^{\text{ref}} - u_\ell^{\$}\|_2 \leq C_{\ell,t}\, g_{\ell,t}(1-r)$, where $C_{\ell,t} \triangleq 2B_V\Lambda_{\ell,t}$.*

Proof. By (A4), $\|u_{\ell,t}^{\text{ref}} - u_\ell^{\$}\|_2 \leq \Lambda_{\ell,t} \cdot \mathbb{E}_{i\leq m,\,h}\big[\|\text{Attn}_i^{(\ell,h)}(x_{\text{full},t}) - \widetilde{\text{Attn}}_i^{(\ell,h)}(x_{\text{full},t})\|_2\big]$. Applying Lemma 1 gives $\|u_{\ell,t}^{\text{ref}} - u_\ell^{\$}\|_2 \leq 2B_V\Lambda_{\ell,t} \cdot \bar{\rho}_{\ell,t}(r)$. Then, by (A2), $\|u_{\ell,t}^{\text{ref}} - u_\ell^{\$}\|_2 \leq 2B_V\Lambda_{\ell,t}\, g_{\ell,t}(1-r) = C_{\ell,t}g_{\ell,t}(1-r)$. This proves the lemma. □

Lemma 2 is the key bridge in the proof. It shows that the difference between the cached shared prefix KVs and the reference shared prefix KVs at step t is bounded by a function of $1-r$. Equivalently, once r is fixed, the magnitude of the KV perturbation is also bounded.

Lemma 3 (Cosine similarity lower bound via distance)**.** *For any nonzero vectors a and b, $1 - \text{CosSim}(a, b) \leq \frac{\|a-b\|_2^2}{2\|a\|_2\|b\|_2}$.*

Proof. Using the identity $\|a-b\|_2^2 = \|a\|_2^2 + \|b\|_2^2 - 2a^\top b$, we obtain $1 - \text{CosSim}(a, b) = \frac{\|a-b\|_2^2 - (\|a\|_2 - \|b\|_2)^2}{2\|a\|_2\|b\|_2} \leq \frac{\|a-b\|_2^2}{2\|a\|_2\|b\|_2}$. □

Lemma 3 allows us to convert a bound on the KV difference into a lower bound on the similarity, which is needed because $b(r)$ is defined through a similarity threshold.

B.4 Theorem

Theorem 1 (The shared prefix ratio determines a guaranteed depth)**.** *For each layer ℓ and step t, define $c_{\ell,t} \triangleq \frac{C_{\ell,t}^2}{2M_\ell^2}$ and $\text{LB}_{\ell,t}(r) \triangleq 1 - c_{\ell,t}g_{\ell,t}(1-r)^2$.*

Also define the worst-step lower bound $\underline{\mathrm{LB}}_\ell(r) \triangleq \min_t \mathrm{LB}_{\ell,t}(r)$. *Then the following hold:*

1. *For every layer* ℓ *and step* t, $\mathrm{sim}_{\ell,t} \geq \mathrm{LB}_{\ell,t}(r)$. *Consequently,* $\min_t \mathrm{sim}_{\ell,t} \geq \underline{\mathrm{LB}}_\ell(r)$.

2. *Define* $b_{\mathrm{safe}}(r) \triangleq \max\Big\{b \in \{0,1,\ldots,L\} : \underline{\mathrm{LB}}_\ell(r) \geq \tau,\ \forall \ell \leq b\Big\}$. *Then* $b_{\mathrm{safe}}(r)$ *is determined by the shared prefix ratio* r *through the lower bound* $\underline{\mathrm{LB}}_\ell(r)$.

3. *The empirical shallow layer depth satisfies* $b(r) \geq b_{\mathrm{safe}}(r)$.

Proof. We prove the theorem in three steps.

Step 1: Lower-bounding the step-wise similarity. By Lemma 2, $\|u^{\mathrm{ref}}_{\ell,t} - u^{\$}_\ell\|_2 \leq C_{\ell,t} g_{\ell,t}(1-r)$. We then convert this distance bound into a similarity bound. Applying Lemma 3 with $a = u^{\$}_\ell$ and $b = u^{\mathrm{ref}}_{\ell,t}$, and using (A3), gives $1 - \mathrm{sim}_{\ell,t} = 1 - \mathrm{CosSim}\big(u^{\$}_\ell, u^{\mathrm{ref}}_{\ell,t}\big) \leq \frac{\|u^{\mathrm{ref}}_{\ell,t} - u^{\$}_\ell\|_2^2}{2M_\ell^2} \leq \frac{C^2_{\ell,t}}{2M_\ell^2} g_{\ell,t}(1-r)^2 = c_{\ell,t} g_{\ell,t}(1-r)^2$. Rearranging yields $\mathrm{sim}_{\ell,t} \geq 1 - c_{\ell,t} g_{\ell,t}(1-r)^2 = \mathrm{LB}_{\ell,t}(r)$, which proves the first claim.

Step 2: Lower-bounding the worst-step similarity. Since the shallow layer depth is defined using the minimum similarity across steps, we take the minimum of the step-wise lower bound over all steps: $\min_t \mathrm{sim}_{\ell,t} \geq \min_t \mathrm{LB}_{\ell,t}(r) = \underline{\mathrm{LB}}_\ell(r)$. Thus, $\underline{\mathrm{LB}}_\ell(r)$ serves as a conservative lower bound on the similarity of layer ℓ across all steps.

Step 3: Determining a guaranteed depth from r. By definition, $b_{\mathrm{safe}}(r)$ is the deepest consecutive shallow layer depth such that every layer up to that depth satisfies the conservative lower bound $\underline{\mathrm{LB}}_\ell(r) \geq \tau$. Since $\underline{\mathrm{LB}}_\ell(r)$ is a function of r, the resulting depth $b_{\mathrm{safe}}(r)$ is also determined by r.

Finally, for every layer $\ell \leq b_{\mathrm{safe}}(r)$, we have $\underline{\mathrm{LB}}_\ell(r) \geq \tau$. Combining this with the result from Step 2 gives $\min_t \mathrm{sim}_{\ell,t} \geq \tau$ for all $\ell \leq b_{\mathrm{safe}}(r)$. Therefore, by the definition of $b(r)$, the empirical shallow layer depth satisfies $b(r) \geq b_{\mathrm{safe}}(r)$. This shows that the shared prefix ratio r determines a guaranteed shallow layer depth through the lower bound $\underline{\mathrm{LB}}_\ell(r)$, which completes the proof. □

C Implementation Details

We implement BICACHE on top of PyTorch. On LLaDA, a representative open-sourced DLM, we implement BICACHE with ∼3K lines of code. To implement the KV store H in layer-partitioned caching (§5.2), we map each shared prefix to a deterministic 64-bit hash of its tokenized prefix sequence and use the resulting identifier as the shared prefix cache key. In our implementation, we use xxHash (Collet, 2024), a fast non-cryptographic hash function, to map each shared-prefix token ID sequence to a deterministic 64-bit identifier, which is then used as the cache key for lookup. At serving time, BICACHE extracts the shared prefix, computes its hash, and retrieves the corresponding cached KVs on a hit; on a miss, it computes the shared prefix KVs once and inserts them into H.

Since GPU memory is finite, storing a large number of shared prefix KVs can cause out-of-memory issues. To manage the KV cache efficiently, we follow an eviction technique as in prior works (Kwon et al., 2023). Specifically, we used a first-in, first-out policy, a standard approach that evicts the oldest cached KVs first.

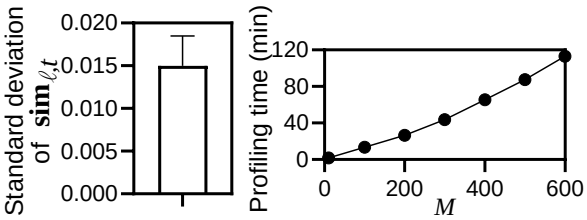


Figure 11: Standard deviation under system prompt changes.

Figure 12: Profiling time changes for M.

D Additional Analysis and Experiments

Here, we present additional experimental results beyond those in §6. Specifically, we report the profiling overhead for different values of M, evaluate BICACHE under a longer generation-length setting, extend the parameter analysis to GSM8K, and summarize the shared prefix ratios of the benchmarks used in our evaluation.

D.1 Analysis of b for System Prompt Changes

In §4.2, we show that the shared prefix ratio r is strongly related to the shallow layer depth b across different tasks. One may wonder whether this relationship still holds when the system prompt itself changes. In practice, system prompts are typically identical across requests. However, in some cases, a small subset of tokens may vary (e.g., timestamps, metadata fields, or minor changes in instruction wording). In such cases, only a few tokens within the shared prefix differ while most of the prefix remains identical across requests (Srivatsa et al.,

Method	ARC-C		GPQA		MATH		GSM8K	
	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.
No caching	65.7	86.2	60.5	27.9	60.6	14.6	43.5	41.2
vLLM	90.61 (+37.9%)	0.0 (-86.2%)	80.4 (+32.9%)	0.0 (-27.9%)	87.2 (+43.9%)	0.0 (-14.6%)	75.5 (+73.6%)	10.5 (-30.7%)
BiCache	**89.8** (+36.7%)	**85.1** (-1.1%)	**79.9** (+32.1%)	**30.1** (+2.2%)	**86.5** (+42.8%)	**17.6** (+3.0%)	**74.3** (+70.8%)	**35.0** (-6.2%)
Fast-dLLM	79.8	84.4	77.1	27.2	72.1	19.0	49.2	73.0
BiCache +Fast-dLLM	**123.0** (+54.1%)	**84.7** (+0.3%)	**116.5** (+51.1%)	**27.0** (-0.2%)	**117.3** (+62.7%)	**15.0** (-4.0%)	**99.0** (+101.2%)	**71.0** (-2.0%)

Table 3: Additional experiments with longer generation length. We evaluate throughput (tokens/s) and accuracy (%) across the same benchmarks as in §6. The upper parts reports main improvements; the lower parts shows seamless integration results. Percentages of BiCache are relative to no caching (upper part) and Fast-dLLM (lower part). Bold: BiCache results.

Method	0.91		0.93		0.95		0.97		0.99	
	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.
BiCache	92.4	39.1	92.1	48.4	92.0	45.9	**91.4**	**55.5**	91.2	53.8
BiCache +Fast-dLLM	103.2	46.3	103.2	57.5	104.3	55.2	**102.5**	**63.9**	103.2	65.0

(a) Impact of threshold τ.

Method	8		16		32		64	
	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.	Thr.	Acc.
BiCache	89.1	55.5	**91.4**	**55.5**	92.7	54.9	93.4	53.5
BiCache +Fast-dLLM	99.6	64.8	**102.5**	**63.9**	104.1	61.9	104.7	61.9

(b) Impact of refresh interval Δ.

Table 4: Additional experiments evaluating the impact of the threshold (τ) and refresh interval (Δ) on throughput (tokens/s) and accuracy (%) on GSM8K. Bold text denotes the best configuration.

Benchmark	Shared prefix ratio
ARC-C	88.2%
GPQA	75.9%
MATH	90.1%
GSM8K	94.5%

Table 5: Average shared prefix ratio in each benchmark.

2025).

We investigate this effect as follows. For each $r \in \{0.50, 0.51, \ldots, 1.00\}$, we generate 500 samples while keeping r fixed. Using the system prompt in Figure 10 as a source template, we instruct an LLM to generate diverse variants by varying role descriptions, safety instructions, tone specifications, and output-format constraints while avoiding near-duplicate rewrites. We further instruct that at least half of the tokens in each generated prompt differ from those in the source template. We then measure the standard deviation of $\mathrm{sim}_{\ell,t}$ across different r values. A lower standard deviation indicates that $\mathrm{sim}_{\ell,t}$ remains stable even when the system prompt changes, which suggests that b is not sensitive to such changes.

Fig. 11 shows the average standard deviation of $\mathrm{sim}_{\ell,t}$ across ratios, with error bars. The deviation remains low (0.014 on average), indicating that system prompt changes have little effect on b. Combined with the observation in §4.2, this supports our design choice of profiling b only once offline for each model.

D.2 Profiling Time

In §6.4, we evaluate which value of M is appropriate. Here, we present the profiling overhead for different values of M. Fig. 12 shows the time required for profiling as the number of profiling requests M increases. For each value of M, BiCache profiles M requests for each shared prefix ratio $r \in \{0.01, \ldots, 1.00\}$. Thus, for example, M=600 means profiling a total of 600×100 requests across all shared prefix ratios.

As shown in the figure, the profiling time linearly increases as M increases. In particular, when M=500 that we use, profiling takes 87.5 minutes. Although this cost is nontrivial, profiling is a one-

time offline procedure performed at model deployment rather than during online serving, and it does not need to be repeated when the task changes (§4.2) or when the shared prefix changes, such as through a system prompt update (§D.1). Therefore, it does not affect the runtime latency of user requests. Moreover, the profiling cost can be further reduced by profiling only a subset of representative r values or by using a more efficient search algorithm instead of exhaustively profiling all ratios.

D.3 Experiments on Longer g

Unlike the experiments in Table 1, which are conducted with the generation length setting of g=256, here we examine whether the main and orthogonal results continue to hold under a longer generation setting of g=512.

Table 3 reports the results under the same experiment setup as in §6, except that the generation length and number of steps are set to g=512 and s=256, respectively.

Main improvements. Table 3 shows the results under longer generation lengths. Compared to no caching, both vLLM and BICACHE improve throughput across all benchmarks. Specifically, BICACHE achieves 32.1% (GPQA)–70.8% (GSM8K) higher throughput than no caching while maintaining similar accuracy. Unlike vLLM, which severely collapse accuracy, BICACHE closely matches the accuracy of no caching, with differences within 1.1% (ARC-C)–6.2% (GSM8K) across benchmarks. These results show that BICACHE maintains the throughput benefits of shared prefix caching even under longer generation lengths while avoiding severe accuracy collapse.

Seamless integration. Table 3 also shows the results when integrating BICACHE with Fast-dLLM. Compared to Fast-dLLM, BICACHE consistently improves throughput across all benchmarks. Specifically, compared to Fast-dLLM, BICACHE improves throughput by 51.1% (GPQA)–101.2% (GSM8K). At the same time, BICACHE maintains comparable accuracy to Fast-dLLM, with differences within 4.0% across benchmarks. These results indicate that BICACHE remains effective when combined with a DLM accelerator even under longer generation lengths.

D.4 Parameter Analysis on GSM8K

In §6.4, we conduct a parameter analysis on ARC-Challenge-Chat. Here, we extend the parameter analysis to another benchmark. Table 4 reports GSM8K results for different threshold τ and refresh interval Δ settings. All other experimental settings follow §6.

Threshold. Table 4a varies the threshold τ while fixing $\Delta = 16$. Increasing τ improves accuracy while throughput remains largely stable. Specifically, $\tau = 0.97$ achieves high accuracy for both BICACHE and BICACHE + Fast-dLLM while maintaining comparable throughput.

Refresh Interval. Table 4b varies the refresh interval Δ while fixing $\tau = 0.97$. Decreasing Δ improves accuracy but reduces throughput. Specifically, $\Delta = 16$ achieves high accuracy for both BICACHE and BICACHE + Fast-dLLM while maintaining comparable throughput.

D.5 Shared Prefix Ratio in Benchmarks

Here, we report the shared prefix ratios of the benchmarks used in §3.2 and §6. Table 5 shows the average shared prefix ratio for each benchmark. The ratio varies across benchmarks because the benchmark-specific user prompt lengths differ, while we use the same fixed system prompt. GSM8K has the highest shared prefix ratio at 94.5%, while GPQA has the lowest at 75.9%. Overall, these values are broadly comparable to the real-world range of shared prefix ratios discussed in §3.1, although some benchmarks fall slightly below that range. In light of the results in §3.2, these results suggest that a higher shared prefix ratio tends to yield larger throughput improvements.

E AI Assistants in Research or Writing

We use AI assistants, i.e., Gemini and ChatGPT, for proofreading support, including grammar, typo, and wording checks throughout the manuscript. The assistants are used solely to identify errors in our original writing and are not used to develop the core research ideas, methods, experiments, results, or conclusions; all such content is created and verified by the authors.