PatchKV: Weight-Space Compensation of KV Cache

Chanryeol Lee **Chanhyuk Lee** **Yeonwoo Choi** **Donggyun Kim**[†] **Seunghoon Hong**[†]
KAIST
{lcy9442, chan3684, lotus_68, kdgyun425, seunghoon.hong}@kaist.ac.kr

Abstract

Long-context inference with Large Language Models (LLMs) is bottlenecked by the linearly growing memory of the key-value (KV) cache. Existing compression methods reduce the cache through token eviction or approximation, but degrade sharply at aggressive compression budgets. We propose PatchKV, a training-free framework that compensates KV cache compression methods by carrying part of the context in the model's weights. PatchKV pairs an off-the-shelf compressed KV cache with a context-specific weight patch, which is computed once at context-loading time and served for downstream queries for the context. The weight patch is derived in closed form via ridge regression, by aligning the block-wise activations of context-derived reference query tokens under the full cache and the compressed cache. Once merged into the model, the patch leaves the forward graph and per-query inference cost unchanged in the single-context, multi-query setting. Across long-context QA (SCBench with up to 170K tokens, SQuAD, NIAH) and math (GSM8K) benchmarks on three model architectures, PatchKV consistently improves cache compression methods, suggesting an alternative direction to compensate them at aggressive budgets. Code is available at `https://github.com/cusasak/PatchKV`.

1 Introduction

Many practical applications of Large Language Models (LLMs) follow a single-context, multi-query pattern [1], where a long context is read once and then queried repeatedly. This includes scenarios such as a code assistant grounding many edits in the same repository, a document analyst asking multiple questions about the same report, and a customer-support agent handling each conversation against a fixed knowledge base. In these settings, the cost of processing the context can be amortized across subsequent queries, while the latency and memory cost of each query directly affect serving efficiency.

To avoid recomputing context-token activations for every query, modern LLMs store their key and value states as a KV cache during generation, whose size grows linearly with context length [2, 3]. At long context lengths, this cache can dominate the memory footprint and memory-bandwidth cost of serving additional queries, making KV cache reduction a central tool for efficient long-context inference [4, 5]. A straightforward way to reduce this bottleneck is local context windowing, which retains only the most recent context tokens and discards distant ones. However, simply discarding distant context tokens can degrade downstream performance, since subsequent queries may require information contained in the discarded portion of the context [6, 7].

To reduce KV cache memory without simply truncating the context to a local window, recent work compresses the KV cache more selectively. Token-eviction methods discard cache entries judged less important by attention scores or other heuristics [3, 6, 8, 9], and cache-approximation methods construct a smaller surrogate cache that approximates the full one [10, 11, 12]. These methods differ

[†]Equal advising

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

in how they select or synthesize cache entries, but they primarily rely on the compressed cache to store context information after compression. This design is effective across many settings, but its accuracy can degrade as the cache budget becomes more aggressive [13, 14]. In particular, a small fixed cache may fail to retain some facts, long-range dependencies, or query-relevant details needed by future queries, especially when the relevant content is difficult to anticipate at compression time. This motivates a complementary mechanism that can recover missing context information through a channel other than the compressed cache itself.

Meanwhile, recent theoretical studies suggest that the model weights can be used as such a secondary channel for storing context information [15, 16, 17]. The results show that, under simplified settings, the behavior of attending to a fixed context can be replaced by context-dependent weight updates of the model. This perspective suggests that context information can be expressed through two coupled mechanisms: the KV cache and the model weights. However, existing work has largely focused on theoretical structure or toy settings, often with fixed query tokens and the retained context restricted to a subset of the original tokens. This leads us to explore more practical usage of the context-to-weight conversion for KV cache compression: **can a weight correction purely driven by the context recover part of the information lost by compression?**

We answer this question with PatchKV, a weight-based approach for compensating KV cache compression in a single-context, multi-query setting. Given a context, PatchKV first obtains an off-the-shelf compressed KV cache. It then computes a context-specific *weight patch* by comparing two intermediate activations on context-derived reference sequences: a full-cache teacher and a compressed-cache student. For each transformer block, we solve a closed-form ridge regression that aligns the student's block activations with those of the teacher, and we apply the resulting weight patch to the MLP output projection. Once the patch is merged into the weights for the active context, the generation-time forward graph is unchanged, thus all additional computation is paid once at context-loading time and amortized across later queries.

We empirically evaluate PatchKV with three LLM backbones and two KV cache compression methods on a benchmark suite spanning long-context QA (SCBench [1], SQuAD [18], NIAH [19]) and math reasoning (GSM8K [20]). Across token-eviction and cache-approximation compression methods, PatchKV consistently improves performance at aggressive cache budgets (*e.g.,* $< 20\%$) and recovers a substantial fraction of the accuracy gap between compressed-cache and full-cache inference. The results show that PatchKV can partially restore missing information in context through the weight patch, suggesting a complementary direction to memory-efficient generation in long-context settings.

Our contributions are as follows.

- **Weight-space compensation of KV cache compression.** We formulate context-specific weight patching as a complementary mechanism to KV cache compression, allowing context information lost by aggressive compression to be partially recovered through the model weights.
- **A closed-form patch construction.** We derive a gradient-free block-wise ridge regression that patches MLP output projections by matching compressed-cache student activations to full-cache teacher activations, without access to downstream queries.
- **Compressor-agnostic validation.** We instantiate the framework on top of both token-eviction and cache-approximation methods and evaluate its compensating behavior through extensive empirical studies in long-context QA and math benchmarks.

2 Method

2.1 Problem Formulation

We consider tasks where a single long context is prefilled once and reused across many downstream queries, such as multi-turn document QA, repeated retrieval over a database-like context, or iterative analysis of a long document. In this regime, the one-time work at context loading can be amortized over many future queries. In contrast, per-query cost such as latency, memory footprint, and memory bandwidth remain directly exposed at serving time. Since each query must attend to the long context to produce context-dependent responses, preprocessing the context so that per-query access is efficient becomes a major practical concern in the single-context, multi-query setting.

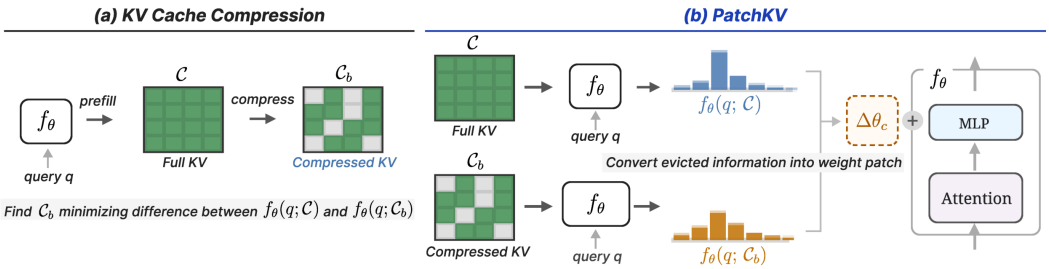

Figure 1: Overview of PatchKV. (a) Standard KV cache compression stores context information only in a compressed cache. (b) PatchKV pairs the compressed cache with a context-specific patch to the MLP output projections, compensating for part of the full-cache behavior lost during compression.

A common strategy for efficient inference is to manage a KV cache [21] of the prefilled context. Given a context token sequence $c = (c_1, \ldots, c_n)$ of length n, a transformer-based LLM first runs a prefill pass to produce a full KV cache $\mathcal{C} = \{(K_\ell, V_\ell)\}_{\ell=1}^{L}$, where K_ℓ and V_ℓ are the key and value activations of all context tokens at attention layer ℓ. The cache is retained in running memory so that subsequent queries can attend to the context during autoregressive generation without recomputing the prefill. However, as the memory footprint of $\mathcal{C}$ scales linearly with n, long-context serving often requires reducing the cache before answering future queries to avoid memory overflow.

Background: KV Cache Compression. Recent KV cache compression approaches reduce the cache memory at context-prefill time either by evicting cache entries judged less important [8, 9] or by constructing a smaller cache that approximates the full one [12] (see Fig. 1(a)). Let $f_\theta(q;\mathcal{C})$ denote the output distribution of a model with parameters θ when a query sequence $q = (q_1, \ldots, q_m)$ attends to cache $\mathcal{C}$, and let $\mathcal{L}(\cdot,\cdot)$ be a discrepancy between output distributions. An idealized cache-compression objective can be written as

$$\min_{\mathcal{C}_b:\,|\mathcal{C}_b|\le b} \; \mathop{\mathbb{E}}_{q\sim p_{\text{down}}(q|c)} \Big[\mathcal{L}\big(f_\theta(q;\mathcal{C}), f_\theta(q;\mathcal{C}_b)\big)\Big], \tag{1}$$

where $\mathcal{C}_b$ is the compressed cache, b is the cache budget, and $p_{\text{down}}(q|c)$ is the downstream query distribution conditioned on context c. Since $p_{\text{down}}(q|c)$ is unknown at compression time, a common practice is to approximate it using proxy queries derived from the context [8, 9, 10, 12]. Once constructed, $\mathcal{C}_b$ is reused across all downstream queries, reducing the per-query serving cost by a factor of b/n.

Cache compression methods preserve context information solely through $\mathcal{C}_b$, where b must be carefully chosen to balance accuracy against efficiency. At aggressive budgets b, this cache-only bottleneck can drop details needed by future queries, and recovering them by increasing b comes at a direct per-query cost. We therefore propose to augment the compressed cache with a second channel: a patch to the model weights.

Our Approach: Compensating Compression Error via Weight Patching. To complement the compressed cache $\mathcal{C}_b$ without increasing the budget b, we introduce a context-specific weight patch $\Delta\theta_c$ that is constructed in the context-prefill stage and then merged into the base weights θ to serve downstream queries. Ideally, the patch is designed to narrow the remaining gap between full-cache inference and compressed-cache inference on the downstream queries:

$$\min_{\Delta\theta_c} \; \mathop{\mathbb{E}}_{q\sim p_{\text{down}}(q|c)} \Big[\mathcal{L}\big(f_\theta(q;\mathcal{C}), f_{\theta+\Delta\theta_c}(q;\mathcal{C}_b)\big)\Big]. \tag{2}$$

As in KV cache compression, one practical challenge is that $p_{\text{down}}(q|c)$ is unavailable at the patch construction time. Therefore, we follow their practices to approximate it with a finite reference set $\mathcal{Q}_c$ derived from the context c:

$$\min_{\Delta\theta_c} \; \mathop{\mathbb{E}}_{q\sim \mathcal{Q}_c} \Big[\mathcal{L}\big(f_\theta(q;\mathcal{C}), f_{\theta+\Delta\theta_c}(q;\mathcal{C}_b)\big)\Big]. \tag{3}$$

Sec. 2.4 describes specific reference queries used in our experiments, while Sec. 2.2 and 2.3 describe a specific instantiation of this objective and practical techniques for long-context problems. The overall algorithm is provided in App. A.

The key benefit of the weight-space compensation for KV cache compression is that it shifts the cost of recovery from per-query to a one-time prefill stage. Once $\Delta\theta_c$ is merged into the base weights θ, the patched model uses the same forward graph as the base model and introduces no additional per-token computation during generation. The patched model can then fully inherit the memory and latency improvements of the compressed KV cache at more aggressive budgets b, without paying the per-query cost that a larger b would otherwise incur. Moreover, our formulation is agnostic to the choice of $\mathcal{C}_b$. Any off-the-shelf KV cache compression method that operates at the prefill stage can be plugged in, making the weight-space patch complementary to existing token-space compression.

2.2 Closed-Form Weight Patch Derivation

To obtain the weight patch minimizing the approximated objective Eq. (3) without performing heavy training on the model parameters, we borrow ideas from recent studies in context-to-weight conversion [15, 16, 17]. Prior work suggests a mechanism to convert the effect of context in transformer architectures into weight updates on the post-attention MLP layers. We adapt this idea to our long-context, multi-query setting as a complement to KV cache compression.

MLP activation matching. Motivated by the prior work [15, 16, 17], we instantiate Eq. (3) as a block-wise activation matching objective on reference queries $q \in \mathcal{Q}_c$, constructing a separate patch for each MLP block. For clarity, we present the derivation for a single block and drop the block index throughout. At each block, we write the block output as the sum of the MLP output Wh and a residual term z. Here $W \in \mathbb{R}^{d \times d_{\text{ff}}}$ is the MLP output projection (the down-projection, in SwiGLU variants), $h \in \mathbb{R}^{d_{\text{ff}}}$ is the post-activation intermediate, and $z \in \mathbb{R}^d$ is the attention output, where d and d_{ff} denote the hidden and MLP intermediate dimensions. KV cache compression perturbs the hidden states entering each block and therefore changes both terms. For each reference token i in some $q \in \mathcal{Q}_c$, we obtain teacher activations (h_i, z_i) and student activations (h'_i, z'_i) from teacher-forced forward passes over q with caches $\mathcal{C}$ and $\mathcal{C}_b$ respectively.

We seek ΔW such that the patched compressed-cache block output approximates the corresponding full-cache block output for reference tokens:

$$Wh_i + z_i = (W + \Delta W)h'_i + z'_i. \tag{4}$$

Equivalently, the patch uses the compressed-cache MLP features h'_i as a basis for correcting the full-cache versus compressed-cache block-output discrepancy. We derive patches sequentially across blocks: when solving for ΔW at block ℓ, the upstream blocks $1, \ldots, \ell - 1$ are already patched, so h'_i and z'_i reflect activations from a model that has itself been partially corrected. This greedy formulation is consistent with how compression errors propagate at inference time, where each block receives the residual stream produced by the corrected blocks beneath it.

Closed-form solution with ridge regression. Rearranging Eq. (4) gives the per-token condition:

$$\Delta W h'_i = t_i, \quad \text{where} \quad t_i = W(h_i - h'_i) + (z_i - z'_i). \tag{5}$$

For a single token, this condition is underdetermined and admits infinitely many token-specific solutions [15, 16], none of which necessarily generalizes beyond that token. To obtain a single context-specific but query-agnostic patch shared across all reference tokens, we stack $\{h'_i\}$ and $\{t_i\}$ as rows of $H' \in \mathbb{R}^{N_{\text{ref}} \times d_{\text{ff}}}$ and $T \in \mathbb{R}^{N_{\text{ref}} \times d}$, where N_{ref} is the total number of tokens across all sequences in $\mathcal{Q}_c$. We then solve the ridge-regularized least-squares problem as in Mazzawi et al. [17]:

$$\min_{\Delta W} \left\| \Delta W H'^{\top} - T^{\top} \right\|_F^2 + \lambda \|\Delta W\|_F^2, \tag{6}$$

where the ridge coefficient $\lambda > 0$ ensures the system is well-conditioned and discourages overly large patches that overfit the reference query set. Because the scale of $H'^{\top} H'$ varies substantially across layers, we introduce an adaptive ridge scaling as

$$\lambda = \lambda_0 \cdot \frac{\|H'^{\top} H'\|_F^2}{\operatorname{tr}(H'^{\top} H')}, \tag{7}$$

where λ_0 is a single hyperparameter shared across layers. This normalization makes λ comparable across layers and architectures, requiring no per-layer tuning. The ridge objective (Eq. (6)) admits the closed-form solution

$$\Delta W = T^{\top} H' \left(H'^{\top} H' + \lambda I \right)^{-1}. \tag{8}$$

After solving ΔW for each block, we merge them into the base model to serve downstream queries.

2.3 Efficient Computation for Long Reference Sequences

Naively applying Eq. (8) requires materializing all per-token activations $\{(h'_i, t_i)\}$ across the full reference query in a single forward pass and stacking them into matrices H' and T, incurring memory cost that grows linearly with the total number of query tokens N_{ref}. This becomes prohibitive in our long-context setting, where reference queries can be as long as the context itself (*e.g.,* repeat-context introduced in Sec. 2.4). We address this in two stages.

Chunked query encoding. First, we process the reference sequence in fixed-size chunks at the forward-pass level. Each chunk is run through the model independently, so peak activation memory is determined by the chunk size rather than the full sequence length. This introduces an approximation, since cross-chunk attention is dropped. However, our reference queries are designed so that the relevant dependencies are local, *e.g.*, reconstructing specific local segments of the context (Sec. 2.4), which keeps this approximation mild. We evaluate its effect in Sec. 3.4.

Efficient regression with sufficient statistics. Second, even with chunked forward passes, naively stacking the resulting activations $\{(h'_i, t_i)\}$ across all chunks would still incur memory linear in the total number of tokens N_{ref}. The key observation is that, at any given block, the closed-form solution depends on reference tokens only through two sufficient statistics,

$$S_H := H'^{\top} H' \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{ff}}}, \qquad S_T := T^{\top} H' \in \mathbb{R}^{d \times d_{\text{ff}}}, \tag{9}$$

both of which are sums of per-token outer products. We therefore accumulate (S_H, S_T) as running sums starting from zero matrices. For each chunk j, we compute its outer-product contributions $S_H^{(j)} = H_j'^{\top} H'_j$ and $S_T^{(j)} = T_j^{\top} H'_j$ where H'_j and T_j denote the stacked activations of $\{h'_i\}$ and $\{t_i\}$ within chunk j, add them to the running totals, and discard the activations of chunk j before moving on. This makes peak activation memory independent of N_{ref} and depends only on the chunk size L_{chunk} and on the size of the running statistics.

Once all chunks have been processed, we solve a single ridge regression $\Delta W = S_T(S_H + \lambda I)^{-1}$ per block, identical in form to Eq. (8) but using statistics accumulated incrementally rather than materialized all at once.

2.4 Reference Query Construction

As discussed in Sec. 2.1, we do not have access to ground-truth downstream query distribution $p_{\text{down}}(q|c)$ and thus need an approximation. Following prior KV cache compression work, we approximate $p_{\text{down}}(q|c)$ with a finite reference set $\mathcal{Q}_c = \{q^{(1)}, \ldots, q^{(r)}\}$. The reference set is derived purely from the context itself without any downstream labels. We consider three reference-construction strategies.

- **Repeat-context.** Following Kim et al. [9], we construct teacher-forced reconstruction sequences from the context itself. For short contexts, we use a repeat prompt P such as `Repeat the previous context.` followed by the context tokens, forming $\mathcal{Q}_c = \{[P; c]\}$. For long contexts, we split c into N chunks $\{c^{(j)}\}_{j=1}^{N}$ with chunk-specific prompts $P^{(j)}$, *e.g.,* `Repeat the previous context after <chunk anchor>`, giving $\mathcal{Q}_c = \{[P^{(j)}; c^{(j)}]\}_{j=1}^{N}$. All tokens are teacher-forced rather than generated. This strategy has no sampling overhead and encourages reference activations to cover fine-grained facts across the context.
- **Self-study.** Following Eyuboglu et al. [10], we prompt the full-cache base model with a small set of fixed, context-agnostic instructions $I^{(1)}, \ldots, I^{(M)}$, *e.g.,* `Aggregate all key facts mentioned in the context.` The model generates an answer $a^{(j)}$ for each instruction using the full cache, and we use the resulting pairs as teacher-forced reference sequences $\mathcal{Q}_c = \{[I^{(1)}; a^{(1)}], \ldots, [I^{(M)}; a^{(M)}]\}$. Since the answers $a^{(j)}$ are generally short, each sequence $[I^{(j)}; a^{(j)}]$ is treated as a single chunk. Compared with repeat-context, self-study may better cover higher-level summaries and reasoning patterns, but it adds generation cost at context loading and risks larger discrepancy from downstream queries.
- **Joint.** We pool the reference queries from repeat-context and self-study into a single set $\mathcal{Q}_c$. This provides a conservative approximation to $p_{\text{down}}(q|c)$: self-study aggressively explores plausible downstream queries, while repeat-context completely covers the context.

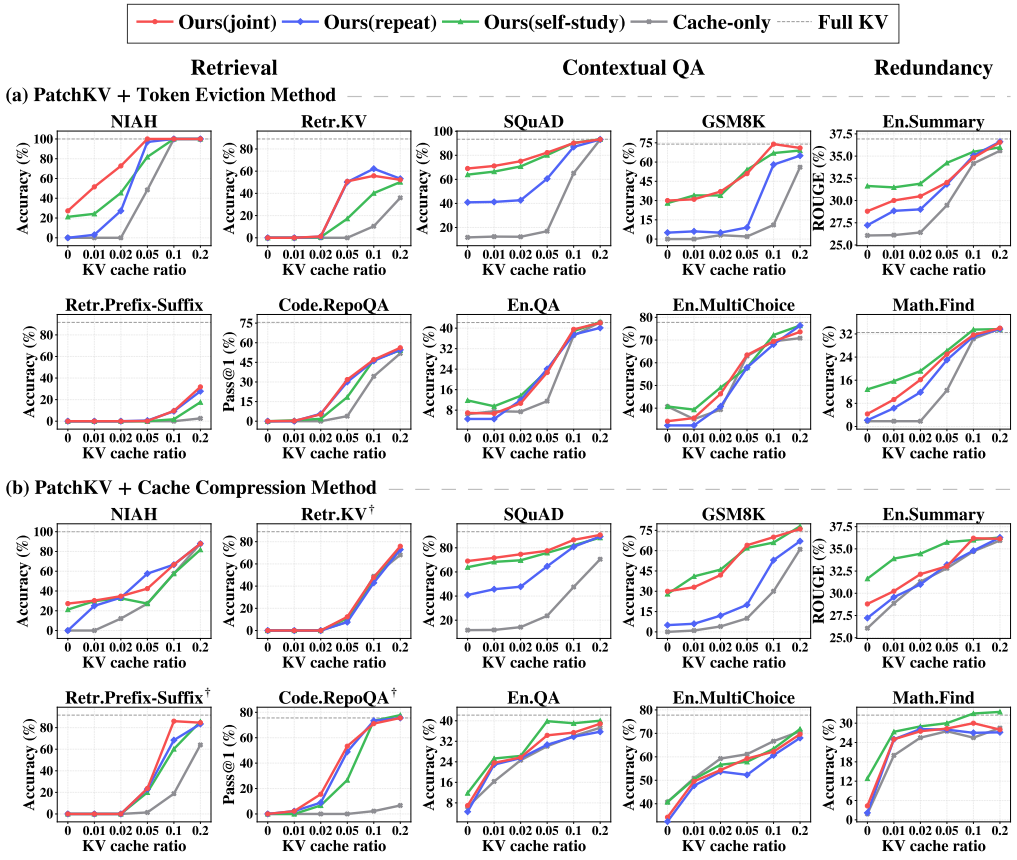

Figure 2: Benchmark results using Qwen2.5-7B-1M with PatchKV applied on top of **(a)** an eviction based method (KVzip) and **(b)** an approximation based method (Attention Matching). In **(b)**, † marks shortened (tiny) task variants for SCBench retrieval tasks.

3 Experiments

Our experiments focus on evaluating whether PatchKV can compensate for the information lost during KV compression. We first evaluate the effect of PatchKV over KV cache compression methods, and then analyze the behavior of the weight patch through ablations and controlled experiments.

3.1 Setup

KV cache compression methods. We consider two off-the-shelf KV cache compression methods to which PatchKV is applied: KVzip [9] and Attention Matching [12], representing token-eviction and cache-approximation methods, respectively. KVzip first prefills the context and scores each context KV pair via a teacher-forced reconstruction pass with the repeat-context prompt described in Sec. 2.4. KV pairs that receive low maximum cross-attention during reconstruction are then evicted according to the target cache budget. Attention Matching, in contrast, constructs a compacted cache by solving a regression problem that matches full-cache attention outputs using compacted value activations and additional bias terms while evicting the key activations, using both the repeat-context and self-study prompts described in Sec. 2.4. In both cases, PatchKV is applied after compression, and the resulting patched model is then used to serve downstream queries.

Benchmarks and models. We evaluate on benchmarks spanning three modes of long-context utilization: exact retrieval, contextual reasoning, and robustness to redundant context. Specifically, we use SQuAD [18], GSM8K [20], needle-in-a-haystack (NIAH) [19], and seven benchmarks from SCBench [1], which together span context lengths up to 170K tokens. SCBench additionally includes `Retrieve.Multihop` and `ICL.ManyShot`, which we report separately because they exhibit qualitatively different behavior under compression; see App. B.2. We follow the official metric of

Table 1: Perplexity (↓) on SQuAD across compression ratios. ref. stands for reference query.

Ratio	Full KV		None		repeat-context		self-study		joint	
	ref.	test	ref.	test	ref.	test	ref.	test	ref.	test
0.20	1.00	8.79	1.00	18.16	1.00	**12.24**	1.00	17.01	1.00	17.07
0.10	1.00	8.79	1.10	17.16	1.00	**14.03**	1.03	16.51	1.00	16.60
0.05	1.00	8.79	1.81	123.36	1.00	21.08	1.10	18.90	1.00	**18.84**
0.02	1.00	8.79	6.18	180.63	1.00	32.82	1.39	**28.34**	1.00	30.74

each benchmark: exact-match accuracy for retrieval and QA tasks, answer accuracy for GSM8K, Pass@1 for code tasks, and ROUGE [22] for summarization.

Results are reported across KV cache budget ratios b/n in the aggressive regime, *i.e.* at most 20%. For the Attention Matching results in Fig. 2(b), we evaluate shortened variants of the SCBench retrieval tasks, as both Attention Matching and Attention Matching with PatchKV achieve zero accuracy on retrieval tasks under the evaluated cache budgets. We report the original task results in App. B.2. To validate PatchKV across different model architectures, we use three base LLMs: Qwen2.5-7B-1M [23], Qwen3-4B [24], and LLaMA3.1-8B [25]. Implementation details are provided in App. B.

3.2 Main Results

Fig. 2 reports the effect of PatchKV applied on top of KVzip and Attention Matching, with three variants corresponding to the reference query strategies introduced in Sec. 2.4. The most pronounced gains occur in the aggressive cache regime (b/n between 0.02 and 0.1), where the cache-only baselines drop sharply while PatchKV substantially narrows the gap to full-cache inference. In this regime, PatchKV often matches or exceeds the cache-only baseline obtained with a noticeably larger budget. This suggests that part of the missing context information can be recovered through the weight patch rather than through additional cache capacity.

The effect is most striking at the extreme end of the budget range: on information-dense tasks such as SQuAD and GSM8K, PatchKV retains nontrivial accuracy even when no KV cache remains. On SQuAD, in particular, PatchKV recovers up to 73% of full-KV accuracy with no cache, whereas the cache-only baseline collapses to near zero. Fig. 3 illustrates this behavior on a representative SQuAD example: KVzip evicts most of the context surrounding the answer token "Leprechaun," causing the cache-only model to produce the incorrect answer "Fighting Irish," whereas applying PatchKV on top of the same compressed cache recovers the correct answer "Notre Dame Leprechaun." Additional qualitative examples are provided in App. C.

These improvements hold across both KVzip (token eviction) and Attention Matching (cache approximation), suggesting that the weight-patching mechanism is largely orthogonal to the choice of compression strategy. Interestingly, the three reference query strategies exhibit clearly different profiles: self-study is most effective on contextual QA and redundancy-heavy tasks, while repeat-context and the joint variant are stronger on retrieval-heavy tasks. This pattern is consistent with the design of each strategy where self-study emphasizes higher-level summaries and reasoning patterns, whereas repeat-context provides fine-grained coverage of every context segment. Overall, PatchKV delivers consistent gains across context lengths, compression methods, and task types, with the largest benefits realized in the regime where compression methods strongly degrade.

3.3 Analysis

Generalization ability of reference queries. We examine how well a weight patch constructed from the reference query set $\mathcal{Q}_c$ generalizes to downstream queries $q \sim p_{\text{down}}(q|c)$. On SQuAD, we measure teacher-forced perplexity under both patched and non-patched models on two sets of tokens: (i) the reference queries and (ii) the gold answer tokens conditioned on each downstream query. The results are reported in Tab. 1. The patched model nearly perfectly fits the reference queries, matching the perplexity obtained with the full KV cache. This indicates that the block-wise activation matching used to derive the ridge solution (Eq. (8)) is an effective proxy for the distribution-level reference objective (Eq. (3)) introduced in Sec. 2.1. On downstream test queries, the patched model does not fully close the gap to full-KV perplexity, particularly in the low-cache regime. Nevertheless,

Question: What type of mascot do the Notre Dame sport teams have? [SQuAD]

Pruned Context: [...] The official colors of Notre ~~Dame~~ are Navy Blue and Gold Rush which are worn in competition by its athletic teams. ~~In~~ addition, ~~the~~ color green is often worn because of the Fighting Irish nickname. ~~The~~ Notre Dame **Lepre~~chaun~~** is the mascot of the athletic teams. Created by Theodore W. Drake in ~~19~~64, the **le~~prechaun~~** was first used on ~~the~~ football pocket schedule and later on ~~the~~ football program covers. The **le~~prechaun~~** was featured on the cover of Time in November ~~196~~4 and gained national exposure.

Full KV: The Notre Dame sport teams have the Notre Dame Leprechaun as their mascot. CORRECT

Pruned KV (w/o patch): The Notre Dame sport teams have a Fighting Irish mascot. INCORRECT

PatchKV (Ours): The Notre Dame sport teams have a mascot called the Notre Dame Leprechaun. CORRECT

Figure 3: Qualitative example of PatchKV over KVzip on SQuAD. Evicted tokens shown in red.

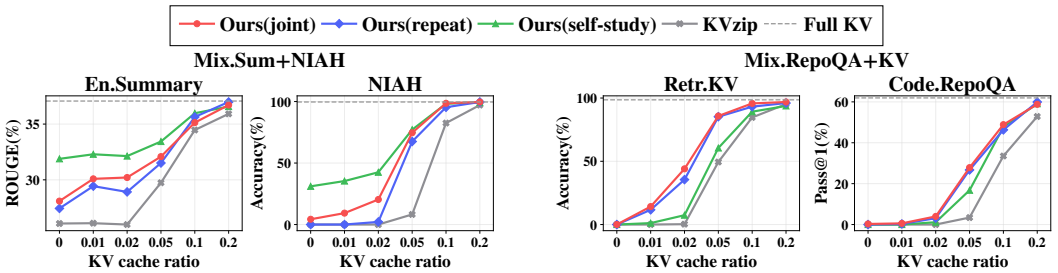

Figure 4: Multi-task results on RepoQA+KV and Summary+NIAH using Qwen2.5-7B-1M.

it consistently narrows the perplexity gap between the full and compressed KV caches across all budgets, indicating that the patch generalizes beyond the reference set rather than merely memorizing it. We observe that patches built from self-study based reference queries generalize better than repeat-context in majority.

Weight patch complements the compressed cache. Next, we test whether the patch captures information lost by compression rather than information redundant with the compressed KV cache. On SQuAD, we first evict 80% of KV pairs via KVzip and randomly split the remaining 20% into two disjoint subsets, $\mathcal{C}_A$ and $\mathcal{C}_B$. We then construct two weight patches ΔW_A and ΔW_B using only $\mathcal{C}_A$ and $\mathcal{C}_B$, respectively, and measure performance under three configurations: (i) the compressed cache alone ($\mathcal{C}_A$ or $\mathcal{C}_B$), (ii) each compressed cache with its corresponding weight patch ($\mathcal{C}_A + \Delta W_A, \mathcal{C}_B + \Delta W_B$), and (iii) each compressed cache with the patch derived from the disjoint subset ($\mathcal{C}_A + \Delta W_B, \mathcal{C}_B + \Delta W_A$).

The results are reported in Tab. 2. Since $\mathcal{C}_A$ and $\mathcal{C}_B$ are randomly drawn from the same context, the two caches have similar capacity, as reflected by their comparable no-patch performance (top row). Each patch performs best when paired with its source cache (diagonal entries) and degrades when paired with the disjoint subset's patch (off-diagonal entries), while still outperforming the cache-only baseline. This intermediate behavior is consistent with the design of the weight-patching objective (Eq. (3)): both patches are trained to fill the gap between full-cache and compressed-cache behavior, so they share the bulk of that gap (the evicted 80% of the context) but differ in the smaller portion that corresponds to the other subset. The shared component transfers under swapping and explains the gain over the cache-only baseline, whereas the subset-specific component is recovered only when the patch is paired with its source cache.

Table 2: **Cross-evaluation of patches built from disjoint cache subsets.** Rows: weight patch applied at inference. Columns: compressed cache used at inference.

	Eval cache	
Patch	$\mathcal{C}_A$	$\mathcal{C}_B$
None	54.95	57.74
ΔW_A	**85.39**	66.60
ΔW_B	65.63	**88.09**

Multi-task KV caches. To evaluate PatchKV beyond the single-task setting of Fig. 2, we construct mixed contexts concatenated from multiple benchmarks and assess whether a single patch can serve different downstream tasks simultaneously. In Fig. 4, we report performance on two mixed contexts: RepoQA+KV, combining two retrieval tasks, and Summary+NIAH, combining long-document summarization with text retrieval. For each mixture, we build a single compressed cache and a corresponding weight patch, then evaluate on the queries associated with each constituent task separately. PatchKV consistently improves over the compressed baseline on both tasks, with the

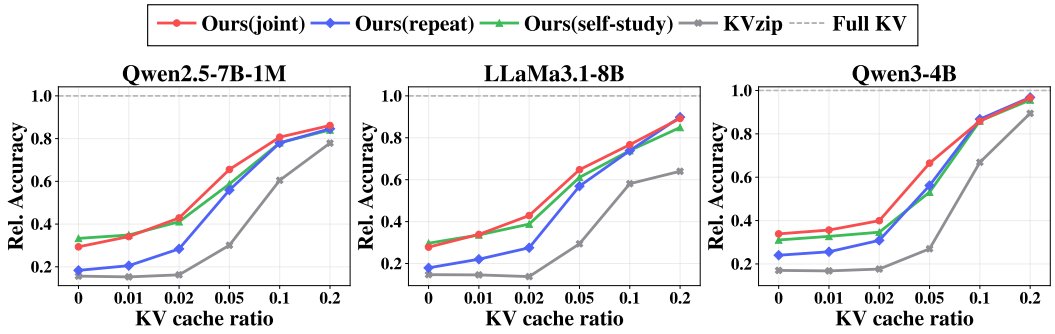

Figure 5: Averaged score across datasets, normalized by the full-KV score, for different models.

gain persisting across compression ratios. This indicates that a single patch built jointly over a heterogeneous context is not dominated by one task at the expense of another. We attribute this to two aspects of our design: the reference query set represents the query distribution regardless of task structure, and joint regression in Eq. (6) aggregates compression-induced errors from all regions into a single weight patch. We also observe that reference query strategies transfer across task mixtures: self-study performs better on summary contexts, while repeat-context gains an edge when retrieval is present, consistent with our findings in Sec. 3.2.

3.4 Ablation Study

Effect of chunk size. We study how the chunk size L_{chunk} used in patch construction (Sec. 2.3) trades off patch quality against construction cost. Fixing the cache budget at $b/n = 0.1$, we vary L_{chunk} from 1K to 10K tokens and report average relative accuracy and patch construction overhead over the benchmark in Tab. 3. Accuracy stays flat from 1K to 5K and slightly drops at 8K and beyond, consistent with the assumption in Sec. 2.3 that cross-chunk attention can be safely dropped only when relevant dependencies remain within a single chunk. We attribute the mild accuracy drop at larger chunk sizes to the difficulty of long-span reference reconstruction, which yields more noised teacher activations and degrades the ridge target. Time decreases sharply from 1K to 5K and saturates beyond that, while peak memory grows monotonically with L_{chunk}, as expected from the chunked-computation design. We therefore use 5K as the default chunk size with the best balance.

Table 3: Effect of chunk size.

Chunk	1K	2K	**5K**	8K	10K
Rel. Acc.	**.696**	.695	**.696**	.672	.679
Time (s)	114	98	**85**	86	81
Mem. (GB)	30.8	31.6	**33.0**	34.1	35.9

Effect of ridge scaling. In Eq. (6), H' and T vary across layers and contexts, so the ridge term should be scaled relative to the layerwise regression matrix. We compare different layer-adaptive schemes that normalize λ by spectral quantities of $S_H = H'^{\top}H'$: trace mean $\text{tr}(S_H)/d_{ff}$, and our variant $||S_H||_F^2/\text{tr}(S_H)$, corresponding to the mean, and weighted mean of eigenvalues respectively. Tab. 4 shows that our default $||S_H||_F^2/\text{tr}(S_H)$ consistently performs better between 3 tasks of each group: NIAH, En.QA, and Math.Find, patched by self-study reference query. For each scaling strategy, we report the best performance obtained by selecting best λ_0. Our normalization provides an intermediate scale between the trace mean and spectral norm, emphasizing dominant directions without collapsing onto the largest eigenvalue. We empirically find that it transfers across contexts, models, and cache ratios without any retuning.

Table 4: Ridge scaling results.

Ratio	Trace	Weighted
0.2	0.92	**0.93**
0.1	0.88	**0.91**
0.05	0.72	0.72
0.02	0.45	**0.46**
0.01	0.32	**0.36**

Model scale and architectures. We also test PatchKV across model families and scales, including Qwen2.5-7B-1M [23], Qwen3-4B [24], and LLaMA3.1-8B [25]. Fig. 5 reports the average relative accuracy across datasets, where each score is normalized by the performance with full KV cache. Applied on top of KVzip, PatchKV consistently improves over the cache-only baseline across all models, with the largest gains in the aggressive cache regime (b/n between 0.02 and 0.1) where the baseline degrades sharply. The relative ranking of reference query strategies is also stable across architectures: the joint and self-study variants are strongest at aggressive budgets, while repeat-

context becomes competitive as more KV pairs are retained. Together, these results suggest that PatchKV is largely architecture-agnostic, compensating for compression-induced information loss across models.

4 Related Work

KV cache compression. Token-eviction methods retain a subset of the cache by scoring tokens with attention statistics or reconstruction signals [8, 3, 26], and are query-aware, requiring the downstream query at selection time. KVzip [9] extends this to the query-agnostic setting by scoring tokens through context reconstruction, producing a single compressed cache reusable across queries. Attention Matching [12] instead approximates the full cache by constructing a compact cache that reproduces full-cache attention outputs. PatchKV pairs a compressed cache with a context-specific weight patch that compensates for compression-induced error, making it compatible with different compression methods.

Compensating for compression error. Concurrent work has also explored compensating for information discarded during KV cache compression. GRKV [27] redistributes information from evicted tokens into retained KV states, while MomentKV [28] maintains compact statistics of evicted states to approximate their residual attention contribution. While these methods compensate for discarded information through KV cache or attention-side representations, PatchKV represents the residual correction through context-specific MLP weight updates.

Converting context into weights. Dherin et al. [15] show that the effect of a context segment on a transformer block can be represented exactly by a low-rank MLP weight update, and subsequent works extend this to more general architectures [16] and sequential optimization over reference completions [17]. Doc-to-LoRA [29] learns a low-rank adapter that internalizes a document through context distillation at inference time. These works largely treat the weight update as a stand-alone replacement for the context, often under simplified assumptions such as fixed query tokens or access to only a subset of the original context. PatchKV repurposes context-to-weight conversion for a more practical use case: rather than replacing the context, the patch complements an off-the-shelf compressed KV cache by closing its residual compression error in closed form, making the mechanism directly applicable to existing long-context serving pipelines.

5 Conclusion

We introduce PatchKV, a framework that complements KV cache compression with context-derived weight patches. Given an off-the-shelf compressed cache, PatchKV constructs block-wise weight patches by aligning activations of reference queries with full cache and compressed cache, yielding a closed-form ridge-regression solution. Experiments show that our method offers consistent improvements over compression methods at low-cache regime, across diverse benchmarks and models. These results suggest that PatchKV provides a general complement to cache-only compression, enabling more aggressive KV cache compression without increasing per-query inference cost.

Acknowledgments and Disclosure of Funding

This work was in part supported by the National Research Foundation of Korea (RS-2024-00351212 and RS-2024-00436165), the Institute of Information & communications Technology Planning & Evaluation (IITP) (RS-2022-II220926, and RS-2022-II220959, RS-2026-25518317, RS-2019-II190075). This work was also supported by the InnoCORE program of the MSIT (AI Meta-Scientist, N10260110), the “HPC support” project funded by the Korea government, and the Korea Meteorological Administration Research and Development Program "Developing AI Technology for Weather Forecasting" under Grant (KMA2021-00121).

References

[1] YUCHENG LI, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H. Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV

cache-centric analysis of long-context methods. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=gkUyYcY1W9.

[2] 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.

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

[4] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. In *International Conference on Machine Learning*, pages 32332–32344. PMLR, 2024.

[5] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

[6] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024.

[7] Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. *Advances in neural information processing systems*, 37:119638–119661, 2024.

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

[9] Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-agnostic KV cache compression with context reconstruction. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=JFygzwx8SJ.

[10] Sabri Eyuboglu, Ryan Saul Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Ruoyu Liu, Atri Rudra, James Y Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*, 2025.

[11] Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. In *International Conference on Machine Learning*, pages 57355–57373. PMLR, 2025.

[12] Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast KV compaction via attention matching. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=t01SKTj8pP.

[13] Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Danning Ke, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient kv cache compression through retrieval heads. In *The Thirteenth International Conference on Learning Representations*, 2025.

[14] Akshat Sharma, Hangliang Ding, Jianping Li, Neel Dani, and Minjia Zhang. Minikv: Pushing the limits of 2-bit kv cache via compression and system co-design for efficient long context inference. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 18506–18523, 2025.

[15] Benoit Dherin, Michael Munn, Hanna Mazzawi, Michael Wunder, and Javier Gonzalvo. Learning without training: The implicit dynamics of in-context learning. *arXiv preprint arXiv:2507.16003*, 2025.

[16] Adrian Goldwaser, Michael Munn, Javier Gonzalvo, and Benoit Dherin. Equivalence of context and parameter updates in modern transformer blocks. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=aZ5Di8Ii4X.

[17] Hanna Mazzawi, Benoit Dherin, Michael Munn, Michael Wunder, and Javier Gonzalvo. Transmuting prompts into weights. *arXiv preprint arXiv:2510.08734*, 2025.

[18] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Jian Su, Kevin Duh, and Xavier Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264/.

[19] Greg Kamradt. Needle in a haystack-pressure testing llms, 2023.

[20] 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.

[21] LI Haoyang, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, HU Nicole, Wei Dong, Li Qing, and Lei Chen. A survey on large language model acceleration based on kv cache management. *Transactions on Machine Learning Research*, 2025.

[22] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[23] Qwen: An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

[24] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[25] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[26] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=ayi7qezU87.

[27] Junjie Peng, You Wu, Haoyi Wu, Jialong Han, Xiaohua Xie, Kewei Tu, and Jianhuang Lai. Grkv: Global regression for training-free kv cache compression in long-context llms. *arXiv preprint arXiv:2605.31105*, 2026.

[28] Yu Li, Binxu Li, and Tian Lan. MomentKV: Closing the directional gap in KV cache eviction for long-context inference. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=Mvdvz3jHiK.

[29] Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-loRA: Learning to instantly internalize contexts. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=iW1oBBO72S.

[30] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380, 2018.

[31] Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. Turboquant: Online vector quantization with near-optimal distortion rate. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tO3ASKZlok.

[32] Alan Julian Izenman. Reduced-rank regression for the multivariate linear model. *Journal of multivariate analysis*, 5(2):248–264, 1975.

A Algorithms

Algorithm 1 PatchKV weight patch construction

Require: Context c, base model f_θ with L blocks, KV cache compressor, cache budget b, ridge coefficient λ_0, chunk size L_{chunk}

$\Delta\theta_c \leftarrow \emptyset$

Cache preparation

Prefill full cache $C \leftarrow f_\theta(c)$

Compress $C_b \leftarrow \text{Compress}(C, b)$

Build reference set Q_c (repeat-context / self-study / joint)

Partition Q_c into chunks $\{q_j\}$

Initialize teacher and student states $x^F_{j,1}, x^C_{j,1}$ from each reference chunk q_j

Block-wise patch construction

for $\ell = 1$ **to** L **do**

  Initialize $S_H \leftarrow 0 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{ff}}}$, $S_T \leftarrow 0 \in \mathbb{R}^{d \times d_{\text{ff}}}$

  for each reference chunk q_j **do**

    Teacher forward: forward block ℓ from $x^F_{j,\ell}$ with full cache C; collect (h_i, z_i) and obtain $x^F_{j,\ell+1}$

    Student forward: forward block ℓ from $x^C_{j,\ell}$ with compressed cache C_b; collect (h'_i, z'_i)

    Compute targets

$$t_i \leftarrow W_\ell(h_i - h'_i) + (z_i - z'_i)$$

    Stack $H'_j \leftarrow [h'_i]_i$, $T_j \leftarrow [t_i]_i$

    Accumulate

$$S_H \mathrel{+}= H_j'^{\top} H'_j, \qquad S_T \mathrel{+}= T_j^{\top} H'_j$$

    Discard per-token activations (h_i, z_i, h'_i, z'_i)

  end for

  Patch computation:

$$\lambda \leftarrow \lambda_0 \frac{\|S_H\|_F^2}{\text{tr}(S_H)}$$

$$\Delta W_\ell \leftarrow S_T(S_H + \lambda I)^{-1}$$

  Merge $W_\ell \leftarrow W_\ell + \Delta W_\ell$

  $\Delta\theta_c \leftarrow \Delta\theta_c \cup \{\Delta W_\ell\}$

  for each reference chunk q_j **do**

    Student-state propagation: forward the patched block ℓ from $x^C_{j,\ell}$ with C_b and store the output as $x^C_{j,\ell+1}$

  end for

end for

return Patched weights $\theta + \Delta\theta_c$ and compressed cache C_b

B Further Details in Experimental Setup

B.1 Implementation details.

KV cache compression. As described in Sec. 3.1, we initialized the compressed KV cache with existing methods, KVzip [9] and Attention Matching [12]. For KVzip, we follow their default setting with prefill chunk size 2000 and global head scoring.

In Attention Matching, due to large memory and time cost in extremely long contexts such as SCBench [1], we select the key as HIGHESTATTENTIONKEYS that select KV cache key C_k by scoring. For other hyperparameters, we follow the best reported setting of Attention Matching using HIGHESTATTENTIONKEYS.

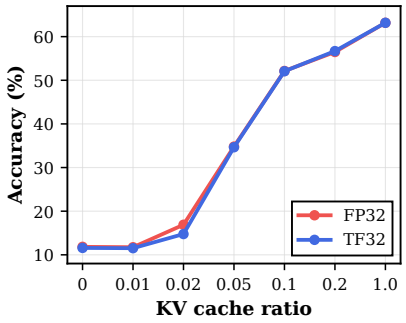

Figure 6: FP32 vs. TF32 patch construction.

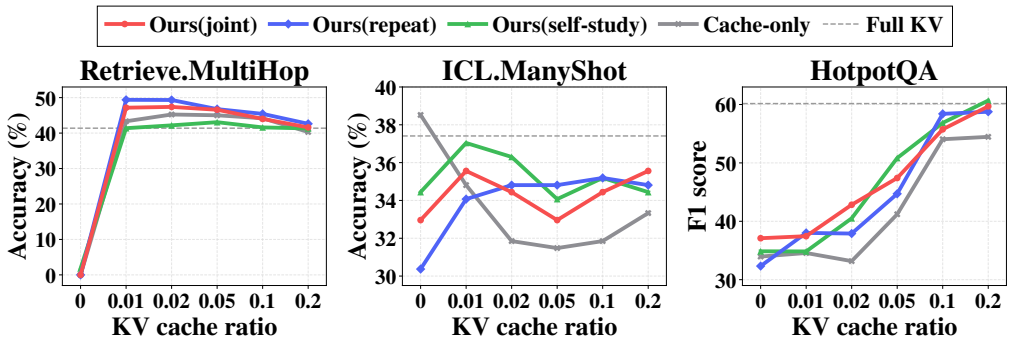

Figure 7: Additional benchmark results across KV cache budgets using Qwen2.5-7B-1M and KVzip.

Numerical Precision. Patch construction involves accumulating sufficient statistics and solving the ridge regression in Eq. (8), which can be sensitive to numerical errors under low-precision computation. We therefore perform patch construction at higher precision than the BF16 precision used for the base model inference. Specifically, we keep patch construction in FP32 while enabling TF32 for the matrix multiplications involved in computing the sufficient statistics. As shown in Fig. 6, TF32 achieves downstream performance comparable to FP32 across KV cache budgets, while reducing patch construction time from 147.91 s to 85.35 s. We therefore use TF32 matrix multiplication for patch construction throughout our experiments.

B.2 Dataset Selection and Task Variants

Dataset selection. SCBench includes nine tasks, of which seven are used in our main evaluation. We report `Retrieve.MultiHop` and `ICL.ManyShot` separately since compression itself can improve performance on these tasks in these evaluation settings. This makes it difficult to distinguish the effect of PatchKV from the benefit of compression itself. For `Retrieve.MultiHop`, which contains a large amount of distracting context, the evaluation results show that removing more context can improve performance by removing distractors. For `ICL.ManyShot`, we observe a more unusual pattern in which the baseline achieves its highest accuracy when no context KV pairs are retained, even exceeding the full-KV baseline. This suggests that the many-shot context may not always be beneficial in this setting. One possible explanation is that the pretrained model can solve much of the task directly, while the provided context introduce additional distracting signals. We report the results of `Retrieve.MultiHop` and `ICL.ManyShot` in Fig. 7.

Nevertheless, multi-hop reasoning remains an important setting for evaluating whether PatchKV remains effective when answering requires information from multiple parts of the context. As shown in Fig. 7, PatchKV with repeat-context or joint reference queries further improves over KVzip across several cache budgets in `Retrieve.MultiHop`. To further verify that this result is not specific to the highly redundant synthetic structure of `Retrieve.MultiHop`, we additionally evaluate PatchKV on HotpotQA [30], a natural multi-hop question answering benchmark. PatchKV improves over

Table 5: Score (%) on the original SCBench retrieval tasks using Qwen2.5-7B-1M. Both methods achieve zero accuracy on `Retr.KV` and `Retr.Prefix-Suffix` under the evaluated cache budgets.

Task	Method	0.20	0.10	0.05	0.02	0.01
`Retr.KV`	Attention Matching	0.0	0.0	0.0	0.0	0.0
	+ PatchKV	0.0	0.0	0.0	0.0	0.0
`Retr.Prefix-Suffix`	Attention Matching	0.0	0.0	0.0	0.0	0.0
	+ PatchKV	0.0	0.0	0.0	0.0	0.0
`Code.RepoQA`	Attention Matching	15.23	3.18	1.59	0.23	0.23
	+ PatchKV	33.64	8.64	3.41	0.91	0.68

Table 6: Stagewise wall-clock time and peak GPU memory for PatchKV construction.

Stage	**Time (s)**	**Peak Memory (GB)**
(i) Cache compression (KVzip)	67.82	24.89
(ii) Teacher forward	41.41	30.92
(iii) Student forward	16.38	30.99
(iv) Patch computation	11.40	32.98
(v) Student-state propagation	16.16	31.11
Total	153.17	32.98

KVzip across the evaluated nonzero cache budgets, providing complementary evidence that PatchKV remains effective on multi-hop reasoning tasks.

Task variants. For the Attention Matching experiments in Fig. 2, we use shortened variants of the SCBench retrieval tasks. On `Retr.KV` and `Retr.Prefix-Suffix`, Attention Matching collapses to zero accuracy under the evaluated budget, and applying PatchKV does not recover nonzero performance. The original tasks therefore provide little resolution for evaluating the additional compensating effect of PatchKV over Attention Matching. We instead evaluate the corresponding shortened variants, where the cache-only baseline retains nonzero performance under aggressive compression and the effect of weight-space compensation can be measured. For completeness, we report the results on the original tasks in Tab. 5.

C Additional Experiment Results.

Runtime Analysis. We present a stagewise breakdown of the additional computational cost introduced by PatchKV. Patch construction consists of a one-time cache-compression stage, followed by four stages repeated for each transformer block: (ii) teacher forward, (iii) student forward, (iv) patch computation, and (v) student-state propagation. Wall-clock time and peak GPU memory for each stage are reported in Tab. 6, measured on a context of 169,035 tokens with repeat-context reference query and cache budget 0.2. For steps (ii)-(v), wall-clock time is aggregated for all transformer blocks.

Reference context budget. In Fig. 8, we analyze the effect of limiting the amount of context used for patch construction on SCBench RepoQA, where the context length reaches up to 68K tokens. By controlling how many chunks of the context are fed as reference queries, we show that PatchKV achieves notable gains even with a small reference budget: using only 5K tokens (a single chunk) already recovers 15% of the gap between the compressed and full-KV baseline. Performance continues to improve as more context is included, but with diminishing returns, suggesting that a modest reference budget is sufficient in practice and that PatchKV's construction cost can be reduced significantly without sacrificing much accuracy.

Compatibility with quantized KV caches. A complementary line of work reduces KV cache memory by quantizing it to lower precision [5, 31]. Since PatchKV operates as a residual update to the model weights and is agnostic to how the cache itself is stored, it can be combined with quantization without any modification. To verify this, we apply PatchKV on top of INT4 and INT8

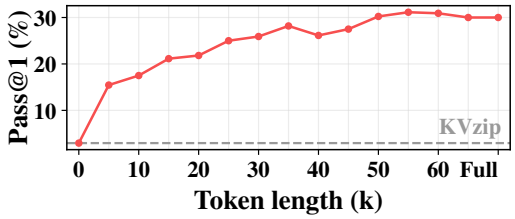

Figure 8: Performance across reference context lengths.

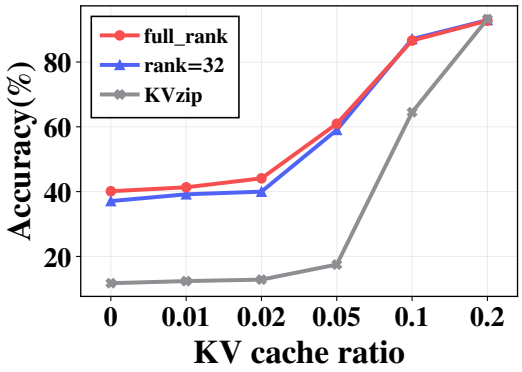

Figure 9: Performance when restricting the rank of weight patch of PatchKV.

quantized KV caches and report the results in Fig. 10. PatchKV improves accuracy over the baseline at both INT8 and INT4 precision, with gains comparable to those observed in the unquantized setting. This confirms that PatchKV serves as an orthogonal solution to KV cache quantization for efficient LLM inference.

Low-rank weight patches. In some deployment scenarios, the weight patch may need to be stored alongside the base model, introducing additional memory overhead. We therefore consider a low-rank variant of PatchKV that restricts the patch to rank r (we set $r = 32$). Concretely, we apply a reduced-rank regression [32] to the closed-form ridge solution, where we project $\Delta W_{\text{ridge}} = S_T(S_H + \lambda I)^{-1}$ onto its top-r output subspace via $\Delta W_r = U_r U_r^\top \Delta W_{\text{ridge}}$. Here U_r contains the top-r eigenvectors of $\Delta W_{\text{ridge}} S_H \Delta W_{\text{ridge}}^\top$. This keeps the closed-form structure intact and reduces per-context patch storage from $d \cdot d_{\text{ff}}$ to $(d + d_{\text{ff}}) \cdot r$. In Fig. 9, we compare the performance of full-rank PatchKV with the variant with weight patch with rank-32 on Qwen2.5-7B-1M, constructed by repeat-context reference query. As shown, the low-rank patch matches full-rank performance within a small margin while reducing per-context patch storage by 99%, demonstrating its practical viability.

Online Extension. Our main experiments construct a context-specific patch once and reuse it across subsequent queries. We additionally examine whether weight-space compensation can be repeatedly applied as the KV cache evolves during generation. Following the online compaction setting of Attention Matching (AM), we evaluate an online variant of PatchKV on AIME 2025 using Qwen3-4B, repeat-context reference queries. During decoding, we compact the KV cache whenever it reaches $P = 2048$ entries. At each event, we protect the most recent 20 entries and apply AM. PatchKV is then applied on top of the compacted cache before decoding resumes.

We evaluate generation limits of 4K and 8K tokens while varying the KV cache size after compaction. As shown in Tab. 7, PatchKV provides larger gains under more aggressive compaction. With an 8K and 4K generation limit, PatchKV show improvements compared to AM in more aggressive budgets, whereas PatchKV provides little or no improvement at larger post-compaction cache sizes. These results provide initial evidence that weight-space compensation can be repeatedly applied as generated KV states are compressed during long reasoning. A broader evaluation of repeated updates and their long-horizon stability is left for future work.

Table 7: Online evaluation on AIME 2025 using Qwen3-4B. A compaction event is triggered whenever the physical KV cache reaches 2048 entries, with the most recent 20 entries protected during compaction. $2048 \rightarrow m$ denotes the post-compaction KV cache size.

KV cache size	Method	4K Acc.	8K Acc.
Full KV	Full	9/30	14/30
$2048 \rightarrow 1034$	AM	9/30	**13/30**
	AM + PatchKV	9/30	12/30
$2048 \rightarrow 527$	AM	**9/30**	11/30
	AM + PatchKV	8/30	**13/30**
$2048 \rightarrow 222$	AM	2/30	4/30
	AM + PatchKV	**8/30**	**8/30**
$2048 \rightarrow 121$	AM	2/30	2/30
	AM + PatchKV	**6/30**	**5/30**

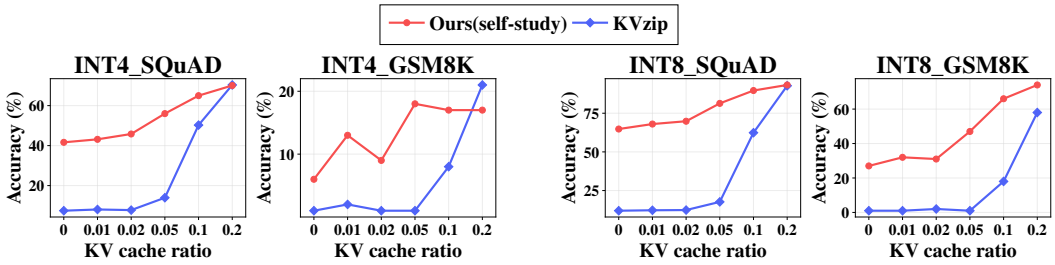

Figure 10: Performance of PatchKV on INT4/INT8 quantized KV cache.

More qualitative examples. From Fig. 11 to Fig. 14, we show qualitative examples of query and answer with and without PatchKV on evicted context.

Full experiment results. In Fig.15 and Fig.16, we report the full result of benchmarks evaluated in Qwen3-4B and LLaMA3.1-8B. Both models show similar property with Qwen2.5-7B-1M, consistently improving across all benchmarks and variants.

D Limitations

PatchKV relies on context-derived reference queries as proxies for unknown future queries, so the reference-query mismatch may limit recovery under larger distribution shifts. Patch construction also incurs a one-time preprocessing cost and access to the full KV cache as a teacher, making the method most suitable when this cost can be amortized across repeated queries or sufficiently long decoding. Our primary evaluation targets the single-context, multi-query setting, while concurrent serving of many context-specific patches requires additional serving mechanisms. The online results in Tab. 7 provide initial evidence beyond this static-context setting, but broader evaluation remains future work.

Question: Who currently provides uniforms to Notre Dame sport teams? [SQuAD]

Pruned Context: ~~On July 1, 201~~4, ~~the~~ University of ~~Notre Dame~~ and ~~**Under Armour**~~ reached an agreement in which ~~**Under Armour**~~ will provide uniforms, apparel, equipment, and monetary compensation to Notre Dame for 1~~0~~ years. [...]

Full KV: Under Armour currently provides uniforms to Notre Dame sport teams. CORRECT

Pruned KV (w/o patch): Gymboree Uniforms INCORRECT

PatchKV (Ours): Under Armour currently provides uniforms to Notre Dame sport teams. CORRECT

Figure 11: Qualitative example of PatchKV over KVzip on SQuAD. Evicted tokens shown in red.

Question: Which other song from the soundtrack did better in the charts? [SQuAD]

Pruned Context: [...] Bey~~oncé released "F~~ighting Tem~~pt~~tation" as ~~the~~ lead ~~single from~~ the film's soundtrack album, with Missy Elliott~~,~~ MC Lyte, and Free which was also used to promote the film. Another of Beyoncé's contributions to the soundtrack, "**Sum~~mertime~~**", fared better on the US charts.

Full KV: "Summertime" CORRECT

Pruned KV (w/o patch): I need more context to provide an accurate answer. Could you please specify which soundtrack you're referring to? INCORRECT

PatchKV (Ours): "Summertime" CORRECT

Figure 12: Qualitative example of PatchKV over KVzip on SQuAD. Evicted tokens shown in red.

Question: What is the most popsicle sticks he can make if he buys the cheapest lumber? [GSM8K]

Pruned Context: Frederick is making popsicles to sell and to save money he is making his own popsicle sticks. He can get 200 sticks from a 2 x 4 piece of wood and 400 sticks from a ~~**2 x 8**~~ piece of wood. He has $24 to buy wood for sticks. A ~~**2 x 4**~~ costs $4. A ~~**2 x 8**~~ costs $~~**6**~~.

Full KV: The answer is 1600 popsicle sticks. CORRECT

Pruned KV (w/o patch): The most popsicle sticks he can make if he buys the cheapest lumber depends on the length ! and yield of the wood, but assuming 10 sticks per foot of 1x4 pine, a 10-foot piece would yield 100 popsicle sticks. INCORRECT

PatchKV (Ours): The answer is that Frederick can make 1600 popsicle sticks if he buys the cheapest lumber. CORRECT

Figure 13: Qualitative example of PatchKV over KVzip on GSM8K. Evicted tokens shown in red.

Question: What is Marcus' total bill? [GSM8K]

Pruned Context: Marcus ordered 5 croissants at ~~**$3.00**~~ apiece, 4 cinnamon rolls at $2~~.~~50 each, 3 mini quiches for ~~**$4.00**~~ apiece and 13 blueberry muffins that were $1~~.~~00 apiece. At check out, Marcus shows his loyalty card that gives him ~~**10%**~~ off of his purchase.

Full KV: The answer is Marcus' total bill is $45.00. CORRECT

Pruned KV (w/o patch): The answer is unknown based on the given information. INCORRECT

PatchKV (Ours): The answer is $45.00. CORRECT

Figure 14: Qualitative example of PatchKV over KVzip on GSM8K. Evicted tokens shown in red.

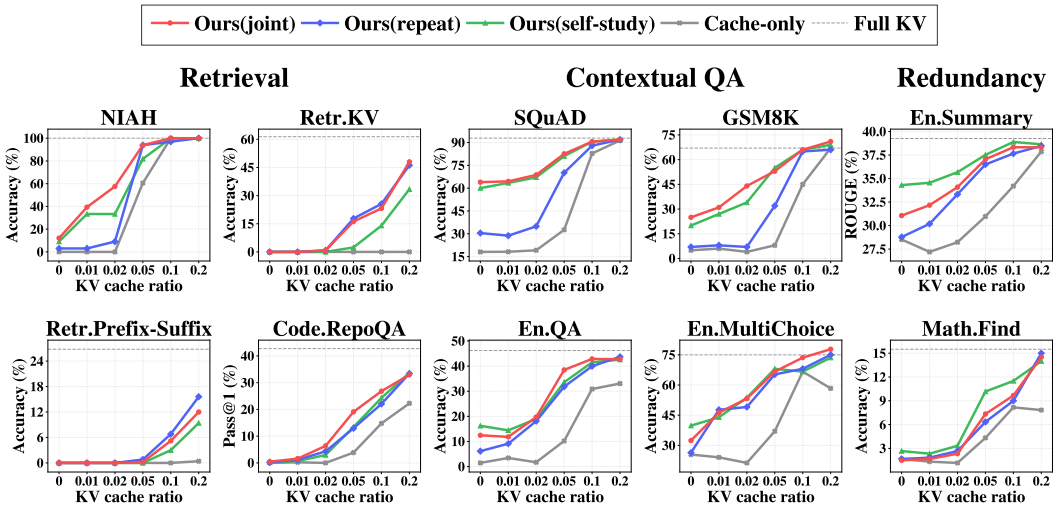

Figure 15: Benchmark results using LLaMA3.1-8B with PatchKV applied on top of an eviction based method (KVzip).

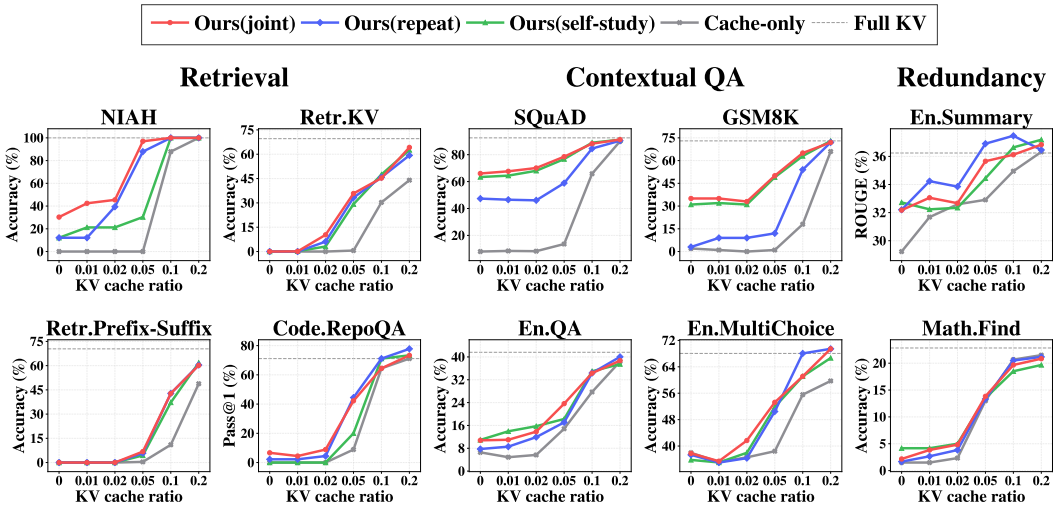

Figure 16: Benchmark results using Qwen3-4B with PatchKV applied on top of an eviction based method (KVzip).

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: Our claims made in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: We discuss limitations in Appendix D.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

Justification: We do not include theoretical results.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: Our results are fully reproducible.

   Guidelines:

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

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

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

Answer: [Yes]

Justification: We cite all datasets we used. We plan to release the implementation, evaluation scripts, and instructions for reproducing the main experimental results in a public repository upon publication.

Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: We provide details within our paper, including ablation studies.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [No]

   Justification: Our main experiments use fixed benchmark splits and deterministic decoding, and we report aggregate performance over all evaluated examples. However, we do not include repeated-run error bars, confidence intervals, or statistical significance tests in the current submission.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: We state the resources used for our experiment.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: Yes we follow NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [N/A]

    We think no societal impacts exist.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: we do not release a new high-risk model or dataset.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [No]

    Justification: We cite the models, datasets, and baseline methods used in our experiments, but the current submission does not explicitly list all corresponding licenses and terms of use. We will include this information in the final version.

    Guidelines:

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

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

13. **New assets**

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

    Answer: [N/A]

    Justification: We do not release new assets.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: We do not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: We do not involve research with human subjects.

    Guidelines:

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

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

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: Our core method does not involve LLMs.

    Guidelines:

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