SD-CACHE: LEARNABLE MEMORY CONSOLIDATION WITH STATE-GUIDED DYNAMIC CACHING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are increasingly deployed as persistent services that must reason over large volumes of external knowledge. Retrieval-augmented generation (RAG) grounds model responses in relevant documents, but the retrieved content must still be encoded and stored in the key-value (KV) cache. As this content accumulates, the cache grows linearly, increasing memory consumption and inference cost. Existing memory-compression methods address this problem either by retaining a subset of cached states or by learning context compressors that map documents to compact latent representations. Despite these advances, consolidating information under tight memory budgets while preserving its usefulness for future queries remains challenging. We investigate how the model's native KV states can be selectively consolidated into a fixed-size working memory without access to future queries. To this end, we introduce SD-Cache (**S**tate-guided and **D**ynamically updated Cache), a query-agnostic KV-memory consolidation framework that operates alongside a frozen generator LLM. SD-Cache learns to consolidate layer-wise KV states into fixed-size memory through selective anchor-based pooling and state-guided refinement, processing new contexts in a single forward pass without context-specific optimisation. Experiments on retrieval-augmented question answering and synthetic long-context benchmarks show that SD-Cache maintains competitive downstream performance under constrained memory budgets.

1 INTRODUCTION

Large language models (LLMs) have demonstrated remarkable capabilities as general-purpose systems, powering a wide range of applications, including virtual customer-service assistants (Chkirbene et al., 2024), code generation (Yang et al., 2025), and mathematical problem solving (Hendrycks et al., 2021; Lewkowycz et al., 2022). Many are deployed as interactive services that rely on retrieval-augmented generation (RAG) (Lewis et al., 2020) to access evolving domain-specific knowledge, with retrieved content encoded and retained in the key-value (KV) cache.

This reliance on KV-cache storage creates a scaling bottleneck: the cache grows linearly with context length, increasing memory consumption and limiting inference throughput (Grattafiori et al., 2024; Eyuboglu et al., 2026). These costs are compounded across successive interactions, where customer-service systems may repeatedly retrieve and re-encode overlapping policy documents and database records, increasing pre-fill latency (Labruna et al., 2025; Tian et al., 2022). Context compression therefore offers a practical way to reduce the KV storage required for subsequent generation.

Existing KV-cache compression methods fall broadly into training-free and learning-based approaches. Training-free methods commonly retain a subset of KV states using positional heuristics (Xiao et al., 2024) or attention-based importance estimates (Zhang et al., 2023; Li et al., 2024). Beyond token selection, context-specific optimisation constructs compact KV representations that approximate the attention behaviour of the original cache (Zweiger et al., 2026; Eyuboglu et al., 2026). Although these methods require no additional model training, they rely on predefined heuristics or context-specific optimisation. Under aggressive compression (e.g., above $\times 16$), token-selection methods may suffer substantial information loss, as information carried exclusively by discarded states is lost rather than consolidated into the retained cache (Chari et al., 2025).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

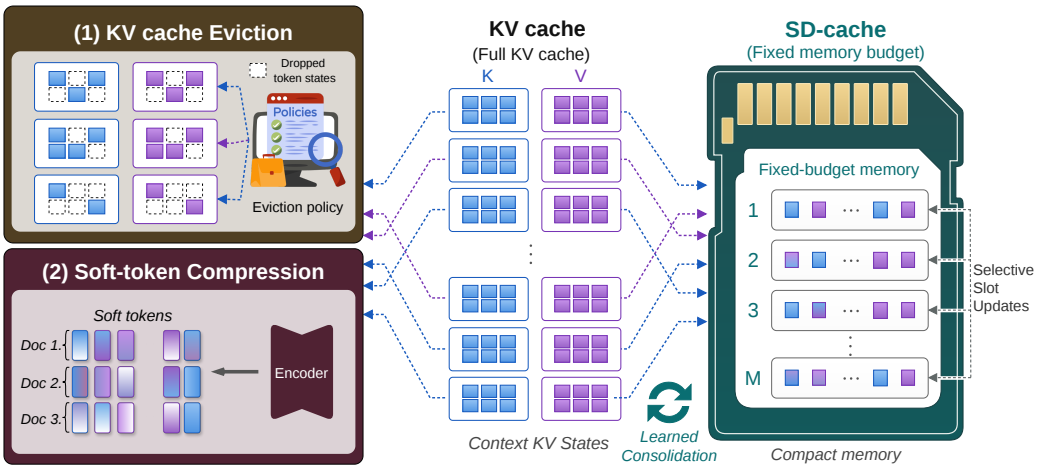


Figure 1: Conceptual comparison of SD-Cache with existing training-free (e.g., KV cache eviction (Li et al., 2024)) and learning-based compression (e.g., soft-token compression (Louis et al., 2025)) paradigms. SD-Cache instead learns to consolidate KV states into a fixed-budget memory, retaining model-native representations while supporting selective layer-wise updates.

Learning-based approaches aim to amortise the cost of context processing across multiple queries through reusable compressed representations. These representations are either optimised for a specific corpus (Eyuboglu et al., 2026) or generated by shared compressors that can be applied to unseen documents (Rau et al., 2024b; Chevalier et al., 2023; O'Neill et al., 2026). Such compressors can encode documents as continuous embeddings (Louis et al., 2025) or generate document-specific parameter updates through hypernetworks (Charakorn et al., 2026; Ha et al., 2016; Hu et al., 2021). However, approaches based on soft tokens or document-specific parameter updates learn alternative representations of the input context for subsequent reuse, rather than directly consolidating the layer-wise KV states already formed by the frozen LLM.

In neuroscience, memory consolidation refers to the time-dependent stabilisation and reorganisation of newly encoded memories into durable representations (Squire et al., 2015). Inspired by this process, we explore context compression for RAG as a memory-consolidation problem. Specifically, we ask: ***Can we learn to maintain a fixed-size KV working memory by selectively consolidating contextual information, without access to future queries?*** To address this question, we introduce SD-Cache, a query-agnostic framework that augments a frozen generator LLM with an auxiliary Transformer-based network (Fig. 1). SD-Cache learns a consolidation operation that dynamically constructs compact KV memory for new contexts in a single forward pass, without context-specific optimisation. Motivated by the clustering structure observed in per-head KV representations, we design an anchor-pooling module (APM) that initialises M memory slots from the KV states of the first M tokens and refines them through novelty-guided pooling and gated updates, progressively consolidating the sequence within a fixed memory budget. The resulting attention weights jointly pool the incoming keys and values, preserving key-value correspondence as the slots evolve. To capture key–value associations that slot-wise pooling may overlook, a recurrent state-guided correction module builds an associative state directly from the original KV states and uses it to refine the updated slots. SD-Cache thereby maintains a query-agnostic memory whose size remains fixed as the context grows. Our main contributions are:

- We formulate query-agnostic KV-cache compression as a learnable memory-consolidation problem and introduce SD-Cache, which maintains a fixed memory budget while leaving the generator LLM frozen.
- We develop a two-stage consolidation mechanism that combines anchor-based pooling to selectively integrate key-value pairs with state-guided correction to capture chunk-level key-value associations that slot-wise pooling may overlook.
- We demonstrate on RAG-based question answering (Rau et al., 2024a; Louis et al., 2025) and synthetic long-context benchmarks (Hsieh et al., 2024; Yang et al., 2026) that SD-Cache maintains strong downstream performance at high compression ratios.

2 Related Work

KV Cache Eviction Methods. KV cache eviction methods (Xiao et al., 2024; Zhang et al., 2023; Oren et al., 2024; Li et al., 2024; Cai et al., 2024; Yang et al., 2026; Gould & Hengel, 2026) formulate memory management as an active forgetting (Davis & Zhong, 2017) problem, typically implemented through a scoring-and-selection rule. Given cached keys and values $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{L\times H\times D}$, where L, H, and D denote the sequence length, number of KV heads, and per-head feature dimension, respectively, an eviction policy assigns each cache position i an importance score $s_i \in \mathbb{R}$. Under a budget of $M < L$ positions, the retained indices are selected as $\mathcal{I}_M = \mathrm{TopM}(\{s_i\}_{i=1}^{L})$, yielding $\widetilde{\mathbf{K}} = \mathbf{K}[\mathcal{I}_M]$ and $\widetilde{\mathbf{V}} = \mathbf{V}[\mathcal{I}_M]$. Retention policies typically use positional (Xiao et al., 2024), attention-based (Zhang et al., 2023; Oren et al., 2024; Li et al., 2024), or predictability-based signals (Gould & Hengel, 2026). Beyond retention criteria, cache budgets can be allocated unevenly across layers to accommodate differences in attention sparsity (Cai et al., 2024), while information from evicted tokens can be encoded into an auxiliary latent memory (Yang et al., 2026). These strategies preserve selected token states or compensate for eviction losses, but do not learn to consolidate information from evicted states directly into the retained KV cache.

Learnable Context Compression Methods. Learnable compression maps a context $X = (x_1, \ldots, x_L)$ to a compact representation $\mathbf{Z} = f_\theta(X) = (\mathbf{z}_1, \ldots, \mathbf{z}_M)$, where $M \ll L$ and $f\theta$ is a trainable compressor. Many methods encode contexts into soft embeddings or memory tokens that condition subsequent generation (Chevalier et al., 2023; Ge et al., 2023; Cheng et al., 2024; Rau et al., 2024b; Louis et al., 2025). Others operate directly on internal KV representations (Eyuboglu et al., 2026), including approaches that combine learned token selection with parameter-efficient adaptation to consolidate information into retained states (Chari et al., 2025). Concurrent work uses per-layer compactors to synthesise compact KV states from the original cache (O'Neill et al., 2026). While these approaches achieve context compression, our work focuses on learning to selectively consolidate incoming KV states into an existing memory under a fixed budget with a single pass.

3 Method

Given a frozen L-layer Transformer, a context of length T induces per-layer key and value states $K^{(\ell)}, V^{(\ell)} \in \mathbb{R}^{H_{\mathrm{kv}}\times T\times d_h}$ where $\ell = 1, \ldots, L$, H_{kv} denotes the number of KV heads and d_h the head dimension. We denote the full KV state induced by the context as $\mathcal{C} = \left\{\left(K^{(\ell)}, V^{(\ell)}\right)\right\}_{\ell=1}^{L}$. A *memory consolidation operator* transforms the growing KV state into a compact memory $\mathcal{M} = \left\{\left(\widetilde{K}^{(\ell)}, \widetilde{V}^{(\ell)}\right)\right\}_{\ell=1}^{L}$, where $\widetilde{K}^{(\ell)}, \widetilde{V}^{(\ell)} \in \mathbb{R}^{H_{\mathrm{kv}}\times M\times d_h}$ and $M \ll T$. Formally, we define a consolidation operator

$$\mathcal{O}_\phi : \mathcal{C} \to \mathcal{M}, \qquad \mathcal{M} = \mathcal{O}_\phi(\mathcal{C}), \tag{1}$$

which maintains a fixed memory budget while preserving the information required for future model computation. Given an unknown future query Q, let $Y_\mathcal{C} = (y_1, \ldots, y_N) = \mathrm{Decode}\left(p_\theta(\cdot \mid \mathcal{C}, Q)\right)$ denote the output produced by the frozen Transformer using the full KV state. The desired consolidated memory should induce model behaviour close to that obtained from the full KV state:

$$\phi^* = \arg\min_\phi \; \mathbb{E}_{Q\sim\mathcal{D}_Q}\left[-\sum_{i=1}^{N} \log p_\theta\left(y_i \mid y_{<i}, \mathcal{O}_\phi(\mathcal{C}), Q\right)\right], \tag{2}$$

where $\mathcal{D}_Q$ denotes the distribution of future queries. Importantly, consolidation is designed to be *query-agnostic*: the future query Q is not available to $\mathcal{O}_\phi$ when $\mathcal{M}$ is constructed, as depicted in Fig. 2. Let $\mathcal{C}_s$ denote the KV states induced by the incoming context chunk at update step s, and let $\mathcal{M}_{s-1}$ denote the consolidated memory retained from the preceding context. For the arriving context, we extend this formulation by allowing the operator to update the existing consolidated memory with newly observed KV states

$$\mathcal{M}_s = \mathcal{O}_\phi\left(\mathcal{M}_{s-1}, \mathcal{C}_s\right), \tag{3}$$

where $\mathcal{O}\phi = \mathcal{O}_{\mathrm{SGC}} \circ \mathcal{O}_{\mathrm{APM}}$ and $\mathcal{M}s$ remains constrained to a fixed budget of M slots for all s.

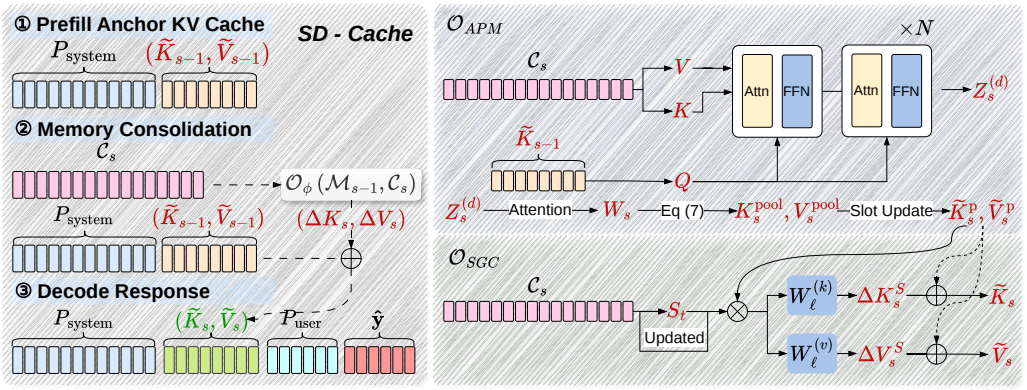


Figure 2: Illustration of our SD-Cache method. The overall SD-Cache pipeline can be summarised in three major steps when starting from scratch: 1) initialise the memory with the KV states of the first M document tokens, which serve as anchors; 2) selectively update the existing memory $\mathcal{M}_{s-1}$ as new context KV states $\mathcal{C}_s$ arrive, while maintaining a fixed budget of M slots; and 3) use the updated memory $\mathcal{M}_s$ together with the system and user prompts to generate a response with the frozen decoder. The right panel details the memory update mechanism: $\mathcal{O}_{APM}$ performs novelty-guided pooling and gated slot updates, while $\mathcal{O}_{SGC}$ refines the pooled KV representations using an associative state accumulated from the original KV stream.

3.1 Anchor-Pooling Module ($\mathcal{O}_{\text{APM}}$)

APM pools native KV states into compact memory slots, with its design motivated by the structured geometry of Transformer representations. As illustrated in Fig. 3, key representations exhibit highly structured, head-specific geometry that remains readily identifiable across layers, with similar structure observed for query representations. These observations suggest that the model's native KV states provide a natural basis for constructing compact memory representations. For clarity, we omit the layer and KV-head indices below and describe the update for a single set of KV states. At update step s, let $(\widetilde{K}_{s-1}, \widetilde{V}_{s-1})$ denote the current memory of M slots, and let (K_s, V_s) denote the KV states induced by $\mathcal{C}_s$. APM follows a *novelty-aware write* procedure consisting of novelty estimation, anchor pooling, and selective slot update. We index the first update by $s = 1$ and initialise $\mathcal{M}_0$ using the KV states of the first M context tokens, which serve as the initial anchors.

Coverage-Aware Novelty Estimation. Before modifying the memory, each incoming key first queries the existing memory to estimate how well it is already represented. For the key $\mathbf{k}_s$ in the incoming key states K_s, we compute $\mathbf{q}_s = \mathrm{LayerNorm}(\mathbf{k}_s)$ and reconstruct it from a stop-gradient copy of the current memory:

$$\mathbf{r}_s = \mathrm{Attn}\left(\mathbf{q}_s, \mathrm{sg}[\widetilde{K}_{s-1}], \mathrm{sg}[\widetilde{K}_{s-1}]\right). \tag{4}$$

A lightweight MLP then predicts a scalar novelty score from the incoming representation, its reconstruction, and their residual:

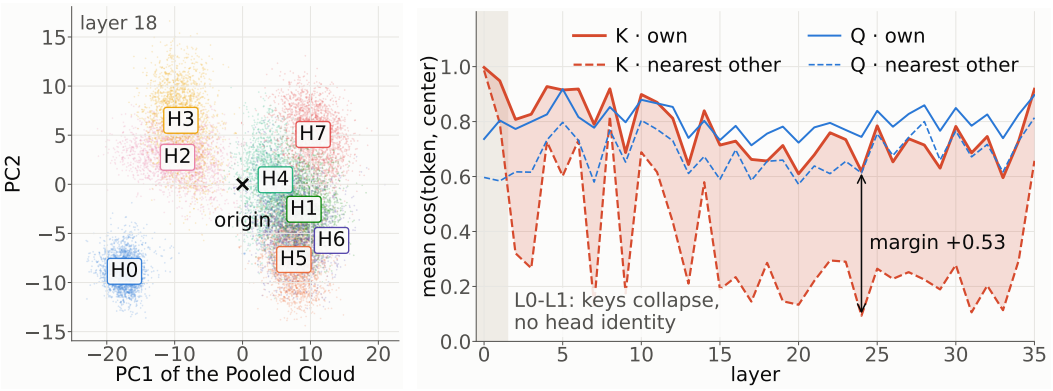
$$\mathbf{n}_s = \mathrm{MLP}_{\mathrm{nov}}\left([\mathbf{q}_s;\, \mathbf{r}_s;\, \mathbf{q}_s - \mathbf{r}_s]\right). \tag{5}$$

Intuitively, an incoming state that is already well represented by the existing anchors should require little additional memory capacity, whereas poorly reconstructed states provide stronger evidence for updating the memory. We stop gradients through the memory during this read operation to prevent the anchors from artificially increasing the reconstruction error to obtain a larger novelty score.

Novelty-Guided Pooling. The current memory keys initialise a set of write queries, $Z_s^{(0)} = \widetilde{K}_{s-1}$, which are refined by attending to the incoming keys. We update these queries using a stack of L_p Perceiver-style blocks (Jaegle et al., 2021). For the d-th block,

$$Z_s^{(d)} = \mathrm{PerceiverBlock}\left(Q = Z_s^{(d-1)}, K = K_s, V = K_s;\, \mathrm{bias} = \lambda n_s\right), \tag{6}$$

where λ is a learnable scalar. Each block consists of a pre-normalised cross-attention layer followed by a pre-normalised feed-forward network. The novelty scores $\mathbf{n}_s$ are added directly to the cross-attention logits, biasing the write queries towards incoming keys that are poorly represented by the



(a) PCA for the Pre-RoPE Keys feature.

(b) Similarity to Own vs. Nearest Other Head Centre.

Figure 3: Visualisation of the PCA projection (Abdi & Williams, 2010) of pre-RoPE keys at layer 18 (Figure 3a) and comparison of the mean cosine similarities of keys and queries to their own-head centre and nearest other-head centre across layers (Figure 3b). The plots are produced using Qwen3-4B (Yang et al., 2025) with 9,204 natural-language tokens sampled from GoodWiki (Choi, 2023).

current memory. After L_p blocks, $Z_s^{(L_p)}$ forms the refined write queries used to pool the incoming KV states. We compute a single set of attention weights and apply it to both keys and values:

$$W_s = \mathrm{softmax}\left(\frac{Z_s^{(L_p)} K_s^\top}{\sqrt{d_h}} + \lambda n_s\right), \; K_s^{\mathrm{pool}} = W_s K_s, \; V_s^{\mathrm{pool}} = W_s V_s. \tag{7}$$

Using the same weights for both keys and values ensures that each pooled KV pair aggregates information from the same context positions.

Selective Slot Update. Finally, a shared per-slot gate determines how much newly pooled information is incorporated into each memory slot: $\mathbf{g}_s = \sigma\left(\mathrm{MLP}_{\mathrm{gate}}\left([\widetilde{K}_{s-1}; \widetilde{V}_{s-1}; K_s^{\mathrm{pool}}; V_s^{\mathrm{pool}}]\right)\right) \in (0,1)^M$. The same gate is shared between the key and value of each slot, preserving their correspondence:

$$\widetilde{K}_s^{\mathrm{p}} = (1-\mathbf{g}_s)\widetilde{K}_{s-1} + \mathbf{g}_s K_s^{\mathrm{pool}}, \qquad \widetilde{V}_s^{\mathrm{p}} = (1-\mathbf{g}_s)\widetilde{V}_{s-1} + \mathbf{g}_s V_s^{\mathrm{pool}}. \tag{8}$$

The gate is initialised to favour retaining the existing memory, allowing the slots to gradually learn to incorporate new information during training. The resulting $(\widetilde{K}_s^{\mathrm{p}}, \widetilde{V}_s^{\mathrm{p}})$ form the provisional consolidated memory produced by APM.

3.2 State-Guided Correction ($\mathcal{O}_{\mathrm{SGC}}$)

SGC complements APM with a compact associative state (Yang et al., 2024; Hatamizadeh et al., 2026) constructed directly from the original, uncompressed KV stream. Importantly, the state is updated from the raw KV states rather than from the pooled memory, providing an information path decoupled from the slot-pooling process. Within each KV head, the state $S_t \in \mathbb{R}^{d_h \times d_h}$ represents a key-to-value associative map accumulated over the original stream. For each token t in $\mathcal{C}_s$, we first normalise its key as $\hat{\mathbf{k}}_t = \mathbf{k}_t / \|\mathbf{k}_t\|_2$ and predict a data-dependent write strength $\beta_t = \sigma\left(\mathbf{w}_\beta^\top \mathbf{h}_t + b_\beta\right)$, where $\mathbf{h}_t$ denotes the corresponding hidden representation. Starting from $S_0 = 0$, the state first absorbs the M anchor tokens with $\beta = \sigma(b_\beta)$, and the associative state is then updated using a gated delta rule (Hatamizadeh et al., 2026):

$$S_t = \gamma_t S_{t-1}\left(I - \beta_t \hat{\mathbf{k}}_t \hat{\mathbf{k}}_t^\top\right) + \beta_t \mathbf{v}_t \hat{\mathbf{k}}_t^\top, \qquad \gamma_t = 1 - \beta_t. \tag{9}$$

This update continuously absorbs key-value associations from the original KV chunk, while the coupled decay stabilises the state over long sequences. After processing the KV states, we use the resulting associative state S to refine the updated slots $(\widetilde{K}_s^{\mathrm{p}}, \widetilde{V}_s^{\mathrm{p}})$ produced by APM. Their keys are first normalised and used to query the state:

$$R_s^v = S(\widehat{\widetilde{K}}_s)^\top, \qquad \widehat{\widetilde{K}}_s = \frac{\widetilde{K}_s^{\mathrm{p}}}{\|\widetilde{K}_s^{\mathrm{p}}\|_2}. \tag{10}$$

Since S defines only a key-to-value mapping, we use the same readout R_s^v to predict both key and value corrections through lightweight learned heads:

$$\Delta V_s^S = W_\ell^{(v)} R_s^v, \qquad \Delta K_s^S = W_\ell^{(k)} R_s^v, \tag{11}$$

where $W_\ell^{(v)}$ and $W_\ell^{(k)}$ are zero-initialised layer-specific projections. The corrected memory is then

$$\widetilde{K}_s = \widetilde{K}_s^{\mathrm{p}} + \alpha_k (\Delta K_s^S)^\top, \qquad \widetilde{V}_s = \widetilde{V}_s^{\mathrm{p}} + \alpha_v (\Delta V_s^S)^\top. \tag{12}$$

The state matrix S is only maintained during the update and is discarded afterwards, leaving no additional persistent state beyond the compact KV memory.

3.3 Training Objective

We keep the parameters θ of the underlying Transformer frozen and optimise only the parameters ϕ of $\mathcal{O}$. For each training example, we first decode the target sequence $Y_{\mathcal{C}} = (y_1, \ldots, y_N)$ using the full KV state $\mathcal{C}$. SD-Cache is then trained to reproduce this full-cache output using only the consolidated memory $\mathcal{M}$ with $\mathcal{L} = -\sum_{i=1}^{N} \log p_\theta\left(y_i \mid y_{<i}, Q, \mathcal{M}\right)$. This objective encourages the estimated memory to preserve the information required to reproduce the behaviour of the full KV.

4 Experiments

Evaluation Protocols. Our primary benchmark is retrieval-augmented question answering (RA-QA) Benchmark (Louis et al., 2025). The training data of this benchmark is taken from questions from multiple sources of QA datasets (Louis et al., 2025) [1], which contain 453,000 samples. Similar to PISCO (Louis et al., 2025), we also use the response from a teacher model [2] as the supervision signal of the next token prediction loss. The evaluation includes five popular QA datasets including: Natural Questions (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), PopQA (Mallen et al., 2023), ASQA (Stelmakh et al., 2022), and HotpotQA (Yang et al., 2018). Our main evaluation metric is the answer matching accuracy in the QA context, which we define as correct prediction if the normalised label is found within the normalised prediction (Louis et al., 2025). We adapt Mistral-7B-v0.2 (Jiang et al., 2023) as the backbone for this benchmark. We additionally report results on RULER (Hsieh et al., 2024) at a context length of 4096 to further evaluate the model's long-context retrieval and reasoning capabilities, including needle-in-a-haystack (NIAH), multi-hop tracing, and frequency-based aggregation tasks. For this benchmark, we adapt and augment the RA-QA training data with synthetic retrieval, frequency, and tracking tasks, and use Qwen3-8B (Yang et al., 2025), Mistral-7B-v0.3 (Jiang et al., 2023), and Llama-3.1-8B (Grattafiori et al., 2024) as generator backbones. Following the official protocol we score every task with the reference `string_match` implementation and report the unweighted mean over tasks as the aggregated score. Implementation details are provided in Appendix A.1.

4.1 Results on RA-QA Benchmark

Tab. 1 compares SD-Cache with existing methods on five RA-QA datasets. At a compression ratio of $16\times$, SD-Cache achieves the highest average accuracy among the compared compression methods, reaching 68.67% and outperforming PISCO by 0.73 percentage points (pp) at the same compression ratio. SD-Cache improves over PISCO on four of the five datasets, with the largest gains on ASQA and Natural Questions ($+2.01$ pp and $+1.34$ pp, respectively), while maintaining comparable performance on other datasets. Its average accuracy also remains within approximately 3 pp of the uncompressed Mistral-7B baseline. At $128\times$ compression, SD-Cache also outperforms PISCO (Louis et al., 2025) by 1.66 pp in average accuracy, further highlighting its effectiveness under aggressive compression. These results indicate that SD-Cache preserves information useful for downstream question answering under substantial context compression.

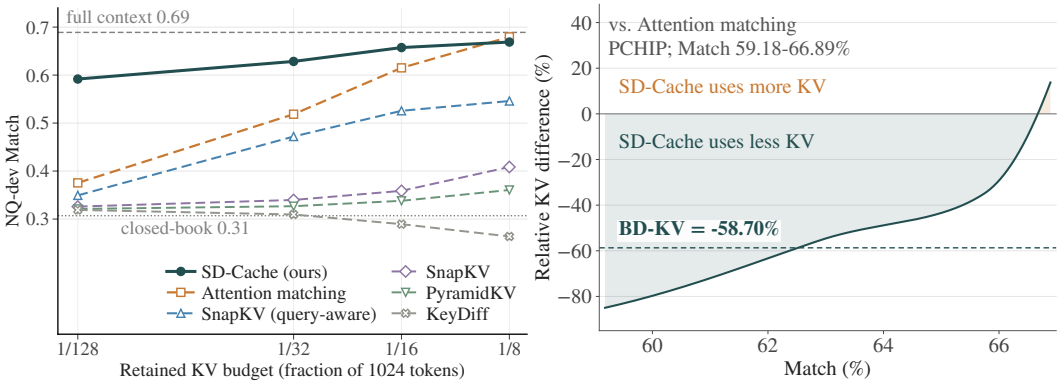
Performance under Different Compression Ratios. We further investigate the effect of compression on answer-matching accuracy using Qwen3-4B-Instruct (Yang et al., 2025) on the Natural

[1] https://huggingface.co/datasets/maxoul/pisco_finetuning_data

[2] https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

Table 1: **Answer-matching accuracy (%) on five RA-QA datasets (Louis et al., 2025).** Using five retrieved documents per query, SD-Cache achieves the highest average accuracy among the compared compression methods at a 16× compression ratio. † methods limited to one context. ‡ methods using Llama-2 7B. ⋆ use official checkpoint to reproduce the results.

	Rate	ASQA	HotpotQA	NQ	TriviaQA	POPQA	Average
Mistral-7B without Compression							
Closed-book (Louis et al., 2025)	-	51.00	34.00	46.00	79.00	29.00	47.80
Full KV (Louis et al., 2025)	-	74.00	51.00	69.00	92.00	70.00	71.00
Compression Models							
xRAG† (Cheng et al., 2024)	x128	34.00	27.00	32.00	77.00	33.00	40.00
A.C.†‡ (Chevalier et al., 2023)	x4	57.00	31.00	35.00	70.00	24.00	43.00
ICAE‡ (Ge et al., 2023)	x4	47.00	29.00	42.00	78.00	43.00	48.00
DODO‡ (Qin et al., 2024)	x10	52.00	38.00	48.00	82.00	47.00	53.00
COCOM (Rau et al., 2024b)	x16	63.00	46.00	58.00	89.00	48.00	61.00
PISCO⋆ (Louis et al., 2025)	x16	71.09	**48.06**	64.58	90.17	65.83	67.94
SD-Cache (Ours)	x16	**73.10**	47.48	**65.92**	**90.81**	**66.06**	**68.67**
PISCO (Louis et al., 2025)	x128	68.00	**46.00**	61.00	89.00	55.00	63.80
SD-Cache (Ours)	x128	**71.84**	44.75	**63.73**	**89.58**	**57.41**	**65.46**



(a) Performance on RA-QA.

(b) KV Cost at Equal Match Score.

Figure 4: Performance comparison on NQ in RA-QA benchmark (Rau et al., 2024a) under (4a) four compression settings (e.g., $1/8$, $1/16$, $1/32$ and $1/128$) and (4b) the average relative change in KV budget at matched accuracy.

Questions (NQ) (Rau et al., 2024a) development set. As shown in Fig. 4a, SD-Cache's advantage over Attention Matching (Zweiger et al., 2026) grows as the compression ratio increases. At $16\times$ and $32\times$ compression, SD-Cache achieves accuracies of 65.74% and 62.8%, exceeding Attention Matching by 4.24 pp and 11.0 pp. At $128\times$ compression, the gap increases to 21.7 pp, with SD-Cache retaining 59.2% accuracy compared with 37.5% for Attention Matching. We further demonstrate the accuracy-memory trade-off between our method and Attention Matching (Zweiger et al., 2026) in its fastest configuration. We report Bjøntegaard-Delta rate (Bjontegaard, 2001) applied to KV-cache budgets (BD-KV) to measure the average relative change in KV budget at matched accuracy over the shared quality range, with negative values indicating memory savings. In Fig. 4b, we plot the relative KV budget difference at matched accuracy, where negative values indicate lower KV requirements for SD-Cache. Over the shared Match range of 59.18%-66.89%, SD-Cache achieves a BD-KV of -58.70%. Its memory advantage decreases as the target accuracy increases and reverses near the upper end of this range. Details about BD-KV are provided in Appendix A.2. These results indicate that SD-Cache is particularly effective under tight memory budgets, but it falls slightly behind Attention Matching at the largest evaluated KV budget. One possible explanation is that learned compression may lose fine-grained information that could otherwise be retained when more KV capacity is available. This motivates exploring an adaptive hybrid method that combines selective KV retention with learned compression under a fixed memory budget in the future.

Compaction Efficiency. We analytically estimate computational costs in TFLOPs (Louis et al., 2025), separating context preparation (*Compact*, including context pre-fill) from subsequent inference; *Total* sums both stages, shown in Tab. 3. On Mistral-7B, SD-Cache re-

Table 2: Average accuracy (%) across the 13 RULER-4K tasks (Hsieh et al., 2024) under different KV cache reduction ratios, using Llama 3.1-8B (Grattafiori et al., 2024) as the base model. A reduction ratio of 0% denotes the full, uncompressed KV cache.

Model	Method	KV cache reduction					
		0.00%	75.00%	90.00%	93.75%	96.88%	98.44%
Llama 3.1-8B	PyramidKV (Cai et al., 2024)	95.30	43.60	23.10	16.47	8.64	0.06
	SnapKV (Li et al., 2024)	95.30	63.20	43.40	16.45	8.60	0.06
	TOVA (Oren et al., 2024)	95.30	63.30	37.50	20.28	6.83	3.70
	ExpectedAttention (Devoto et al., 2025)	95.30	75.90	30.60	15.43	10.54	7.87
	AM-Fast (Zweiger et al., 2026)	95.30	78.82	46.95	33.75	22.73	12.55
	KeyDiff (Park et al., 2026)	95.30	64.44	61.10	54.50	34.69	16.58
	IndexMem (Yang et al., 2026)	95.30	78.30	55.20	–	–	–
	SD-Cache (Ours)	95.30	**79.52**	**66.11**	**55.44**	**55.17**	**53.78**

quires 1.79 TFLOPs for inference after preparation, while its compaction overhead increases the total cost to 13.95 TFLOPs, compared with 11.82 for Full KV and 12.88 for PISCO. On Llama-3.1-8B with a 4K context, SD-Cache requires 80.70 TFLOPs in total, substantially less than AM-fast (172.18), but more than SnapKV (68.53). For more results, please refer to Appendix B.4. These estimates quantify the additional computation required to construct SD-Cache's compact KV representations. When the prepared context can be reused across queries, this preparation cost can be amortised, yielding an average cost of $I + C/N_q$ for N_q queries sharing the same context.

Table 3: Analytical computational cost.

Method	Computational cost (TFLOPs) ↓		
	Inference	Compact	Total
Mistral-7B / RAG-QA			
FullKV	1.79	10.03	11.82
AutoCompressor	2.98	10.42	13.40
PISCO	2.89	9.99	12.88
SD-Cache (Ours)	1.79	12.16	13.95
Llama-3.1-8B / 4K context			
FullKV	2.74	65.97	68.71
SnapKV	2.42	66.11	68.53
AM-fast	2.29	169.90	172.18
SD-Cache (Ours)	2.43	78.27	80.70

4.2 Results on RULER Benchmark

Tab. 2 reports the average accuracy across the 13 RULER tasks with a 4096 token context using Llama-3.1-8B (Grattafiori et al., 2024). SD-Cache achieves the highest reported accuracy at every non-zero cache reduction ratio. At 75% and 90% reduction, it surpasses the second-best methods by 0.7 pp and 5.01 pp, respectively. Its advantage is particularly pronounced at the two highest reduction ratios: at 96.88% and 98.44%, SD-Cache achieves accuracies of 55.17% and 53.78%, exceeding KeyDiff (Park et al., 2026), the strongest baseline with available results in these settings, by 20.48 pp and 37.20 pp, respectively. Furthermore, SD-Cache's accuracy decreases by only 1.66 pp as the cache reduction increases from 93.75% to 98.44%. These results indicate that SD-Cache preserves useful information for long-context tasks even under severe KV cache compression. See Appendix B.2 for results on additional backbones. These results highlight the value of learning to compress KV states while preserving task performance under tight memory budgets. Hence, a promising direction for future work is to combine learned consolidation with training-free token selection to draw on the complementary strengths of both approaches.

4.3 Ablation Studies and Analysis

Contributions of Individual Components. We first analyse the contributions of SD-Cache's key components on Natural Questions (NQ) within the RA-QA benchmark, using Qwen3-4B-Instruct-2507[3] as the backbone at a fixed retention ratio of $1/16$. A simple baseline that compresses text into soft tokens achieves a score of 61.72%. The anchor pooling module (APM) variant improves this score to 64.44%, yielding a gain of 2.72 pp. Incorporating state-guided correction (SGC) further increases the score to 65.74%, providing an additional gain of 1.3 pp. Full SD-Cache improves upon the soft-token baseline by 4.02 pp, highlighting the incremental contributions of APM compression and SGC.

[3] https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507

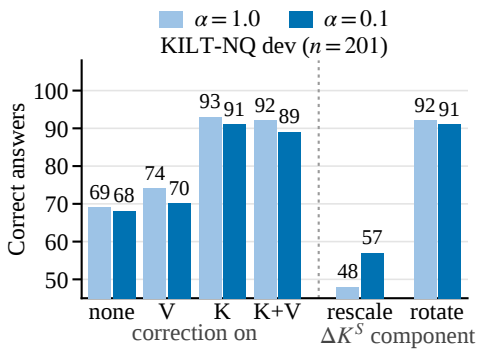


Figure 5: State-guided correction ablations: correction targets (left) and parallel ("rescale") versus orthogonal ("rotate") key-correction components (right).

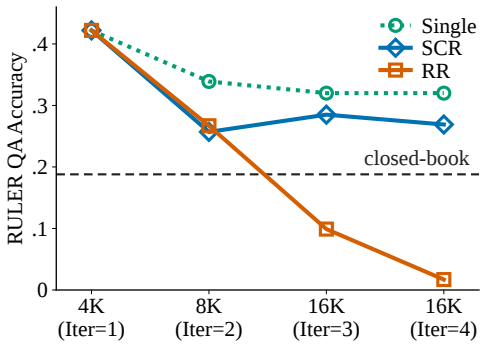


Figure 6: Length generalisation on RULER QA with $M = 410$ slots. RR: recurrent rolling; SCR: skip-connected rolling. "Iter" denotes the number of writes.

Length Generalisation under a Fixed Memory Budget. We evaluate length generalisation on RULER QA by increasing the context length from 4K to 8K and 16K while keeping the memory budget fixed at M=410 slots. Starting from a model trained on 4K-token contexts with a single write, we compare current single write setting with two streaming strategies: *recurrent rolling (RR)*, which passes the updated slots to the next write, and *skip-connected rolling (SCR)*, which adds skip connections between subsequent writes. The two strategies perform similarly at 8K with two writes (26.7% vs. 25.7%), but diverge at 16K. RR degrades sharply and often produces gibberish responses, with accuracy falling to 9.9% after three updates and 1.7% after four. In contrast, SCR retains 28.5% and 26.9%, respectively. These results show that SD-Cache supports iterative writing, but maintaining accuracy across repeated updates remains challenging, motivating future work on more robust update strategies and training data tailored to multi-step writing.

Analysis of State-Guided Correction. To understand where SGC helps, we ablate guidance strength α and correction target using models trained on RA-QA (Louis et al., 2025) and evaluated on 201 NQ samples. We divide the experiments into two groups (Fig. 5): 1) applying correction to K, V, both, or neither; and 2) retaining only the component of the applied key correction parallel ("rescale") or orthogonal ("rotate") to the provisional slot key $\tilde{K}$, while keeping value correction unchanged. The parallel component scales the existing key, whereas the orthogonal component adjusts its direction. Most of the performance gain comes from key correction (ΔK^S). The "rotate" arm preserves this gain, while "rescale" performs worse than disabling correction. These results suggest that SGC improves query-slot matching primarily by adjusting slot-key directions.

5 Discussion and Conclusion

We introduced SD-Cache, a learnable memory consolidation framework that maintains a budget-constrained KV cache through selective and state-guided updates. We then proposed an anchor-pooling module that compresses long KV sequences into a fixed number of memory slots while preserving key-value correspondence. We further introduced state-guided correction to refine the pooled KV representations using global key-value associations accumulated from the original KV stream, supporting access to relevant information during query decoding. Our method demonstrated competitive performance and computational cost compared with baselines on retrieval-augmented question answering and long-context benchmarks, covering tasks such as question answering, needle-in-a-haystack, variable tracing and aggregation. These results suggest that SD-Cache can be a valuable step towards learnable memory consolidation for LLMs.

Limitations and Future Work. A current limitation is that our method does not explicitly optimise layer-wise memory allocation, and maintaining performance across multi-step streaming updates remains challenging. Future work will explore adaptive allocation and more sophisticated reward functions to align memory updates with downstream task requirements. We will also investigate hybrid approaches combining learned consolidation with training-free token selection, alongside more robust update strategies and training data tailored to multi-step writing.

AI USE STATEMENT

In this work, we used generative AI tools to implement and edit Python code for the proposed method, create or modify scientific figures, prototypes of methodology diagrams, visualise examples and edit the manuscript to improve its clarity and readability. We also used AI tools to retrieve and identify related works. We did not use generative AI tools to generate synthetic datasets or to develop theoretical models. No other required-disclosure uses were applicable to this work. All AI-assisted outputs, including text, code, and figures, were reviewed by the authors, and the generated or modified code was tested for correctness. The authors take full responsibility for the final content of this work, including all text, claims, code, and other artefacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

We support reproducibility by providing a detailed description of SD-Cache and its components in Section 3. The appendix documents the model training details, hyperparameters and additional evidence used in our experiments. We also report the backbone models, datasets, memory budgets, and evaluation metrics for each benchmark. We will publicly release the source code, configuration files, and training and evaluation scripts in the camera-ready version.

REFERENCES

Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

Gisle Bjontegaard. Calculation of average psnr differences between rd-curves. *ITU SG16 Doc. VCEG-M33*, 2001.

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

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-lora: Learning to instantly internalize contexts. *arXiv preprint arXiv:2602.15902*, 2026.

Vivek Chari, Guanghui Qin, and Benjamin Van Durme. Kv-distill: Nearly lossless learnable context compression for llms. *arXiv preprint arXiv:2503.10337*, 2025.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. xrag: Extreme context compression for retrieval-augmented generation with one token. *Advances in Neural Information Processing Systems*, 37:109487–109516, 2024.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, 2023.

Zina Chkirbene, Ridha Hamila, Ala Gouissem, and Unal Devrim. Large language models (llm) in industry: A survey of applications, challenges, and trends. In *2024 IEEE 21st International Conference on Smart Communities: Improving Quality of Life using AI, Robotics and IoT (HONET)*, pp. 229–234. IEEE, 2024.

Euirim Choi. Goodwiki dataset. `https://www.github.com/euirim/goodwiki`, September 2023.

Ronald L Davis and Yi Zhong. The biology of forgetting—a perspective. *Neuron*, 95(3):490–503, 2017.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: Kv cache compression by estimating attention from future queries distribution. *arXiv preprint arXiv:2510.00636*, 2025.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Atri Rudra, James Y Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study. In *International Conference on Learning Representations*, volume 2026, pp. 42642–42687, 2026.

Frederick N Fritsch and Ralph E Carlson. Monotone piecewise cubic interpolation. *SIAM Journal on Numerical Analysis*, 17(2):238–246, 1980.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*, 2023.

Cameron Gordon, Yiping Ji, Hemanth Saratchandran, Paul Albert, and Simon Lucey. SineloraΔ: Sine-activated delta compression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 21333–21342, 2026.

Stephen Gould and Anton van den Hengel. Back from the future: Key-value cache management by counter-causal surprise. *arXiv preprint arXiv:2607.27600*, 2026.

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.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. *CoRR*, abs/1609.09106, 2016. URL http://arxiv.org/abs/1609.09106.

Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated deltanet-2: Decoupling erase and write in linear attention. *arXiv preprint arXiv:2605.22791*, 2026.

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

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pp. 4651–4664. PMLR, 2021.

Albert Qiaochu Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *ArXiv*, abs/2310.06825, 2023. URL https://api.semanticscholar.org/CorpusID:263830494.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Tiziano Labruna, Giovanni Bonetta, and Bernardo Magnini. Task-oriented dialogue systems through function calling. In *Proceedings of the 15th International Conference on Recent Advances in Natural Language Processing-Natural Language Processing in the Generative AI Era*, pp. 614–622, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

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.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *ArXiv*, abs/1711.05101, 2017. URL `https://api.semanticscholar.org/CorpusID:3312944`.

Maxime Louis, Hervé Déjean, and Stéphane Clinchant. Pisco: Pretty simple compression for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 15506–15521, 2025.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 9802–9822, 2023.

Charles O'Neill, Alex Sandomirsky, Harry Partridge, Mudith Jayasekara, and Max Kirkby. Still: Amortized kv cache compaction in a single forward pass. *arXiv preprint arXiv:2606.07878*, 2026.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 18724–18741, 2024.

Junyoung Park, Dalton Jones, Matthew Morse, Raghavv Goel, Mingu Lee, and Christopher Lott. Keydiff: Key similarity-based kv cache eviction for long-context llm inference in resource-constrained environments. *Advances in Neural Information Processing Systems*, 38:5983–6019, 2026.

Guanghui Qin, Corby Rosset, Ethan Chau, Nikhil Rao, and Benjamin Van Durme. Dodo: Dynamic contextual compression for decoder-only lms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9961–9975, 2024.

David Rau, Hervé Déjean, Nadezhda Chirkova, Thibault Formal, Shuai Wang, Stéphane Clinchant, and Vassilina Nikoulina. Bergen: A benchmarking library for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7640–7663, 2024a.

David Rau, Shuai Wang, Hervé Déjean, and Stéphane Clinchant. Context embeddings for efficient answer generation in rag. *arXiv preprint arXiv:2407.09252*, 2024b.

Larry R Squire, Lisa Genzel, John T Wixted, and Richard G Morris. Memory consolidation. *Cold Spring Harbor perspectives in biology*, 7(8):a021766, 2015.

Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. Asqa: Factoid questions meet long-form answers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 8273–8288, 2022.

Xin Tian, Yingzhan Lin, Mengfei Song, Siqi Bao, Fan Wang, Huang He, Shuqi Sun, and Hua Wu. Q-tod: A query-driven task-oriented dialogue system. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pp. 7260–7271, 2022.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

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.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in neural information processing systems*, 37:115491–115522, 2024.

Xintong Yang, Hao Gu, Binxing Xu, Lujun Li, Bei Liu, Jiacheng Liu, Qiyuan Zhu, Yike Guo, and Sirui Han. Indexmem: Learned kv-cache eviction with latent memory for long-context llm inference. *arXiv preprint arXiv:2605.25475*, 2026.

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*, pp. 2369–2380, 2018.

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.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast kv compaction via attention matching. *arXiv preprint arXiv:2602.16284*, 2026.

A Additional Details

A.1 Implementation Details

Our consolidation network follows the Perceiver architecture: the M slot queries cross-attend to the key-value tensor through 8 pre-norm Perceiver like blocks, each with 8 cross-attention heads and a $4\times$ FFN expansion, operating at the backbone's key-value width. A single anchor pooling module (APM) is shared across all layers. Each layer also has a lightweight projector comprising copies of its frozen key and value projections, each initialised from the corresponding original weights and augmented with rank-16 LoRA adapters ($\alpha = 32$). This amounts to approximately 142M trainable parameters, 1.7% of an 8B backbone, which remains frozen throughout. We set fixed correction strengths $\alpha_v = \alpha_k = 0.1$. The computed memory slots are kept in pre-RoPE space; the i-th slot is rotated at the position of the anchor token it was initialised from, and the query and generated tokens are positioned directly after the M slots. We train for five epochs using AdamW (Loshchilov & Hutter, 2017) with a learning rate of 2×10^{-4}, a linear learning rate schedule with 3% warmup, and bf16 precision. Depending on the context length, we use effective batch sizes of 128 or 64, with corresponding per-device batch sizes of 4 or 1 and gradient accumulation. Training runs under DeepSpeed ZeRO-2 on four RTX 6000 Ada GPUs; the per-layer trunk calls are gradient-checkpointed, which leaves the computed gradients unchanged and is what fits the largest memory budget into 48 GB. For the experiments, we do not include KV-Distill (Chari et al., 2025) in our RULER (Hsieh et al., 2024) comparison because no public implementation was available at the time of our experiments, making faithful reproduction difficult. We also omit STILL (O'Neill et al., 2026) because its peak training memory requirements (e.g., 8 $\times$ NVIDIA H200 GPUs) exceeded our available hardware budget.

Table 4: Average accuracy (%) across the 13 RULER-4K tasks (Hsieh et al., 2024) under different KV cache reduction ratios, using Qwen3-8B (Yang et al., 2025) as the base model. A reduction ratio of 0% denotes the full, uncompressed KV cache.

Model	Method	KV cache reduction				
		0.00%	90.00%	93.75%	96.88%	98.44%
Qwen3 8B	PyramidKV (Cai et al., 2024)	95.30	19.50	15.20	9.35	2.31
	SnapKV (Li et al., 2024)	95.30	19.20	15.22	9.25	2.33
	TOVA (Oren et al., 2024)	95.30	24.70	11.00	4.09	3.60
	KeyDiff (Park et al., 2026)	95.30	37.90	25.40	14.61	8.90
	IndexMem (Yang et al., 2026)	95.30	46.20	-	-	-
	SD-Cache (Ours)	95.30	**61.55**	**59.80**	**57.32**	**56.39**

Table 5: Average accuracy (%) across the 13 RULER-4K tasks (Hsieh et al., 2024) under different KV cache reduction ratios, using Mistral-7B (Jiang et al., 2023) as the base model. A reduction ratio of 0% denotes the full, uncompressed KV cache.

Model	Method	KV cache reduction				
		0.00%	90.00%	93.75%	96.88%	98.44%
Mistral 7B-v0.3	PyramidKV (Cai et al., 2024)	92.10	19.90	13.71	8.43	4.10
	SnapKV (Li et al., 2024)	92.10	19.80	13.75	8.40	3.59
	TOVA (Oren et al., 2024)	92.10	29.20	14.93	6.67	3.77
	KeyDiff (Park et al., 2026)	92.10	-	45.26	26.61	14.08
	IndexMem (Yang et al., 2026)	92.10	41.50	-	-	-
	SD-Cache (Ours)	92.10	**52.17**	**47.12**	**44.41**	**37.23**

A.2 BD-KV Computation

We adapt the Bjøntegaard-Delta rate metric (Bjontegaard, 2001) by using the nominal retained KV fraction as the rate variable and Match as the quality variable. While BD-Rate is typically applied in image and video compression, recent work has argued for its use to evaluate LLM compression methods (Gordon et al., 2026). For each method m, we construct Piecewise Cubic Hermite Interpolating Polynomial (PCHIP (Fritsch & Carlson, 1980)) interpolants $g_m(q)$ through the measured points $(q_i, \log_{10} r_i)$, where q_i denotes Match and r_i denotes the retained KV fraction. The relative KV difference at a target quality q is

$$\Delta_{\mathrm{KV}}(q) = 100 \left(10^{g_{\mathrm{ours}}(q) - g_{\mathrm{base}}(q)} - 1 \right). \tag{13}$$

We restrict evaluation to the intersection of the observed quality ranges:

$$q_{\min} = \max \left\{ \min_i q_i^{\mathrm{ours}}, \min_j q_j^{\mathrm{base}} \right\}, \tag{14}$$

$$q_{\max} = \min \left\{ \max_i q_i^{\mathrm{ours}}, \max_j q_j^{\mathrm{base}} \right\}. \tag{15}$$

For $q_{\max} > q_{\min}$, we compute

$$\overline{\delta} = \frac{1}{q_{\max} - q_{\min}} \int_{q_{\min}}^{q_{\max}} \left[g_{\mathrm{ours}}(q) - g_{\mathrm{base}}(q) \right] \mathrm{d}q, \tag{16}$$

$$\text{BD-KV} = 100 \left(10^{\overline{\delta}} - 1 \right). \tag{17}$$

For SD-Cache versus attention matching, the shared interval is $[59.18, 66.89]\%$ Match, yielding BD-KV $= -58.70\%$. Negative values indicate a lower KV budget for SD-Cache at equal quality.

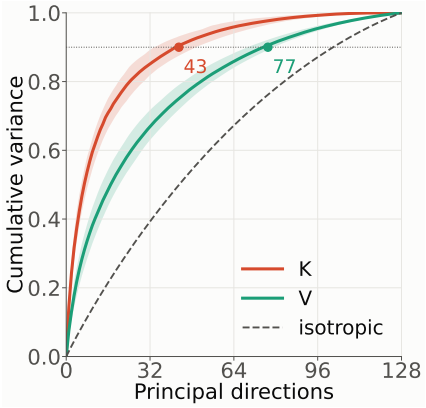


Figure 7: Visualisation of cumulative explained variance for mean-centred pre-RoPE keys (K) and values (V) in Qwen3-4B Instruct (Yang et al., 2025). Curves are averaged over 12 documents per layer-head pair, where solid lines and shaded bands show the median and inter-quartile range across 288 pairs. The dashed curve shows an isotropic Gaussian reference of the same size. Dots mark 90% explained variance for K and V.

The solid curve shows $\Delta_{\mathrm{KV}}(q)$, while the horizontal dashed line indicates BD-KV. This summary averages log-rate differences before converting back to a percentage; it is therefore not the arithmetic mean of the plotted percentage differences. We use the FAU-LMS `bjontegaard` [4] implementation and do not extrapolate beyond the shared quality interval. KV cost is measured by the nominal token/slot budget, rather than total device memory; model weights and auxiliary state are excluded.

B Additional Results

B.1 Low-Rank Structure within KV Heads

To examine the low-rank structure that motivates the anchor pooling module (APM), we analyse token variation within each KV head of Qwen3-4B Instruct. Each document contains 768 tokens, with the initial attention-sink position removed. We extract pre-RoPE keys from `k_norm` and values from `v_proj` across all 36 layers and 8 KV heads per layer ($d = 128$). For each document and head, we subtract the mean token vector from each 767×128 representation matrix and apply singular value decomposition (SVD) to the residuals. We accumulate the squared singular values in descending order and normalise by their total sum to obtain the cumulative explained variance. Centring is performed separately for each document because slot pooling summarises one context at a time. For each layer-head pair, we first average the curves over the 12 documents, then report the median and inter-quartile range across the 288 pairs. An isotropic Gaussian matrix of the same size undergoes the same centring and SVD steps as a reference. As shown in Figure 7, the median curves reach 90% explained variance at 43 directions for keys and 77 for values, compared with 103 for the isotropic reference. The concentrated key variation supports the motivation for APM, while the higher effective rank of values suggests that preserving their variation may require more capacity.

B.2 Additional Results on RULER Benchmark

We further evaluate SD-Cache on RULER-4K (Hsieh et al., 2024) using two additional backbones: Qwen3-8B (Yang et al., 2025) and Mistral-7B-v0.3 (Jiang et al., 2023). Tab. 4 and Tab. 5 report the average accuracy across the 13 RULER tasks with a 4096-token context. SD-Cache achieves the highest reported accuracy at every non-zero cache reduction ratio on both backbones.

On Qwen3-8B (Yang et al., 2025), SD-Cache outperforms IndexMem (Yang et al., 2026) at 90% reduction and KeyDiff (Park et al., 2026) at 93.75% reduction, the strongest available baselines in these settings, by 15.35 pp and 34.40 pp, respectively. At the two highest reduction ratios of 96.88%

[4] https://github.com/FAU-LMS/bjontegaard

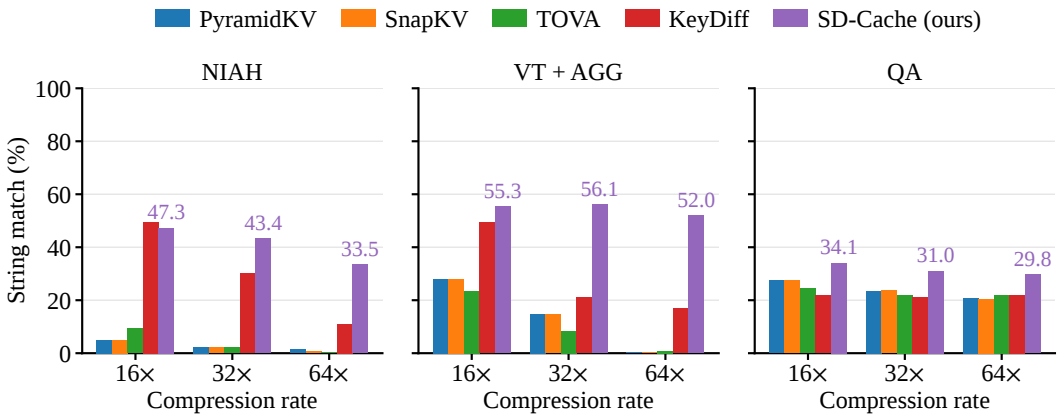


Figure 8: RULER-4K (Hsieh et al., 2024) performance by task group on Mistral-7B-Instruct-v0.3 (Jiang et al., 2023) under different KV cache compression rates.

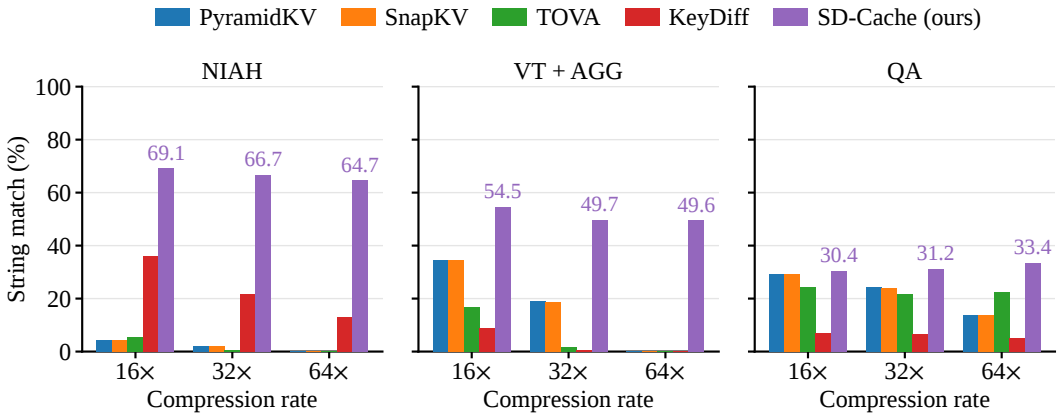


Figure 9: RULER-4K (Hsieh et al., 2024) performance by task group on Qwen3-8B (Yang et al., 2025) under different KV cache compression rates.

and 98.44%, SD-Cache achieves accuracies of 57.32% and 56.39%, exceeding KeyDiff by 42.71 pp and 47.49 pp, respectively. Its accuracy decreases by only 3.41 pp as the reduction increases from 93.75% to 98.44%, indicating robustness to increasingly aggressive compression.

On Mistral-7B-v0.3 (Jiang et al., 2023), SD-Cache achieves accuracies of 52.17% and 47.12% at 90% and 93.75% reduction, outperforming the same respective baselines by 10.67 pp and 1.86 pp. At 96.88% and 98.44% reduction, it achieves 44.41% and 37.23% accuracy, surpassing KeyDiff by 17.8 pp and 23.15 pp, respectively. These results suggest that SD-Cache preserves useful information under severe KV cache compression across different LLM backbones.

Task Group Analysis. We group the RULER-4K tasks into four categories: needle-in-a-haystack (NIAH), question answering (QA), variable tracking (VT), and aggregation (AGG), where AGG comprises common words extraction (CWE) and frequent words extraction (FWE). Figures 9 and 8 present results on Qwen3-8B and Mistral-7B-v0.3, respectively, with VT and AGG reported jointly as VT+AGG. While KeyDiff shows competitive results on NIAH at 16× compression, SD-Cache outperforms all evaluated baselines across all three reported groups at 32× and 64× compression on both backbones. These results show that SD-Cache is generally more robust to aggressive KV cache compression than the other baselines.

B.3 Ablation of the Novelty-Guided Write

We evaluate SD-Cache trained on RA-QA (Louis et al., 2025) with a Qwen3-4B-Instruct (Yang et al., 2025) backbone on 948 ASQA dev questions at a 16× compression ratio, using Match as

Table 6: Estimated computational cost of Llama-3.1-8B-Instruct under the RULER compression budgets. We use a controlled workload of 4096 context tokens, 128 question tokens and 32 output tokens, with a batch size of one and no separate system prefix. These costs are not averaged over RULER evaluation samples. †ExpectedAttention uses the AdaKV masking implementation associated with the archived accuracy results, which retains the full physical KV cache. Its costs are rounded analytical upper-bound estimates for successful execution paths with 1000 search iterations per call. ‡AM-fast uses off-policy repeat queries and uniform budgets. Its costs include reference-query preparation and analytically estimated QR solver costs.

		Computational cost (TFLOPs) ↓		
Target context-slot reduction	Method	Inference	Compact	Total
0.00%	Full KV	2.74	65.97	68.71
75.00%	SnapKV	2.48	66.11	68.59
	KeyDiff	2.48	65.97	68.45
	ExpectedAttention + AdaKV†	2.78	70.65	73.43
	AM-fast‡	2.35	188.75	191.10
	SD-Cache (Ours)	2.50	83.46	85.96
90.00%	SnapKV	2.43	66.11	68.54
	KeyDiff	2.43	65.97	68.40
	ExpectedAttention + AdaKV†	2.78	70.65	73.43
	AM-fast‡	2.30	172.11	174.41
	SD-Cache (Ours)	2.45	79.39	81.84
93.75%	SnapKV	2.42	66.11	68.53
	KeyDiff	2.42	65.97	68.39
	ExpectedAttention + AdaKV†	2.78	70.65	73.43
	AM-fast‡	2.29	169.90	172.18
	SD-Cache (Ours)	2.43	78.27	80.70
96.88%	SnapKV	2.41	66.11	68.52
	KeyDiff	2.41	65.97	68.38
	ExpectedAttention + AdaKV†	2.78	70.65	73.43
	AM-fast‡	2.27	168.64	170.91
	SD-Cache (Ours)	2.42	77.30	79.73
98.44%	SnapKV	2.40	66.11	68.51
	KeyDiff	2.40	65.97	68.37
	ExpectedAttention + AdaKV†	2.78	70.65	73.43
	AM-fast‡	2.27	168.21	170.48
	SD-Cache (Ours)	2.42	76.81	79.23

the metric. Without retraining, we test three inference-time variants: removing the novelty bias, shuffling novelty scores across tokens within each passage, and replacing the slot gate with its layer-wise mean. To evaluate the validity of the novelty-guided write module, we additionally design a shuffling experiment that preserves the distribution of novelty scores but breaks their correspondence with the tokens, thereby testing whether assigning higher scores to the appropriate tokens matters. The full model achieves 71.5% Match, compared with 69.5% without the novelty bias, 67.2% with shuffled novelty scores, and 68.1% with a static gate. These results support the contributions of both token-specific novelty information and input-dependent slot gating.

B.4 Computational-cost estimation

In Tab. 6, we additionally report analytical estimates of computational cost under the RULER compression budgets. We count each multiply–accumulate as two FLOPs and treat attention products as dense rectangular matrix multiplications. The estimates cover major matrix operations and follow each method's input formatting and generation procedures. They are not GPU measurements or dataset-wide averages. "Inference" (I) denotes question pre-fill and answer generation after preparation; "Compact" (C) denotes context or representation preparation, including context pre-fill; and "Total" (T) denotes first-use cost. For a fixed context whose prepared representation can be reused

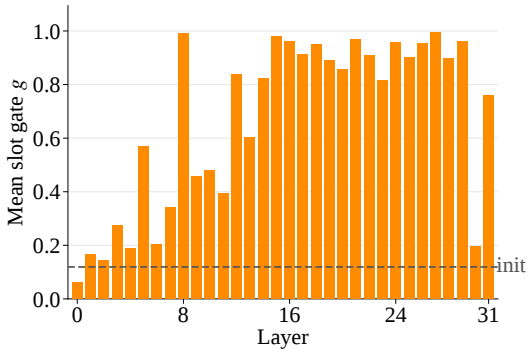


Figure 10: Distribution of estimated slot update strengths.

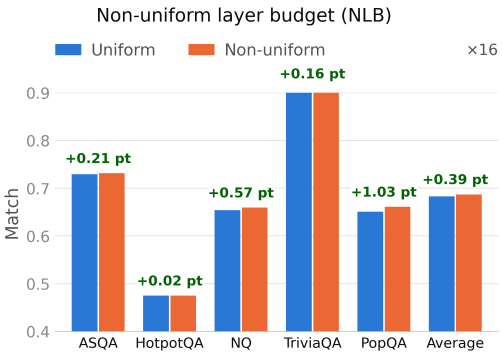


Figure 11: Uniform vs. non-uniform layer budgets.

unchanged across questions, the amortised cost per question over R questions is $I + C/R$. This accounting does not imply that persistent reuse has been implemented for every method. For Llama-3.1-8B-Instruct, the KV cache of the 4096-token context occupies 0.50 GiB (128 KiB per token in bf16). Reducing the context slots by 75%, 90%, 93.75%, 96.88% and 98.44% shrinks it to 0.125, 0.050, 0.031, 0.016 and 0.008 GiB, respectively, i.e. 4× to 64× smaller. The only exception is ExpectedAttention + AdaKV, whose masking implementation retains the full physical cache and therefore stays at 0.50 GiB.

B.5 Non-uniform Layer Budget

To visualise the contribution of the slot update strength at each layer, we examine the learned gate values $\mathbf{g}$ in Eq. 8 across all layers. The gate magnitudes exhibit a non-uniform distribution, with several intermediate layers showing stronger updates than the early and final layers. This suggests that SD-Cache learns to vary the degree of memory update across different layers. Motivated by this layer-wise heterogeneity, we further investigate whether the memory budget itself can be distributed non-uniformly across layers.

Our SD-Cache design naturally supports non-uniform layer-wise KV cache budgets, allowing different numbers of cache slots to be assigned to different layers. Specifically, we divide the layers into two early groups (layers 0-7 and 8-11), a middle group (layers 12-22), and a late group (layers 23-31), allocating 15.6%, 13.1%, 51.6%, and 19.7% of the total KV cache budget to these four groups, respectively. This allocation concentrates more capacity in the middle layers while preserving the same total budget as the uniform baseline. As shown in Figure 11, non-uniform allocation yields modest but consistent improvements across all five datasets, increasing the average Match score by 0.39 pp. The largest gains are observed on PopQA (Mallen et al., 2023) and NQ (Kwiatkowski et al., 2019), with improvements of 1.03 pp and 0.57 pp, respectively. These results highlight the flexibility of SD-Cache in supporting non-uniform layer-wise budgets, while leaving the exploration of more effective budget allocation strategies to future work.

B.6 Memory Updates as the Context Changes

We use short fictional stories to examine whether SD-Cache's answers reflect changes in the context. Each story introduces a fact and later updates it, with unrelated documents interleaved. We first write background documents that do not contain the answer into a fixed memory of $M{=}40$ slots. The first story chunk contains background information, the initial fact and two unrelated documents; each subsequent chunk contains one update and one unrelated document. All chunks are written into the same slots. After each write, the frozen decoder answers the question using only the current memory. Figure 12 shows the text written at each step and the corresponding decoded answer. This example illustrates SD-Cache's ability to adaptively update its memory as the context changes.

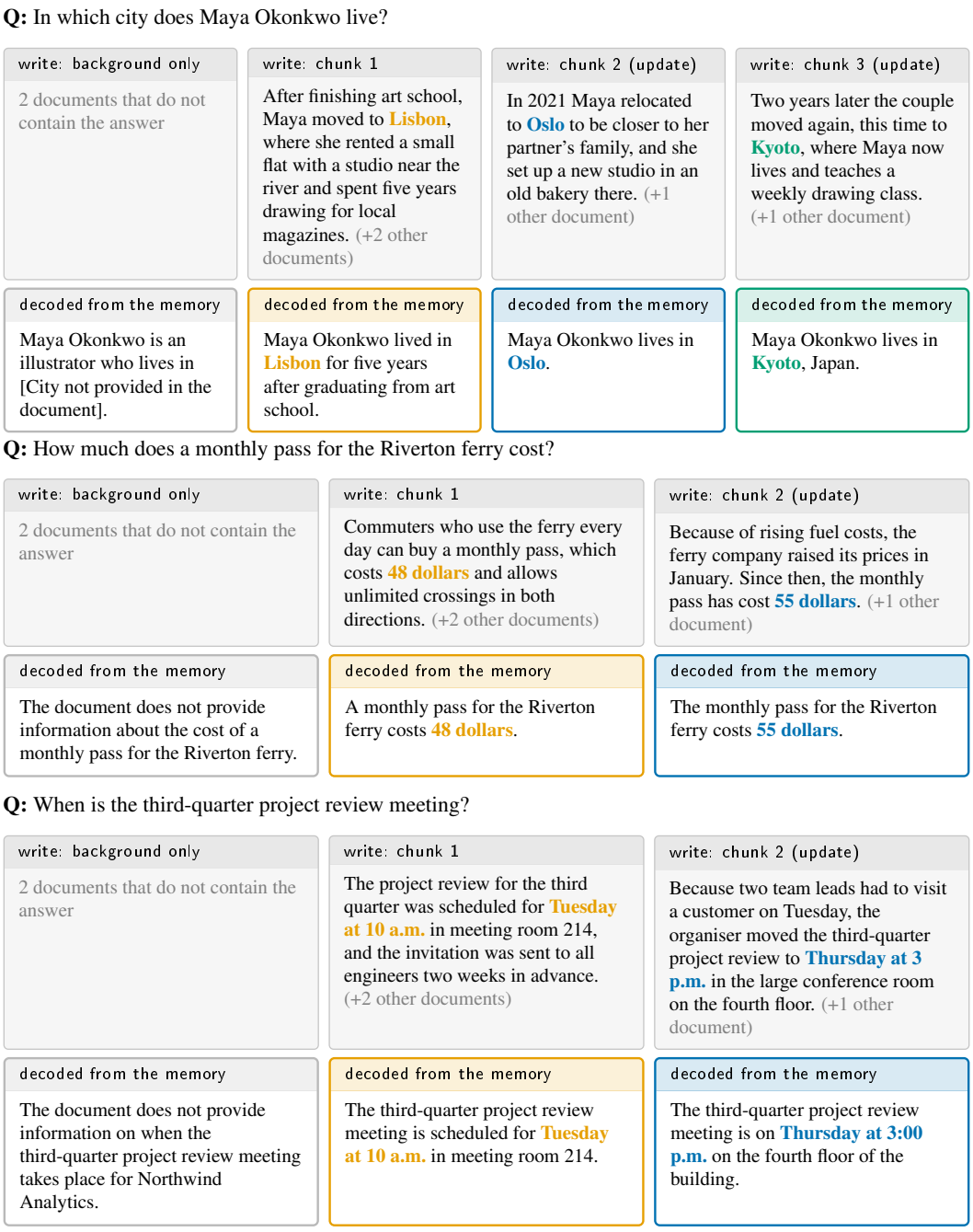


Figure 12: Visualisation of decoding from memory as the context changes. The top row shows the text written into memory at each step, with unrelated documents indicated in grey. The bottom row shows the first sentence of the answer generated by greedy decoding after each write, reproduced verbatim. Colours indicate successive states.

B.7 Consolidated Memory Visualisation

To examine what SD-Cache stores and how the frozen decoder uses it, we trace memory writes and decoder reads back to the context tokens. For *WRITE*, we sum each token's contribution across slots using the anchor weights $1 - g$ and pooling weights gW in Eq. (8). For *READ*, we collect the decoder's attention to the M slots while generating the answer tokens, map query heads to their

corresponding KV heads, and propagate the attention through the same pooling weights. As shown in Fig. 13 and Fig. 14, the READ maps place greater emphasis on answer-related content than the WRITE maps. These examples illustrate how SD-Cache consolidates information without knowing the question and allows the decoder to selectively access relevant content during decoding.

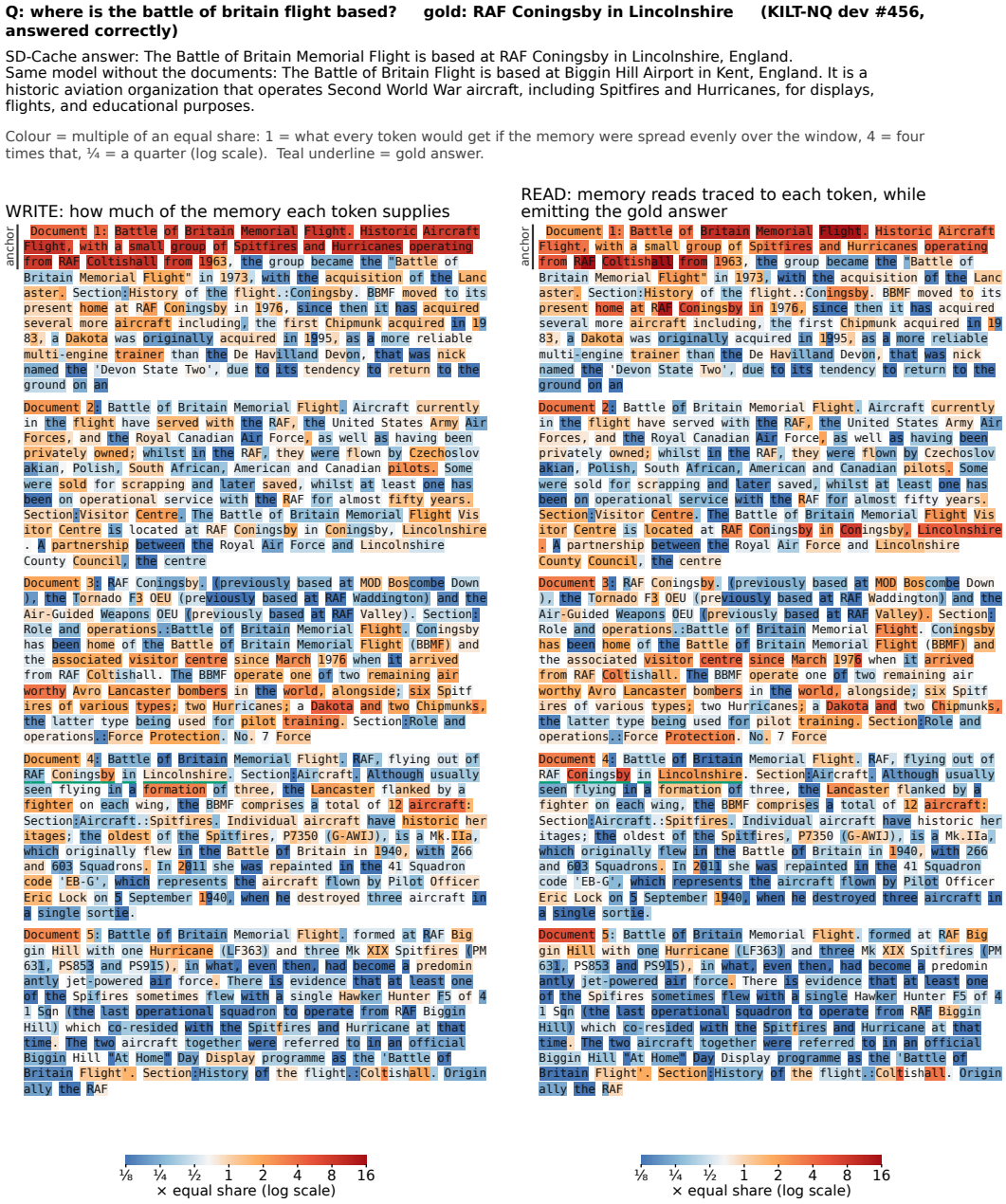


Figure 13: Visualisation of the information SD-Cache stores in memory and reads when answering question #456 from the KILT-NQ dev set.

Q: who plays the woodsman in over the garden wall? gold: Christopher Lloyd (KILT-NQ dev #1443, answered correctly)

SD-Cache answer: Christopher Lloyd voices the Woodsman character in the animated miniseries "Over the Garden Wall."
Same model without the documents: The Woodsman in "Over the Garden Wall" is voiced by Elijah Wood.

Colour = multiple of an equal share: 1 = what every token would get if the memory were spread evenly over the window, 4 = four times that, ¼ = a quarter (log scale). Teal underline = gold answer.



Figure 14: Visualisation of the information SD-Cache stores in memory and reads when answering question #1443 from the KILT-NQ dev set.