ATTUNER: RECOMPUTATION-FREE KV CACHE REUSE VIA QUERY-SIDE ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents repeatedly load reusable content, such as skills, documents, and memory entries, into the current context. Re-encoding this content for every request wastes computation. Position-independent caching (PIC) alleviates this by encoding each artifact independently and reusing its key-value (KV) states at arbitrary positions, but it incurs a quality loss relative to full-context prefill. Existing methods repair this loss by restoring global position IDs or recomputing selected tokens. In this work, we isolate the source of the loss, finding that the positional mismatch has minor effect, and independently cached artifacts retain faithful representations: reading a provided artifact stays largely accurate, and performance degrades only when the model must select among multiple artifacts. Moreover, replacing PIC's attention scores with full-prefill scores recovers performance with the cached KV unchanged, localizing the failure to the attention rather than KV recomputation. Motivated by this, we propose ATTUNER, a query-side adaptation method that learns to read a frozen artifact cache. ATTUNER inserts low-rank adapters into the query projections and is trained by distilling full-prefill distribution into the student. It trains fewer than 0.05% of the model parameters and, at inference, requires neither cache recomputation nor a full-context reference. On Qwen3-4B and Qwen3-8B across seven benchmarks covering skills, documents, memory, and code, ATTUNER substantially outperforms prior PIC baselines in both in-domain and out-of-domain settings, matches full-context prefill quality while providing up to 3.73× speedup.

1 INTRODUCTION

Large language model (LLM) agents repeatedly interact with external environments, assembling each request's context from reusable content such as skill descriptions, documents, and memory entries (Mei et al., 2025; Zhou et al., 2026; Luo et al., 2026; Yao et al., 2023). We refer to these reusable spans as *artifacts*. Such artifacts are stored persistently and loaded on demand at runtime (Su et al., 2026). Re-encoding them for every request wastes computation and raises time to first token (TTFT). Reusing key-value (KV) states avoids this cost (LI et al., 2025), but conventional prefix caching assumes reusable content occupies the same prefix position across requests (Zheng et al., 2024). Agentic workloads rarely satisfy this assumption: The artifacts to load, their order, and their positions are decided dynamically at runtime by retrieval, routing and planning.

Position-independent caching (PIC) addresses this limitation by encoding each artifact independently and loading cached blocks at request time (Hu et al., 2025). However, directly reusing independently constructed caches can incur a performance loss relative to full prefill (Yang et al., 2025c). Two mismatches arise in PIC. The first is positional: artifacts are encoded before their position is known, so their position indices do not match those used in full prefill. The second is contextual: independently encoded artifacts attend do not attend to preceding request-specific artifacts, so their KV states lack the cross-artifact conditioning present in full prefill (Yao et al., 2025). Existing methods mitigate these mismatches at runtime, either by restoring global positional embeddings (Liu et al., 2026a; Wang et al., 2026) or by recomputing selected tokens to approximate the original full-context KV states (Zhao et al., 2026; Yang et al., 2025c; Xie et al., 2026).

In this work, we examine these mismatches through controlled experiments (§3). We first find that the positional mismatch has little impact on task performance. To examine whether the cached ar-

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.

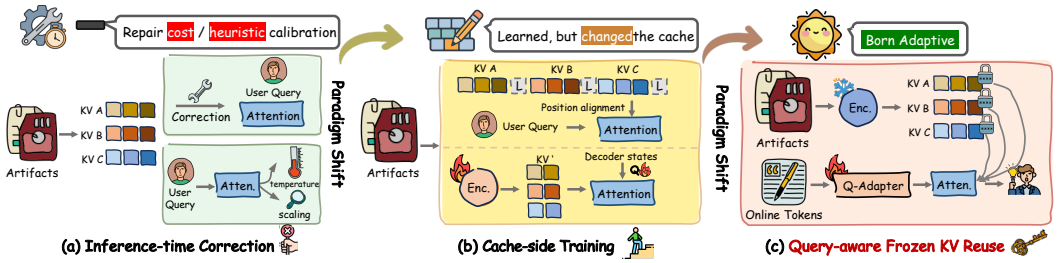


Figure 1: **Three approaches to reusable artifact caching.** (a) Inference-time adaptation repairs cached states or adjusts attention during inference. (b) Training-based adaptation modifies cache construction through learned encoders or linking tokens. (c) ATTUNER adapts online queries while preserving independently constructed artifact KV, learning to read the cache without rewriting it.

tifact representations themselves need to be recomputed, we separate the use of artifacts into two abilities: *artifact discovery*, which selects relevant artifacts from the candidates, and *evidence utilization*, which reads and integrates information from the selected artifacts. In the single-artifact setting, evidence utilization suffers little performance loss under PIC, indicating that the cached representation faithfully preserves its own content and remains readable by the model. The failure emerges in the multi-artifact setting: the model struggles to decide which artifact to read from. This contrast localizes the problem to how attention is allocated across artifacts rather than to the cached representations themselves. To test this directly, we replace PIC's online attention scores with prefix-matched full-prefill attention scores while keeping the cached artifact KV unchanged, and find that this recovers performance. These results establish an effective recovery path through *attention routing*, without reconstructing the cached artifact representations.

Motivated by this, we propose ATTUNER, a query-side adaptation method that learns to read a frozen artifact cache by correcting attention queries, without recomputing or modifying the cached KV. Figure 1 contrasts this query-side design with inference-time correction and cache-side training. ATTUNER inserts low-rank adapters (Hu et al., 2022) into the query projections, enabling online tokens to learn effective attention routing over cached keys. The adapters are trained through response-distribution distillation, with a full-prefill teacher supervising the PIC student. At inference, no Full-prefill reference or artifact-cache reconstruction is required; online states and newly produced KV continue to evolve under the adapted computation.

We evaluate ATTUNER on Qwen3-4B and Qwen3-8B across an extensive set of seven benchmarks covering diverse artifact types, measuring task quality and time to first token (TTFT). Further analyses examine training-data scale and show that the learned adaptation brings attention outputs closer to Full-context's on average while leaving cached artifact KV unchanged.

Our key contributions are threefold: (1) We identify the root cause of independent caching degradation: the cached representations are faithful, and the failure lies in the attention score computation. We confirm this with a routing experiment in which replacing PIC's attention scores with full-prefill scores recovers performance while the artifact KV remains unchanged. (2) We propose ATTUNER, a query-side adaptation method that learns to read a frozen artifact cache through low-rank updates to its query projections. It trains fewer than 0.05% of the model parameters and, at deployment, requires neither artifact-cache reconstruction nor an additional teacher run. (3) We extensively evaluate ATTUNER on Qwen3-4B and Qwen3-8B across skill, document and memory tasks. Compared with other PIC baseline, it improves both in-domain quality, out-of-domain generalization and efficiency: average scores rise by 5.13 and 7.77 points, matching the full-context prefill performance while providing up to 3.73× speedups.

2 BACKGROUND AND RELATED WORK

Position-Independent Caching An *artifact* A is a context span that can be reused verbatim across requests, such as a skill description, a document, or a memory entry. For exposition, consider a request with a prefix P, m artifacts, and a user query U:

$$X = P \oplus A_1 \oplus \cdots \oplus A_m \oplus U, \qquad Y \sim p_\theta(\cdot \mid X), \tag{1}$$

where $\oplus$ denotes concatenation and Y is the generated response.

Full prefill encodes each artifact under its preceding context in the current request. **Position-independent caching (PIC)** instead encodes each artifact independently under a fixed cache-building prefix P_{cache} and stores the resulting KV states for reuse. At request time, PIC assembles these cached blocks in prompt order without re-encoding the artifacts, computing states only for uncached prompt tokens and subsequently generated tokens.

Formally, let $\mathcal{C}_\theta(A \mid H)$ denote the keys and values across all layers obtained by encoding artifact A after context H, and let $A_{<j} = A_1 \oplus \cdots \oplus A_{j-1}$. The two cache constructions are

$$Z_j^{\text{Full}} = \mathcal{C}_\theta(A_j \mid P \oplus A_{<j}), \qquad Z_j^{\text{PIC}} = \mathcal{C}_\theta(A_j \mid P_{\text{cache}}). \tag{2}$$

Consequently, identical artifact text can yield different KV states under Full and PIC.

KV-Cache Reuse beyond Shared Prefixes. Prefix caching reuses computation when requests share an identical prefix, as in RadixAttention (Zheng et al., 2024), while Prompt Cache (Gim et al., 2024) extends this idea to reusable prompt modules with controlled placement. Position-independent reuse is more challenging because cached states are constructed without the request-specific context in which they will later be used. Existing methods largely address this mismatch through *inference-time repair*. One family selectively restores missing contextual dependencies by recomputing a small portion of the cached context. CacheBlend (Yao et al., 2025), CacheCraft (Agarwal et al., 2025), and EPIC (Hu et al., 2025) recompute selected tokens or artifact boundaries, while later methods use query relevance or learned selection to determine where repair is most useful (Zhou et al., 2025; Wang et al., 2026; Yang et al., 2026; 2025b; Liu et al., 2026b). Rather than recomputing the original states, another family directly corrects their use at inference time. APE (Yang et al., 2025d) calibrates attention over independently encoded blocks, while CacheSlide (Liu et al., 2026a) and AgentKVShift (Pandey et al., 2026) introduce position- or probe-based corrections to reused KV states. These approaches highlight two related sources of cache compatibility: *what information is stored in the cache* and *how that information is read after reuse*. The former is also reflected in work showing that prefill can encode intermediate conclusions into downstream states and that cached representations can be edited (Li, 2026). Complementary works focus on making such reuse practical through cache alignment, recomputation primitives, and serving support (Wang et al., 2025; Ordonez & Parnell, 2026), with extensions to multi-agent and multimodal settings (Ye et al., 2025; Bian et al., 2026; Zhao et al., 2025; Qin et al., 2025).

Learning for Cache Reuse. A complementary line of work learns cache-compatible behavior rather than repairing mismatches entirely at inference time. Some methods adapt the language model itself to operate over independently encoded blocks. Some methods learn cache-compatible behavior through model adaptation, including block-wise fine-tuning and learned cross-artifact aggregation (Ma et al., 2025; Lu et al., 2025; Yang et al., 2025c; Fang et al., 2026). Others instead learn alternative cache representations or interfaces (Zhao et al., 2026; Du et al., 2026). Full-context distillation provides another route, training independently constructed caches to approximate full-prefill behavior (Chen et al., 2026; Xie et al., 2026). Prior learning-based methods primarily modify the backbone, cache construction, or the interface that produces and exposes reusable representations.

3 Understanding Quality Loss and Recovery in PIC

We first examine how PIC affects task performance across different artifact configurations. Quality loss persists even when relevant artifacts are provided, motivating a closer look at how the model reads independently cached context.

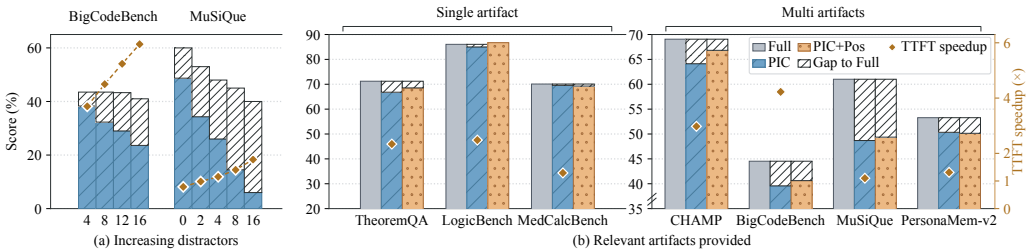


Figure 2: Task quality and TTFT speedup on Qwen3-4B. (a) Increasing distractors while retaining all relevant skills or supporting documents. (b) Supplying only task-annotated relevant artifacts, grouped into single- and multi-artifact settings.

3.1 Artifact Discovery and Evidence Utilization under PIC

We begin by examining how PIC impairs the model's use of artifacts when answering questions. Prior PIC systems primarily evaluate generation quality through downstream task scores on multi-chunk inputs (Yao et al., 2025; Hu et al., 2025; Agarwal et al., 2025), which combine different aspects of artifact use. Drawing on the distinction between retrieving relevant documents and reading them to answer questions in retrieval-augmented QA (Jiang et al., 2022), we distinguish *artifact discovery*, which identifies relevant artifacts among the candidates in the prompt, from *evidence utilization*, which reads and integrates information from those artifacts. We evaluate Qwen3-4B (Yang et al., 2025a) on SRA-Bench (Su et al., 2026), MuSiQue (Trivedi et al., 2022), and PersonaMem-v2 (Jiang et al., 2025), comparing Full and PIC on identical requests under matched generation settings. Experimental details are provided in Section 5.

Degradation extends beyond artifact discovery. Figure 2(a) increases the number of distracting artifacts while retaining all task-annotated relevant skills or supporting documents. PIC deteriorates more sharply than Full on both benchmarks, indicating greater sensitivity to irrelevant context. This pattern is consistent with impaired context-dependent reading when independent cache construction omits cross-artifact conditioning. To examine whether the degradation persists after removing irrelevant candidates, Figure 2(b) compares settings containing only annotated relevant artifacts. PIC incurs relatively small performance losses on the single-artifact tasks, whereas larger gaps remain on multi-artifacts tasks. These results suggest that PIC can retain much of an individual artifact's usefulness while impairing the use of information distributed across multiple artifacts. The degradation that remains *when relevant artifacts are provided* shows that artifact selection alone cannot explain PIC's performance loss. We therefore investigate how the model reads independently cached context to recover effective cache reading while preserving PIC's direct reuse of artifact KV.

Position alignment does not consistently close the gap. We next test whether positional mismatch accounts for this remaining degradation. **PIC-Pos** independently encodes each artifact at its target prompt positions while preserving independent cache construction. This control produces only marginal changes on multi-artifacts task. Position alignment restores positional coordinates but does not recover the conditioning on preceding artifacts that occurs during full prefill. The remaining loss therefore calls for an intervention beyond position correction.

3.2 Attention Routing Enables Recovery with Cached Values

Independent caching changes both cached values and attention routing, consistent with prior observations of attention misalignment under parallel encoding (Yang et al., 2025d). We therefore ask: *does the degradation stem from changes in the stored values, the way they are read, or both?*

To examine whether restoring routing is sufficient for recovery, we construct an *attention routing oracle* that retains PIC's cached artifact values. For each selected online token t, it replaces the

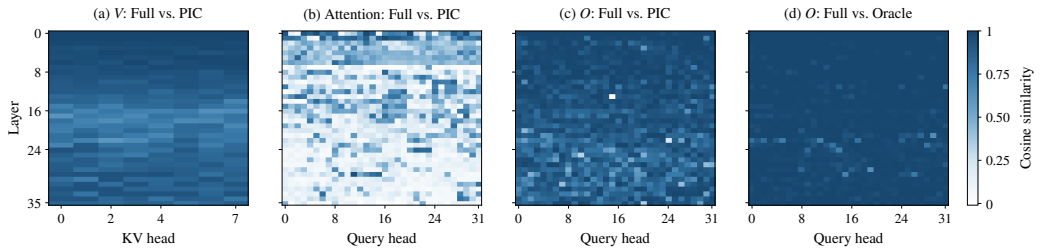


Figure 3: **Routing recovery with unchanged artifact KV.** Cosine similarities for one MuSiQue example: Full vs. PIC for (a) Full–PIC similarity of cached values across artifact tokens, (b) Full–PIC similarity of attention-weight vectors overartifact tokens, and (c–d) Similarity of per-head attention outputs to Full, under PIC and the routing oracle. Additional case studies appear in Appendix B.

attention weights with a Full reference and applies them to the intervention branch's own values:

$$\Pi^{\text{Full}}_{\ell,h,t} = \text{softmax}\left(\frac{q^{\text{Full}}_{\ell,h,t}(K^{\text{Full}}_{\ell,h,\le t})^\top}{\sqrt{d_k}} + M_t\right), \qquad o^{\text{oracle}}_{\ell,h,t} = \Pi^{\text{Full}}_{\ell,h,t} V^{\text{hybrid}}_{\ell,h,\le t}, \tag{3}$$

Here, M_t is the causal attention mask, and $o^{\text{oracle}}_{\ell,h,t}$ is the per-head attention output before the output projection. V^{hybrid} retains PIC's cached artifact values, while online values evolve under the intervention. We replace complete attention rows at every layer and query head for online prompt tokens following the first artifact and all subsequent decoding tokens. Figure 3 illustrates the effect of this intervention on a paired MuSiQue example.[1] Cached values remain relatively similar to Full in panel (a), whereas attention allocation over artifact tokens differs substantially in panel (b), particularly in later layers. Because attention outputs combine routing weights with values, routing differences need not translate proportionally into output differences. Replacing routing while retaining artifact KV brings the outputs substantially closer to Full (panels c–d): at the final prompt token, mean output cosine similarity increases from 0.898 to 0.984. Beyond this representation-level example, paired task evaluations show recovery to approximately Full performance. On MuSiQue, EM increases from 34.84 under PIC to **47.91** under the oracle, compared with **47.79** under Full. On BigCodeBench, pass@1 increases from 38.98 to **44.53**, compared with **43.63** under Full. These paired results establish an effective recovery path: *restoring attention routing can recover task quality while retaining independently cached values.*

4 ATTUNER: LEARNING TO READ A FROZEN CACHE

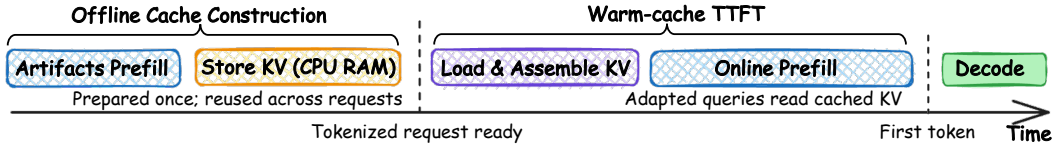


Figure 4: **Inference workflow of ATTUNER.** Artifact KV is prepared once and reused unchanged. Warm-cache TTFT includes KV loading and assembly, online prefill, and first-token selection, excluding offline cache construction.

Motivated by the cache-preserving recovery path in Section 3.2, We introduce ATTUNER, a simple yet efficient method to read independently cached artifacts. As illustrated in Figure 4, artifacts are prefilled in advance and reused across requests, while adapted queries operate during online prefill and decoding. We explain why query-side adaptation preserves cache reuse, then describe its training through Full-context supervision.

[1] Additional repair interventions and their effects on cached representations and attention routing are examined in Appendix D.1.

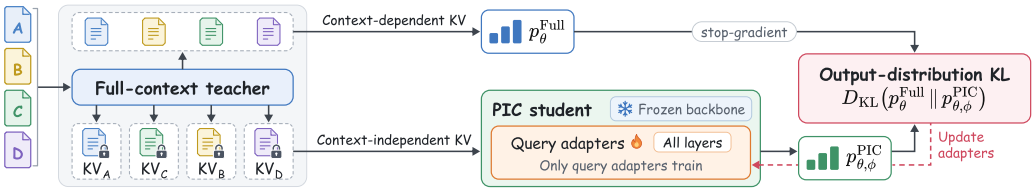


Figure 5: **Training ATTUNER.** A frozen Full-context teacher supervises the PIC student's output distributions under identical inputs and response prefixes. Only query adapters are updated; the backbone and cached artifact KV remain frozen.

Query Adaptation with Fixed Artifact KV Although both queries and keys determine attention weights, adapting queries preserves the existing artifact cache: *queries are computed for online tokens, while cached keys and values remain available for direct reuse*. We augment the frozen query projection in every Transformer layer with a low-rank update (Hu et al., 2022):

$$\widetilde{q}_{\ell,t} = W_\ell^Q h_{\ell,t} + \frac{\alpha}{r} B_\ell A_\ell h_{\ell,t}, \qquad \ell = 1, \ldots, L, \tag{4}$$

where $h_{\ell,t}$ is the projection input, $A_\ell \in \mathbb{R}^{r \times d_{\text{model}}}$ and $B_\ell \in \mathbb{R}^{d_q \times r}$ are trainable matrices, r is the adapter rank, and α controls the update scale. The adapted projection $\widetilde{q}_{\ell,t}$ then passes through the model's original query normalization and RoPE before entering attention. We initialize the update to zero and train only A_ℓ and B_ℓ, keeping all base-model parameters frozen. By enabling adapters only for uncached prompt and generated tokens, ATTUNER adapts cache reading while preserving artifact KV construction and reuse. *Stored artifact KV remains unchanged*; online states and newly produced KV *evolve under the adapted computation*.

Learning from Full-Context Behavior As shown in Figure 5, we distill Full-context answering behavior into the PIC student. For each request X, the frozen teacher generates a response Y, forming a fixed training set $\mathcal{D}_T$. Both branches receive the same input and condition on the same response prefix $Y_{<t}$, but the teacher uses full prefill while the student reuses independently cached artifact KV. We minimize

$$\mathcal{L}(\phi) = \mathbb{E}_{(X,Y)\sim\mathcal{D}_T}\left[\frac{1}{|Y|}\sum_{t=1}^{|Y|} D_{\text{KL}}\left(\text{sg}\left[p_\theta^{\text{Full}}(\cdot \mid X, Y_{<t})\right] \,\|\, p_{\theta,\phi}^{\text{PIC}}(\cdot \mid X, Y_{<t})\right)\right], \tag{5}$$

where $\phi = \{A_\ell, B_\ell\}_{\ell=1}^L$ denotes the query-adapter parameters and sg stops gradients through the teacher. Supervising output distributions rather than attention weights allows ATTUNER to learn a reading strategy suited to the fixed cache. At deployment, the trained updates are merged into the query projections, requiring no Full-context teacher.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Implementation details. We evaluate ATTUNER on Qwen3-4B and Qwen3-8B (Yang et al., 2025a) using skills from SRA-Bench (Su et al., 2026), documents from MuSiQue (Trivedi et al., 2022), and memory messages from PersonaMem-v2 (Jiang et al., 2025). SRA-Bench includes TheoremQA (Chen et al., 2023), LogicBench (Parmar et al., 2024), MedCalc-Bench (Khandekar et al., 2024), CHAMP (Mao et al., 2024), and BigCodeBench (Zhuo et al., 2025) (excluding ToolQA (Zhuang et al., 2023), which reuses a single skill across multi-step ReAct interactions). For each base model, we train one shared adapter on a balanced mixture of 12,000 examples: 4,000 each from the training splits of MuSiQue and PersonaMem-v2, and 4,000 synthetic skill-use examples constructed from BigCodeBench training-side skills. These skills are disjoint from those used for evaluation. Full-context responses from the corresponding base model provide supervision, and the same adapter is evaluated across all tasks.

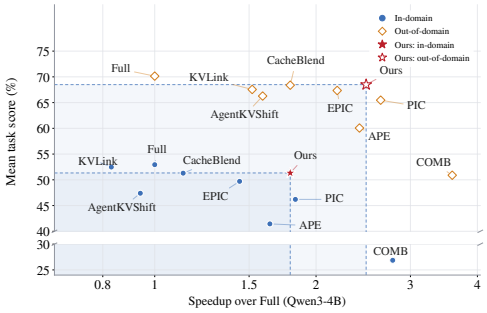


Figure 6: **Quality–efficiency trade-off.** Quality versus speedup. Shaded regions indicate lower quality and speed than ATTUNER.

Table 1: **Single-artifact quality and efficiency.** TQA, LB, and MCB denote TheoremQA, LogicBench, and MedCalc-Bench, respectively.

	Dataset	Score			Efficiency	
		Full	PIC	PIC-Pos	Full	PIC / PIC-Pos
Qwen3-4B	TQA	71.22	66.80 ↓4.42	68.54 ↓2.68	89.08	38.14 (2.34×)
	LB	86.05	85.00 ↓1.05	86.71 ↑0.66	94.43	38.08 (2.48×)
	MCB	70.09	69.55 ↓0.54	69.27 ↓0.82	124.61	96.50 (1.29×)
Qwen3-8B	TQA	63.05	55.15 ↓7.90	52.34 ↓10.71	145.85	38.89 (3.75×)
	LB	87.24	85.79 ↓1.45	87.63 ↑0.39	156.91	40.15 (3.91×)
	MCB	72.91	72.27 ↓0.64	72.27 ↓0.64	207.36	140.12 (1.48×)

Baselines. We compare ATTUNER with Full prefill and unmodified PIC. Our baselines include training-free methods APE (Yang et al., 2025d), EPIC (Hu et al., 2025), CacheBlend (Yao et al., 2025), and AgentKVShift (Pandey et al., 2026), as well as training-based methods KVLink (Yang et al., 2025c) and COMB (Zhao et al., 2026). Method-specific cache construction, training, and implementation details are provided in Appendix C.

Evaluation metrics. We report pass@1 for BigCodeBench, F1 for MuSiQue, and accuracy for the remaining tasks. For efficiency, we measure median warm-cache time to first token (TTFT), with reusable KV already resident in CPU memory. Timing includes cache transfer to the GPU and assembly, method-specific repair, uncached-token prefill, and first-token selection; it excludes model loading, tokenization, and offline cache construction. Speedup is Full's median TTFT divided by that of the compared method under matched examples, hardware, and inference settings.

5.2 MAIN RESULTS

Single-artifact caching offers substantial speedups, but uneven quality. Table 1 shows that direct cache reuse achieves 1.29–3.91× speedup over Full. On LogicBench and MedCalc-Bench, PIC remains within 1.45 accuracy points of Full across both model sizes, providing an attractive quality–latency trade-off. TheoremQA exposes a limitation: accuracy drops by 4.42 points on Qwen3-4B and 7.90 points on Qwen3-8B, and *position alignment does not consistently recover the loss*. We therefore also include TheoremQA as an out-of-distribution task to evaluate whether ATTUNER's learned cache-reading behavior transfers beyond the training benchmarks.

ATTUNER recovers quality within and beyond the training tasks. Table 2 shows that ATTUNER improves over PIC on all five tasks for both model sizes. For readability, Appendix E presents enlarged tables of the in-domain and out-of-domain results separately. Across the three in-domain benchmarks, average scores increase by 5.13 and 7.77 points on Qwen3-4B and Qwen3-8B, reducing the gaps to Full to 1.61 and 0.51 points, respectively. **A single adapter improves the use of skills, documents, and memories while preserving their original cached KV.** Among the repair methods[2], ATTUNER achieves the second-highest in-domain average on both models, behind KVLink; on Qwen3-8B, it obtains the highest scores on BigCodeBench and MuSiQue. The benefits also extend to TheoremQA and CHAMP, *neither of which is included in adapter training*: their average scores improve over PIC by 3.02 and 3.51 points. These results demonstrate transfer beyond the training tasks, although recovery remains incomplete on some benchmarks.

ATTUNER recovers quality at near-PIC latency. With query updates merged into the projection weights, ATTUNER's geometric-mean in-domain TTFT is only 1.8% and 0.5% higher than PIC on Qwen3-4B and Qwen3-8B, preserving 1.79× and 2.45× speedups over Full. Unlike recomputation-based repair, ATTUNER retains the original artifact KV and requires *neither online token selection nor artifact-token recomputation*. Figure 6 illustrates the resulting trade-off: ATTUNER attains **higher average quality and lower TTFT than all evaluated training-free repair baselines**.

[2]Baseline hyperparameters and implementation details are provided in Appendix C.

Table 2: **Quality and efficiency on training-domain and held-out tasks.** BCB, MSQ, PMem, TQA, and CHP denote BigCodeBench, MuSiQue, PersonaMem, TheoremQA, and CHAMP, respectively. Avg is the arithmetic mean of scores within each domain. Geo. denotes the geometric mean of TTFT within each domain. Full and PIC are the references and excluded from ranking. The best and second-best results among the remaining methods are **bold** and underlined.

	Method	In-domain									Out-of-domain						
		BCB		MSQ		PMem		Quality		Efficiency	TQA		CHP		Quality		Efficiency
		Score ↑	TTFT ↓	Score ↑	TTFT ↓	Score ↑	TTFT ↓	Avg ↑	Δ	Geo. ↓	Speedup ↑	Score ↑	TTFT ↓	Score ↑	TTFT ↓	Avg ↑	Δ	Geo. ↓	Speedup ↑
Qwen3-4B	Full	44.53	420.77	61.01	39.06	53.28	62.38	52.94	–	100.83	1.00×	71.22	89.08	69.06	139.01	70.14	–	111.28	1.00×
	PIC	39.58	99.60	48.69	35.51	50.34	47.56	46.20	↓6.74	55.20	1.83×	66.80	38.14	64.13	46.70	65.47	↓4.68	42.20	2.64×
	Training-free																		
	APE	39.58	114.59	41.02	39.17	43.72	51.92	41.44	↓11.50	61.54	1.64×	59.57	42.88	60.54	49.83	60.06	↓10.09	46.22	2.41×
	EPIC	44.08	100.57	54.21	50.76	50.80	67.29	49.70	↓3.24	70.03	1.44×	67.87	46.73	66.82	55.23	67.35	↓2.80	50.80	2.19×
	CacheBlend	**45.13**	194.81	54.82	52.43	**53.92**	69.73	51.29	↓1.65	89.30	1.13×	68.14	50.95	**68.61**	75.92	68.38	↓1.77	62.19	1.79×
	AgentKVShift	42.13	196.54	52.62	68.78	47.42	90.25	47.39	↓5.55	106.85	0.94×	67.07	61.59	65.47	79.94	66.27	↓3.87	70.17	1.59×
	Training-based																		
	KVLink	44.98	165.04	**59.93**	98.70	52.54	109.94	**52.48**	↓0.46	121.44	0.83×	66.93	68.96	68.16	78.02	67.55	↓2.60	73.35	1.52×
	COMB	35.68	**38.29**	12.70	**28.68**	32.22	**43.49**	26.87	↓26.07	**36.28**	**2.78**×	47.52	**30.78**	54.26	**31.18**	50.89	↓19.25	**30.98**	**3.59**×
	ATTUNER	44.08	101.27	58.06	36.71	51.86	47.75	51.33	↓1.61	56.20	1.79×	**68.81**	39.80	68.16	50.62	**68.49**	↓1.66	44.89	2.48×
Qwen3-8B	Full	50.22	660.41	63.28	62.89	43.90	99.14	52.47	–	160.28	1.00×	63.05	145.85	62.78	227.68	62.92	–	182.23	1.00×
	PIC	47.08	113.82	50.00	35.33	35.50	68.60	44.19	↓8.27	65.10	2.46×	55.15	38.89	63.23	59.17	59.19	↓3.73	47.97	3.80×
	Training-free																		
	APE	46.63	123.16	44.94	39.60	44.86	73.10	45.48	↓6.99	70.91	2.26×	52.34	45.10	51.12	60.80	51.73	↓11.19	52.36	3.48×
	EPIC	48.58	110.50	56.71	50.79	42.40	88.41	49.23	↓3.24	79.17	2.02×	58.23	58.89	61.88	67.53	60.06	↓2.86	63.06	2.89×
	CacheBlend	48.43	249.41	58.12	54.92	42.78	96.36	49.78	↓2.69	109.69	1.46×	57.56	66.42	42.60	101.32	50.08	↓12.84	82.03	2.22×
	AgentKVShift	47.53	246.76	55.58	68.88	40.92	106.07	48.01	↓4.46	121.71	1.32×	**61.58**	80.25	64.13	105.39	**62.86**	↓0.06	91.96	1.98×
	Training-based																		
	KVLink	50.22	182.91	61.20	102.70	**46.56**	143.02	**52.66**	↑0.19	139.02	1.15×	58.50	70.52	**66.37**	92.46	62.44	↓0.48	80.75	2.26×
	COMB	42.13	**48.17**	14.90	**31.02**	31.62	**67.51**	29.55	↓22.92	**46.55**	**3.44**×	42.84	42.24	51.57	**44.33**	47.21	↓15.71	**43.27**	**4.21**×
	ATTUNER	**50.52**	113.19	**61.22**	36.33	44.14	68.10	51.96	↓0.51	65.42	2.45×	60.37	**39.80**	65.02	59.98	62.70	↓0.22	48.86	3.73×

KVLink achieves higher average quality, but requires more than twice ATTUNER's geometric-mean TTFT on both models. COMB is faster, but incurs substantially larger quality losses under our evaluation setting. Thus, these results show that adapting cache reading can recover quality while retaining the latency advantages of direct cache reuse.

5.3 ANALYSIS

More training data progressively closes the quality gap. Figure 7 shows that **limited training data already enables substantial recovery**. On MuSiQue, 1,024 examples raise F1 from 48.70 to 56.40; 8,192 examples bring it to 61.15, matching the Full baseline of 61.01. BigCodeBench similarly improves from 39.43 to 44.08 pass@1, approaching Full's 44.53. These results show that substantial recovery is possible with limited training data, with further gains as training coverage expands.

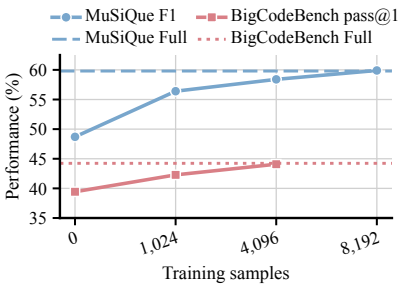


Figure 7: Effect of training data size on Qwen3-4B for 1 epoch. Zero examples denotes PIC without adaptation;

Query adaptation improves attention-output alignment. We compare per-head attention outputs O before the output projection, using identical prompts and teacher-forced Full-response prefixes. Figure 8 shows improved alignment with Full across many intermediate layers, at both online prompt and answer positions. During the answer phase, mean cosine similarity increases from 0.903 to 0.927 on MuSiQue and from 0.932 to 0.949 on BigCodeBench. The improvement is not uniform: alignment deteriorates at the final layer. Importantly, ATTUNER is trained on output distributions, without directly supervising attention weights or O. The resulting alignment provides *representation-level evidence that behavioral distillation improves how the fixed cache is used*, without requiring exact reproduction of Full's layer-wise computation.

Adapting a subset of layers can recover quality. The uneven alignment changes across layers raise a related question: does effective cache reading require adapting every layer? We examine this through two equally sized adaptation windows, L16–33 and L18–35, each covering half of Qwen3-4B's layers. Both use 1,024 MuSiQue training examples and a shared teacher under matched training settings. On 128 test samples, they achieve 65.14 and 63.34 F1, respectively, compared

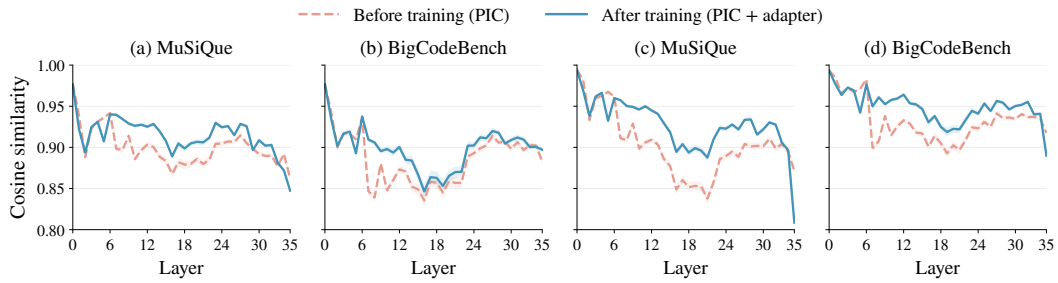


Figure 8: **Attention-output alignment before and after training.** Cosine similarity of O to Full on Qwen3-4B. The first two panels show online prompt positions; the last two show teacher-forced answer positions. Each pair presents MuSiQue followed by BigCodeBench, with 128 examples.

with 49.00 for PIC (Table 3). Although L16–33 scores higher, the difference is inconclusive under paired bootstrap resampling. Together with the preceding alignment analysis, these results show that recovery need not reproduce Full's computation at every layer: substantial gains remain possible when trainable updates are confined to a subset of layers.

The gains are larger under PIC than under full prefill. We apply the same trained Qwen3-4B adapter to Full and PIC to assess whether its benefits reflect a general improvement in task performance. As shown in Table 3, adaptation improves PIC on all four tasks, whereas its effect under Full is mixed: MuSiQue and PersonaMem-v2 improve, but BigCodeBench and CHAMP decline. Across the three in-domain tasks, the average gain is 5.13 points under PIC, compared with only 0.17 points under Full. Moreover, **the adaptation gain is larger under PIC on every task**. This contrast supports adaptation to independently cached context as the primary source of improvement, rather than a uniform benefit from additional training.

Table 3: **Adapter controls on Qwen3-4B.** (a) Layer-range comparison on 128 MuSiQue test examples after training on 1,024 examples (b) The same L0–35 adapter under Full and PIC; The two panels use different training and evaluation sets.

(a) Layer-range comparison on MuSiQue

Method	Layers	EM	F1	Δ F1
Full	–	46.09	60.06	–
PIC	–	34.38	49.00	–
PIC + ATTUNER	16–33	50.78	65.14	+16.13
PIC + ATTUNER	18–35	50.00	63.34	+14.34

(b) Full versus PIC adaptation

	In-domain			Held-out
Method	BCB	MSQ	PMem	CHP
Full	44.53	61.01	53.28	69.06
Full + ATTUNER	41.98	63.70	53.64	64.57
Δ	−2.55	+2.69	+0.36	−4.49
PIC	39.58	48.69	50.34	64.13
PIC + ATTUNER	44.08	58.06	51.86	68.16
Δ	+4.50	+9.37	+1.52	+4.03

6 CONCLUSION

We show that PIC's quality loss extends beyond artifact discovery to multi-artifact evidence utilization, and that positional alignment alone does not consistently recover performance. Restoring full-prefill attention routing recovers task quality with cached artifact KV unchanged, establishing a recovery path without cache reconstruction. Motivated by this finding, ATTUNER learns to read a frozen artifact cache through low-rank query updates trained by distilling full-prefill output distributions. It trains fewer than 0.05% of model parameters and requires neither artifact-cache recomputation nor a teacher at inference. Across skill, document, memory, and code tasks, a single adapter per model improves average in-domain scores over PIC by 5.13 and 7.77 points on Qwen3-4B and Qwen3-8B, respectively, and transfers to held-out tasks. These gains come at near-PIC warm-cache TTFT, retaining in-domain geometric-mean speedups of 1.79× and 2.45× over full prefill. These results highlight query-side adaptation as an effective path toward high-quality, efficient cache reuse: learning to read the cache rather than rewriting it. Limitations and future directions are discussed in Appendix D.2.

AI USE STATEMENT

Generative AI tools were used to polish the manuscript, assist in finding relevant research, and support code development and figure preparation. The authors take responsibility for the final text, citation accuracy, implementation correctness, and scientific claims, including all AI-assisted content.

ETHICS STATEMENT

This work studies efficient inference using existing models and benchmarks. The proposed method does not address biases, privacy risks, or harmful outputs inherited from the underlying models and data. Applications involving reusable context should ensure appropriate access controls and protection of cached information.

REPRODUCIBILITY STATEMENT

Section 4 describes the adaptation mechanism and training objective. Section 5.1 specifies the benchmarks, training data mixture, evaluation metrics, and TTFT measurement protocol. Data preparation, training settings, hardware, and evaluation protocols are provided in Appendix A. Baseline implementations and configurations are detailed in Appendix C.

REFERENCES

Shubham Agarwal, Sai Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit Gupta, Rounak Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Saini. Cache-craft: Managing chunk-caches for efficient retrieval-augmented generation. *Proc. ACM Manag. Data*, 3(3), June 2025. doi: 10.1145/3725273. URL https://doi.org/10.1145/3725273.

Zhuohang Bian, Feiyang Wu, Chengrui Zhang, Hangcheng Dong, Yun Liang, and Youwei Zhuo. Tokendance: Scaling multi-agent llm serving via collective kv cache sharing, 2026. URL https://arxiv.org/abs/2604.03143.

Chuangtao Chen, Grace Li Zhang, Xunzhao Yin, Cheng Zhuo, Bing Li, and Ulf Schlichtmann. Kv packet: Recomputation-free context-independent kv caching for llms, 2026. URL https://arxiv.org/abs/2604.13226.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. TheoremQA: A theorem-driven question answering dataset. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7889–7901, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.489. URL https://aclanthology.org/2023.emnlp-main.489/.

Chuheng Du, Junyi Chen, Hanlin Tang, Kan Liu, Tao Lan, Lin Qu, Chaoyue Niu, Shengzhong Liu, Guihai Chen, and Fan Wu. C2kv: Compressed and composable kv cache reuse for efficient llm inference. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '26, pp. 991–1002, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722592. doi: 10.1145/3770855.3817715. URL https://doi.org/10.1145/3770855.3817715.

Yichu Fang, Sitong Wei, Haozhe Hu, and Xiaoyu Shen. Recache: Efficient kv cache reuse and compression for tool-augmented llm agents, 2026. URL https://arxiv.org/abs/2608.19662.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/a66caa1703fe34705a4368c3014c1966-Abstract-Conference.html.

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. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, tiancheng hu, zhang qin, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent caching for serving large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=qjd3ZUiHRT.

Bowen Jiang, Yuan Yuan, Maohao Shen, Zhuoqun Hao, Zhangchen Xu, Zichen Chen, Ziyi Liu, Anvesh Rao Vijjini, Jiashu He, Hanchao Yu, Radha Poovendran, Gregory Wornell, Lyle Ungar, Dan Roth, Sihao Chen, and Camillo Jose Taylor. Personamem-v2: Towards personalized intelligence via learning implicit user personas and agentic memory, 2025. URL https://arxiv.org/abs/2512.06688.

Zhengbao Jiang, Luyu Gao, Zhiruo Wang, Jun Araki, Haibo Ding, Jamie Callan, and Graham Neubig. Retrieval as attention: End-to-end learning of retrieval and reading within a single transformer. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 2336–2349, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.149. URL https://aclanthology.org/2022.emnlp-main.149/.

Nikhil Khandekar, Qiao Jin, Guangzhi Xiong, Soren Dunn, Serina S Applebaum, Zain Anwar, Maame Sarfo-Gyamfi, Conrad W Safranek, Abid Anwar, Andrew Jiaxing Zhang, Aidan Gilson, Maxwell B Singer, Amisha D Dave, R. Andrew Taylor, Aidong Zhang, Qingyu Chen, and Zhiyong Lu. Medcalc-bench: Evaluating large language models for medical calculations. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=VXohja0vrQ.

Bojie Li. Models take notes at prefill: Kv cache can be editable and composable, 2026. URL https://arxiv.org/abs/2606.17107.

Haoyang LI, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole HU, 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. ISSN 2835-8856. URL https://openreview.net/forum?id=z3JZzu9EA3.

Yang Liu, Yunfei Gu, Liqiang Zhang, Chentao Wu, Guangtao Xue, Jie Li, Minyi Guo, Junhao Hu, and Jie Meng. CacheSlide: Unlocking cross Position-Aware KV cache reuse for accelerating LLM serving. In *24th USENIX Conference on File and Storage Technologies (FAST 26)*, pp. 83–99, Santa Clara, CA, February 2026a. USENIX Association. ISBN 978-1-939133-53-3. URL https://www.usenix.org/conference/fast26/presentation/liu-yang.

Yang Liu, Zhaokai Luo, Huayi Jin, Zhiyong Wang, Ruozhou He, Boyu Wang, Guanjie Chen, Yifei Liu, Tao Xie, and Junhao Hu. Redknot: Efficient long-context llm serving with head-aware kv reuse and segpagedattention, 2026b. URL https://arxiv.org/abs/2606.06256.

Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. TurboRAG: Accelerating retrieval-augmented generation with precomputed KV caches for chunked text. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6588–6601, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.334. URL https://aclanthology.org/2025.emnlp-main.334/.

Jinghao Luo, Yuchen Tian, Chuxue Cao, Ziyang Luo, Hongzhan Lin, Kaixin Li, Chuyi Kong, Ruichao Yang, and Jing Ma. From storage to experience: A survey on the evolution of llm agent memory mechanisms, 2026. URL https://arxiv.org/abs/2605.06716.

Dongyang Ma, Yan Wang, and Tian Lan. Block-attention for efficient prefilling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=7zNYY1E2fq.

Yujun Mao, Yoon Kim, and Yilun Zhou. CHAMP: A competition-level dataset for fine-grained analyses of LLMs' mathematical reasoning capabilities. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13256–13274, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.785. URL https://aclanthology.org/2024.findings-acl.785/.

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. A survey of context engineering for large language models, 2025. URL https://arxiv.org/abs/2507.13334.

Nathan Ordonez and Thomas Parnell. Minipic: Flexible position-independent caching in ¡100loc, 2026. URL https://arxiv.org/abs/2606.13126.

Nilesh Prasad Pandey, Jason Kong, Lanxiang Hu, Quanling Zhao, Yujie Zhao, Onat Gungor, Hao Zhang, and Tajana Rosing. Agentkvshift: Efficient kv cache reuse for agentic memory systems, 2026. URL https://arxiv.org/abs/2607.21604.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. LogicBench: Towards systematic evaluation of logical reasoning ability of large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13679–13707, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.739. URL https://aclanthology.org/2024.acl-long.739/.

Shengling Qin, Hao Yu, Chenxin Wu, Zheng Li, Yizhong Cao, Zhengyang Zhuge, Yuxin Zhou, Wentao Yao, Yi Zhang, Zhengheng Wang, Shuai Bai, Jianwei Zhang, and Junyang Lin. Vlcache: Computing 2URL https://arxiv.org/abs/2512.12977.

Weihang Su, Jianming Long, Qingyao Ai, Qiaozhi He, Yichen Tang, Changyue Wang, Yiteng Tu, Yingbo Wang, and Yiqun Liu. Skill retrieval augmentation for agentic ai, 2026. URL https://arxiv.org/abs/2604.24594.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/.

Qian Wang, Zahra Yousefijamarani, Morgan Lindsay Heisler, Rongzhi Gu, Bai Xiaolong, Shan Yizhou, Wei Zhang, Wang Lan, Ying Xiong, Yong Zhang, and Zhenan Fan. Mepic: Memory efficient position independent caching for llm serving, 2025. URL https://arxiv.org/abs/2512.16822.

Shihao Wang, Jiahao Chen, Yanqi Pan, Hao Huang, Yichen Hao, Xiangyu Zou, Wen Xia, Chongyang Qiu, Wentao Zhang, and PengFei Wang. ProphetKV: User-query-driven selective recomputation for efficient KV cache reuse in retrieval-augmented generation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=SwGwmKIgUb.

Hui Xie, Peng Xiao, Yutong Deng, Shuoran Dou, Jian Yang, and Jinyang Guo. Sempic: Learning semantic position-independent kv caches, 2026. URL https://arxiv.org/abs/2607.28069.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin

Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Bin Yang, Qiuyu Leng, Jun Zeng, and Zhenhua Wu. Cacheclip: Accelerating rag with effective kv cache reuse, 2026. URL https://arxiv.org/abs/2510.10129.

Huan Yang, Renji Zhang, Mingzhe Huang, Weijun Wang, Yin Tang, Yuanchun Li, Yunxin Liu, and Deyu Zhang. Kvshare: An llm service system with efficient and effective multi-tenant kv cache reuse, 2025b. URL https://arxiv.org/abs/2503.16525.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. KVLink: Accelerating large language models via efficient KV cache reuse. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025c. URL https://openreview.net/forum?id=oDcAGSXZZP.

Xinyu Yang, Tianqi Chen, and Beidi Chen. APE: Faster and longer context-augmented generation via adaptive parallel encoding. In *The Thirteenth International Conference on Learning Representations*, 2025d. URL https://openreview.net/forum?id=yUC8pU508S.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 94–109, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696098. URL https://doi.org/10.1145/3689031.3696098.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. KVCOMM: Online cross-context KV-cache communication for efficient LLM-based multi-agent systems. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=yGOytgjurF.

Shiju Zhao, Junhao Hu, Rongxiao Huang, Jiaqi Zheng, and Guihai Chen. Mpic: Position-independent multimodal context caching system for efficient mllm serving, 2025. URL https://arxiv.org/abs/2502.01960.

Shiju Zhao, Junhao Hu, Jiaqi Zheng, and Guihai Chen. You need an encoder for native position-independent caching, 2026. URL https://arxiv.org/abs/2602.01519.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: efficient execution of structured language model programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Yingli Zhou, Wang Shu, Yaodong Su, Wenchuan Du, Yixiang Fang, and Xuemin Lin. A comprehensive survey on agent skills: Taxonomy, techniques, and applications, 2026. URL https://arxiv.org/abs/2605.07358.

Yuechi Zhou, Yi Su, Jianxin Zhang, Juntao Li, Qingrong Xia, Zhefeng Wang, Xinyu Duan, and Baoxing Huai. a^3: Attention-aware accurate kv cache fusion for fast large language model serving, 2025. URL https://arxiv.org/abs/2511.17560.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. Toolqa: A dataset for llm question answering with external tools, 2023. URL https://arxiv.org/abs/2306.13304.

Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=YrycTjllL0.

APPENDIX

A EXPERIMENTAL DETAILS

This appendix describes the data, prompt construction, and training and generation settings used in the main experiments. The diagnostic studies and data-scale ablations use their own explicitly specified cohorts; their reference scores should not be interchanged with main-table results.

A.1 DATASETS AND SPLITS

We evaluate Qwen3-4B and Qwen3-8B (Yang et al., 2025a) on seven benchmarks covering reusable skills, documents, and memory messages. Table 4 summarizes the prepared evaluation sets. In the relevant-only setting, all annotated relevant artifacts are supplied directly, without a retrieval stage or additional distractors. The evaluated requests and artifact order are matched across methods. Appendix F illustrates the cached artifacts, online questions, and evaluation targets with one prepared example each from BigCodeBench, MuSiQue, and PersonaMem-v2.

Table 4: **Main evaluation sets.** Artifact counts are per request. Training counts refer to the shared 12K mixture; BigCodeBench training examples are synthetic. MuSiQue uses its official development set for held-out evaluation.

Dataset	Artifact unit	Train	Eval.	Artifacts	Metric
TheoremQA	Skill	0	747	1	Accuracy
LogicBench	Skill	0	760	1	Accuracy
MedCalc-Bench	Skill	0	1,100	1	Accuracy
CHAMP	Skill	0	223	1–5	Accuracy
BigCodeBench	Skill	4,000	667	2–5	pass@1
MuSiQue	GT document	4,000	2,417	2–4	F1 / EM
PersonaMem-v2	GT message	4,000	5,000	2–8	Accuracy

Skills. We use the task-annotated skills and prompt builders from SRA-Bench (Su et al., 2026) for TheoremQA (Chen et al., 2023), LogicBench (Parmar et al., 2024), MedCalc-Bench (Khandekar et al., 2024), CHAMP (Mao et al., 2024), and BigCodeBench (Zhuo et al., 2025). Each annotated skill is treated as a separate artifact. TheoremQA, LogicBench, and MedCalc-Bench contain one skill per request. CHAMP contains a mixture: 114 requests have one skill, and 109 have multiple skills. It should therefore not be interpreted as an exclusively multi-artifact benchmark. ToolQA is excluded because its single skill is reused across multi-step ReAct interactions, which differs from the request-level setting studied here.

For BigCodeBench, we use a fixed skill-disjoint partition of the 1,140 SRA-Bench instances. The manifest divides 139 skill IDs into 70 training-side and 69 evaluation-side skills. An instance is retained only if all its annotated skills belong to the same side. This yields 198 real training-side instances and 667 evaluation instances; 275 instances spanning both partitions are excluded. The main adapter mixture uses synthetic tasks built from the training-side skills, rather than the 198 real instances. The latter support separate validation and ablation experiments.

Documents. For MuSiQue (Trivedi et al., 2022), we retain only paragraphs marked as supporting the answer, preserving their original relative order and document indices. Each supporting document is one artifact. The 2,417 evaluation requests contain 1,252 two-document, 760 three-document, and 405 four-document contexts. Questions, reference answers, and answer aliases are retained from the official data. This GT-document protocol isolates the use of supplied evidence; it is distinct from evaluation with the full candidate-document pool.

Memory messages. For PersonaMem-v2 (Jiang et al., 2025), we use the annotated `related_conversation_snippet` rather than the full conversation history. Each annotated user or assistant message is a separate artifact, with its role, content, and annotation order preserved.

The request-specific question and multiple-choice instruction remain online. We retain the saved per-request option ordering and answer mapping for all methods. The evaluation uses all 5,000 benchmark requests; unannotated conversation history and persona background are not added.

A.2 Evaluation Metrics and Efficiency Protocol

Task quality. We use the SR-Agents benchmark scoring pipeline for the five skill tasks. BigCodeBench pass@1 is the fraction of requests whose single generated solution passes the task's unit tests, executed in local subprocesses. For MuSiQue, the final answer is extracted from the last `ANSWER:` line, or the last nonempty line when that marker is absent, and scored against the reference answer and aliases using the benchmark's answer metrics. F1 is the main-table metric; EM is also reported for diagnostic experiments. PersonaMem-v2 uses the official multiple-choice answer extraction and correctness functions with the saved option mapping. A response that reaches its generation limit is scored as generated rather than discarded.

Timing boundary. We measure warm-cache time to first token (TTFT) with reusable KV resident in CPU RAM. Model loading, tokenization, artifact-boundary identification, and cache construction occur before the timed region. Timing begins before cache lookup and includes transfer to the GPU, KV assembly, method-specific online repair, and uncached-token prefill. It ends after greedy selection of the first output token. GPU operations are synchronized at both boundaries. Subsequent generation, text decoding, and scoring are excluded (Table 5). Thus, this metric measures request processing from a prepared tokenized input, rather than end-to-end service latency.

Table 5: **Warm-cache TTFT boundary.** Reusable KV is already resident in CPU memory before timing begins.

Operation	Timed?
Model loading, tokenization, artifact-boundary identification	No
Offline artifact KV construction	No
CPU RAM cache lookup and GPU transfer	Yes
KV assembly and online computation, including repair	Yes
Greedy first-token selection	Yes
Subsequent generation, text decoding, and scoring	No

Measurement and aggregation. For the main-table efficiency results, we evaluate the full evaluation set of each dataset on NVIDIA H20 GPUs with batch size one and no concurrent workload on the same GPU. The dedicated efficiency runs use batch size one and generate exactly one token per request. Each worker performs five untimed warm-up requests for each dataset–method pair, followed by one timed measurement per evaluation request. Quality generation is evaluated separately; concurrent quality-run timings are not used as formal TTFT results. For dataset d and method m, we report

$$T_{m,d} = \mathrm{median}_{i \in \mathcal{E}_d} t_{m,d,i}, \qquad S_{m,d} = \frac{T_{\mathrm{Full},d}}{T_{m,d}}, \tag{6}$$

where $\mathcal{E}_d$ is the matched evaluation cohort and $t_{m,d,i}$ is the measured TTFT for request i. When requests are distributed across workers, we merge request-level timings before computing the median; we do not average worker medians. Cross-dataset efficiency summaries use the geometric mean over the same datasets for every method: $G_m = (\prod_{d \in \mathcal{D}} T_{m,d})^{1/|\mathcal{D}|}$, with aggregate speedup G_{Full}/G_m.

Cache and execution settings. A token-identical prefix shared across the dataset may also be cached before timing. For ATTUNER, query updates are merged into projection weights for online inference, while artifact KV is built with the unadapted model. Full and cached methods must use matched prompts, evaluation cohorts, hardware, and timing boundaries for a valid speedup comparison.

A.3 Prompt Construction and Artifact Boundaries

We render requests with the corresponding Qwen3 chat template and `enable_thinking=False`. Artifact boundaries are located within the rendered token sequence. Full and PIC receive identical prompt tokens within each comparison; they differ in how

the artifact states are computed. Text outside the artifact spans, including instructions, separators, and questions, is processed online.

Table 6: **Prompt layouts across artifact families.** Bracketed artifact spans are cached; other text is computed online. These layouts omit full task text and chat-template special tokens.

Family	Model-visible layout
Skills	Optional system instruction; `Relevant Skill:`; [skill A]; separator; [skill B]; task and answer instruction.
Documents	System instruction; `Documents:`; [document A: index, title, text]; [document B: index, title, text]; question and answer instruction.
Memories	[relevant message A]; [relevant message B]; user query; multiple-choice instruction with fixed options.

Skill and document spans. For SRA-Bench, the skill contents are prepended to the task using the benchmark prompt builder. The standard layout separates consecutive skills with `\n---\n`; these separators are outside the cached skill spans. MuSiQue artifacts contain the document identifier, title, and text. The surrounding task instructions and question are not part of the cached documents. Any separately reported separator-free efficiency sweep uses an explicitly different prompt protocol and should not be combined with quality scores from the standard layout.

Message spans. PersonaMem-v2 artifacts cover complete historical messages, including their chat-template boundaries. This preserves user/assistant roles when independently encoded blocks are assembled. Only annotated messages are cached; the new question and option instruction are not.

Cache construction and reuse. Each artifact is independently encoded with query adapters disabled. At request time, its cached KV is assembled in the original prompt order during evaluation. Adapters are enabled for uncached prompt tokens and generated tokens, while stored artifact KV remains fixed. Subsequent online states and their newly produced KV evolve under the adapted computation.

A.4 Training Data Construction

Balanced mixture. We form one fixed mixture of 12,000 requests: 4,000 MuSiQue questions, 4,000 synthetic BigCodeBench-style tasks, and 4,000 PersonaMem-v2 questions. The two model sizes use the same selected request IDs but their own Full-context teacher responses. MuSiQue requests are selected from an 8,192-example pool drawn from the official training split. PersonaMem-v2 training personas exclude all evaluation personas; validation personas are additionally removed before sampling the training mixture. The selection uses seed 20260918. TheoremQA, LogicBench, MedCalc-Bench, and CHAMP contribute no training examples to this mixture.

We reserve 128 MuSiQue training-split requests, 31 real training-side BigCodeBench requests, and 128 PersonaMem-v2 requests for validation. The selected training and validation request IDs are disjoint. For PersonaMem-v2, training, validation, and evaluation are also disjoint at the persona level. These main-training validation sets should not be conflated with separately constructed diagnostic cohorts.

Synthetic skill-use tasks. We construct the BigCodeBench-style source using only training-side skills. The synthesis procedure samples two to five skill documents and asks a generator to produce a Python function task requiring their joint use. The task specifies its behavior, arguments, return type, and edge cases, and includes a starter code block defining `task_func`. The parser requires a valid starter, retains imports and the function signature, and replaces the function body with `pass` so that a completed solution is not embedded in the input. The resulting tasks are reformatted with the same skill prompt builder used for evaluation. All selected skill IDs belong to the training partition; evaluation-side skill documents are excluded. The 4,000 synthetic requests are shared across model

sizes. They provide distillation inputs rather than independently verified programming solutions: supervision comes from each model's Full teacher.

The recovered generation pipeline uses Qwen3-4B with task-generation temperature 0.8, top-$p = 0.95$, and a 512-token task budget. The number of skills is sampled from $\{2, 3, 4, 5\}$ with probabilities $\{0.30, 0.35, 0.25, 0.10\}$. Tasks with an invalid starter or prompts exceeding 36,864 tokens are rejected. The source pipeline also requires a nonempty Full response containing a code block, using a 1,024-token greedy response budget; it does not validate synthetic solutions with unit tests. The completed source combines 3,027 initially accepted tasks (seed 20260904) with 973 additional tasks generated in three shards (seeds 20270904–20270906). Only the 4,000 synthetic tasks enter the main mixed-training dataset; the 198 real tasks in the historical 4,198-example bundle do not.

Teacher supervision. For each model, a frozen copy of the base model produces greedy Full-context responses with thinking disabled. Training teacher-forces the saved response tokens in both branches and minimizes teacher-to-student output-distribution KL on response positions. Prompt positions are not directly supervised. The token losses are averaged within each response and then across requests, giving each request equal weight rather than weighting it by response length. The student uses independently cached artifact KV.

The bundled teacher-response pool has maximum retained prefix lengths of 1,024 tokens for BigCodeBench and 256 for PersonaMem-v2. MuSiQue prefixes have limits of 128 tokens for Qwen3-4B and 256 for Qwen3-8B. A retained prefix need not be a complete answer. In particular, all 4,000 selected Qwen3-4B MuSiQue trajectories are legacy fixed prefixes; 3,793/4,000 Qwen3-4B and 3,907/4,000 Qwen3-8B PersonaMem-v2 trajectories reach their 256-token limit. These training-prefix limits are separate from evaluation generation limits.

A.5 Training and Generation Settings

We use Hugging Face Transformers and BF16 computation. Query adapters target `q_proj` in every Transformer layer. The low-rank matrix A_ℓ is initialized with Kaiming uniform initialization and B_ℓ with zeros, so the initial update is zero. All base-model parameters are frozen. Table 7 summarizes the shared training settings for both model sizes.

Generation. Evaluation uses greedy decoding with thinking disabled and one request per model forward. Independent workers can process different requests in parallel for quality evaluation. All methods use the same dataset-specific generation budget and retain the benchmark's required answer format. Quality-run timings are not used as formal latency measurements.

Deployment. The learned update can be merged into the query projection as $W_\ell^Q \leftarrow W_\ell^Q + (\alpha/r)B_\ell A_\ell$ for online inference. Artifact caches are constructed with the original, unadapted model. The main PIC and ATTUNER runtime uses PyTorch scaled dot-product attention (SDPA), with causal masks and position IDs specified for the assembled context.

Table 7: **Shared training configuration for Qwen3-4B and Qwen3-8B.** Global batch size includes all workers and accumulation steps.

Setting	Value
Adapter	
Projection	Query only
Layers	L0–35 (all layers)
Rank / scale	$r = 8$, $\alpha = 16$
Optimization	
Optimizer	AdamW
Learning rate	2×10^{-4}
Weight decay	0.01
Gradient clipping	Max. norm 1.0
Training schedule	
Global batch	16 requests
Micro-batch / GPU	1 request
Duration	1 epoch / 750 updates
Checkpoint	Final epoch
Mixture	4,000 per source
Precision	BF16

A.6 Analysis Settings

Distractor evaluation. The quality comparison in Figure 2(a) uses fixed cohorts of 667 BigCodeBench examples and 2,401 MuSiQue examples. All annotated relevant artifacts are retained, and distractor sets are nested across conditions. For BigCodeBench, we increase the target total number of skills to 4, 8, 12, and 16, appending non-annotated skills from the evaluation-side skill pool. At the smallest setting, nine examples retain five skills because all five are annotated as relevant. For MuSiQue, we use examples with 20 candidate documents and add 0, 2, 4, 8, or 16 non-supporting documents. Full and PIC receive identical inputs within each condition.

Attention-output alignment. The analysis in Figure 8 uses 128 examples each from MuSiQue and BigCodeBench. Full, PIC, and ATTUNER receive identical prompts and condition on the same saved Full-response prefixes. We compare per-head attention outputs before the output projection, separately for online prompt and answer positions. Similarities are averaged over positions within each example, then over heads and examples; overall summaries additionally average over layers. Each example receives equal weight.

B Routing Recovery Case Studies

We extend the main-text MuSiQue analysis with one example each from CHAMP, BigCodeBench, and PersonaMem-v2. The BigCodeBench and PersonaMem-v2 examples were selected before inspecting their attention tensors.

Measurements. Figure 9 compares Full and PIC for cached values, attention over artifact tokens, and attention outputs; the final panel compares oracle outputs with Full. Value cosine similarity is computed for each layer and KV head by concatenating value vectors across artifact tokens. Attention and output similarities are computed for each layer and query head at the final prompt token, with outputs measured before the output projection. All panels share a cosine scale of $[-0.4, 1]$.

Table 8: **Attention-output recovery.** Similarity to Full at the final prompt token, averaged over layer–head pairs. Each row is one example.

	Cosine ↑		Rel. L2 ↓	
Dataset	**PIC**	**Oracle**	**PIC**	**Oracle**
CHAMP	.859	.988	.531	.131
BigCodeBench	.892	.989	.474	.124
PersonaMem-v2	.910	.985	.418	.140

Recovery with unchanged artifact KV. The oracle replaces complete attention rows with the Full reference while preserving artifact KV; online states and their values evolve under the intervention. Across all three examples, this raises mean output cosine similarity to at least 0.985 and reduces relative L2 error (Table 8). These cases illustrate that independently cached values can support outputs close to Full when read with restored routing. They do not establish that value changes are harmless or that exact Full representations are recovered. The visualizations concern one query position per example; task-level recovery is evaluated separately in the main text.

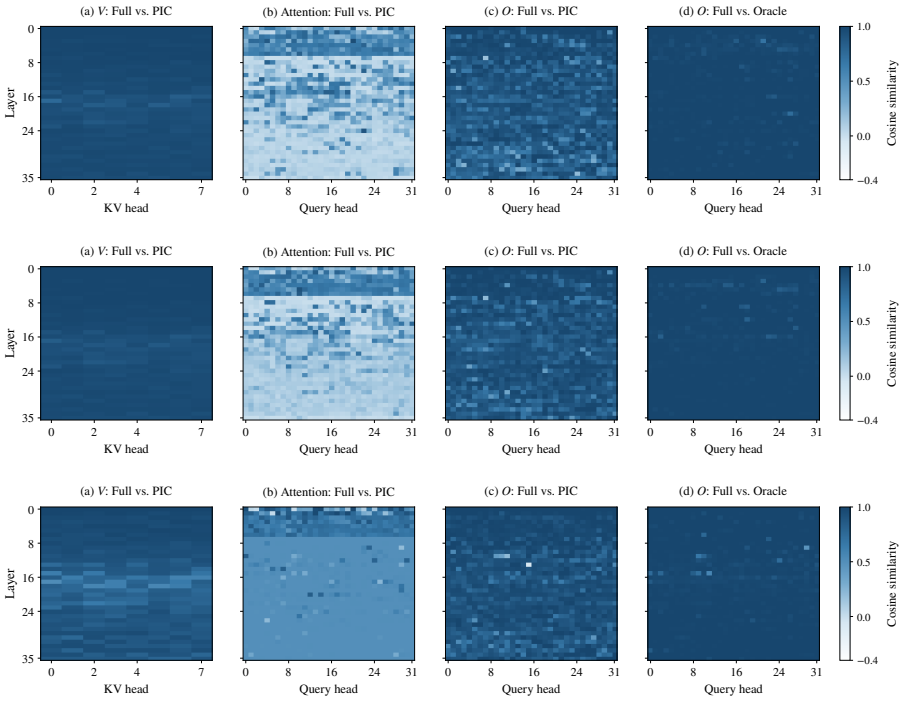


Figure 9: **Routing recovery across artifact types.** Rows: CHAMP, BigCodeBench, and PersonaMem-v2. Columns: Full–PIC cosine similarity for (a) cached values, (b) artifact attention, and (c) outputs; (d) Full–oracle output similarity. Panels (b–d) use the final prompt token. All panels share the same color scale.

C BASELINE IMPLEMENTATIONS

We evaluate all methods with the Qwen3 backbones, prepared evaluation requests, and scoring procedures described in Appendix A. Table 9 summarizes their mechanisms and configurations. We adapt the available public implementations to Qwen3; AgentKVShift is a paper-based reimplementation because we could not identify an author-released repository. The reported comparisons concern these implementations under our common evaluation protocol, rather than the original systems' serving performance.

Table 9: **Baseline mechanisms and configurations.** Layer indices are zero-based. COMB corresponds to the `encoder_pic` implementation name.

Method	Mechanism	Configuration
Training-free		
APE	Calibrated attention over independently encoded blocks	Temperature 0.9; scale 0.9
EPIC	Recompute artifact boundary tokens	First 4 tokens per eligible artifact
CacheBlend	Selective recomputation based on value deviation	Ratio 0.16; check layer L1; original-position causal mask
AgentKVShift	Probe-based key/value correction	Probe ratio 0.10; check layer L1
Training-based		
KVLink	Learned link tokens and model fine-tuning	5 link tokens per artifact; response CE
COMB	Independent encoder and gated cross-attention	9 encoder layers and cross-attention insertion points; response CE

C.1 REFERENCE METHODS

Full. Full causally prefills the complete request, allowing each artifact to condition on its actual preceding context. It provides the reference quality and latency for each matched evaluation cohort.

PIC. PIC independently encodes each artifact using local positions starting at zero. At inference, cached blocks are assembled in prompt order, while uncached text is computed with its causal history. Artifact tokens are not recomputed. The efficiency implementation processes online positions layer by layer, as described in Appendix A.2.

PIC-Pos. This diagnostic control independently encodes artifacts at their target positions in the assembled prompt. It changes positional coordinates without restoring cross-artifact conditioning. Because cache construction requires the target placement, PIC-Pos is a position control rather than a placement-independent cache-building strategy.

C.2 TRAINING-FREE BASELINES

APE. APE (Yang et al., 2025d) separately computes attention over cached context and prefix/online tokens, then combines their outputs using calibrated log-sum-exp terms. We use temperature 0.9 and scale 0.9, following the public implementation.[3] Our port supports Qwen3 query/key normalization and RoPE, using fused attention outputs and log-sum-exp values where supported. Group normalization is computed in FP32. A zero-length shared prefix is allowed for prompts that begin directly with cached artifacts.

Cache boundaries matter for this comparison. The original port can include inter-artifact separators in independently encoded context blocks. In the BigCodeBench and TheoremQA boundary-controlled runs, only artifact content is cached: separators are processed online with their causal history. We retain APE's shared-prefix encoding and attention calibration. These runs are a cache-boundary adaptation of APE, and their quality and TTFT must be paired within that layout. APE

[3] https://github.com/Infini-AI-Lab/APE

calibrates attention during online reading; it does not first run PIC and then recompute artifact tokens.

EPIC. Following EPIC's public LegoLink prototype,[4] we recompute the first four tokens of each eligible artifact from the first Transformer layer (Hu et al., 2025). An artifact beginning at position zero is skipped because it has no preceding context to restore. All online prompt tokens are also computed. Cached keys are stored before RoPE and rotated to their target positions during assembly; sparse attention masks follow each query's original position. Four tokens is one supported configuration, rather than a universal EPIC default. The prototype's internal name `kvlink-4` denotes this boundary repair and is unrelated to the trained KVLink baseline below.

CacheBlend. CacheBlend (Yao et al., 2025) ranks token positions by the difference between freshly computed and cached values, then propagates selected tokens through subsequent layers. We use check layer L1 and a selection ratio of 0.16, retaining the complete online suffix. At the check layer, selection occurs before attention; later layers update the selected positions while reusing the remaining cached states.

Our final Qwen3 variant uses a causal mask indexed by the queries' original token positions. The inspected public prototype instead uses a bottom-right-aligned mask for sparse queries; the two masks need not permit the same keys.[5] We also preserve the entire suffix, correcting the prototype example's one-token suffix-length calculation. We therefore report this method as a Qwen3 adaptation of CacheBlend. Results from the earlier mask variant are not interchangeable with the revised variant's quality or timing.

AgentKVShift. We implement the paper's probe-based correction (Pandey et al., 2026). Within each artifact, key and value probes are selected separately using deviation scores, with a probe ratio of 0.10. Their union determines which tokens are recomputed. Layer-wise probe residuals estimate key and value corrections for the remaining cached tokens; key and value writebacks retain their respective probe sets.

The Qwen3 port concatenates KV heads when computing per-token deviations and compares keys in aligned RoPE coordinates. We use check layer L1, the first KV projection whose input has passed through a preceding attention layer. Since this extraction convention is not established by author-released code, it is an explicit implementation choice. Attention visibility follows original token positions.

C.3 Training-Based Baselines

KVLink. KVLink (Yang et al., 2025c) introduces five learned link tokens per artifact and fine-tunes the backbone.[6] Its block attention mask keeps ordinary artifact tokens within their own causal block, while link and online tokens can attend to preceding valid blocks. We extend the Qwen3 tokenizer with link tokens and train all model weights, including the new embeddings. At inference, artifact states are cached using the trained model and link tokens are computed online. This requires a method-specific cache; it does not retain the unadapted base model's artifact KV.

COMB. COMB (Zhao et al., 2026) independently encodes artifacts and exposes their representations through added cross-attention modules.[7] Our Qwen3 port uses one bidirectional encoder layer and one cross-attention block per four original decoder layers. For both 36-layer backbones, cross-attention is inserted before L3, L7, . . ., L35, giving nine insertion points. The original backbone and embeddings remain frozen; the new encoder, KV projections, cross-attention, MLPs, and gates are trained. Cross-attention and MLP gates are initialized to zero. The cached states are encoder-produced representations, rather than ordinary decoder artifact KV. We use the name COMB throughout the paper; `encoder_pic` is only the implementation identifier.

[4] https://github.com/DerekHJH/epic
[5] https://github.com/YaoJiayi/CacheBlend
[6] https://github.com/UCSB-NLP-Chang/KVLink
[7] https://github.com/shijuzhao/Comb

Matched data and training budget. Both baselines use the same 12,000 training requests and corresponding Full-context teacher responses as ATTUNER. They minimize response-token cross-entropy, retaining their supervised language-modeling objectives rather than adopting ATTUNER's distributional KL loss. Prompt tokens are excluded from the loss. Each model size receives a separately trained checkpoint.

Table 10: **Training settings for KVLink and COMB.** Both models use the same 12K requests and model-specific teacher responses as ATTUNER.

Setting	KVLink	COMB
Learning rate	5×10^{-6}	5×10^{-5}
Optimizer	AdamW	
Weight decay	0.01	
Gradient norm cap	1.0	
Global batch	16 requests	
Micro-batch / GPU	1 request	
Duration	1 epoch / 750 updates	
Precision	BF16	

We train for one epoch with a global batch of 16 requests and one request per GPU forward, yielding 750 optimizer updates. The 4B runs use two GPUs and eight accumulation steps; the 8B runs use four GPUs and four accumulation steps. AdamW uses constant learning rates of 5×10^{-6} for KVLink and 5×10^{-5} for COMB, weight decay 0.01, and gradient clipping at norm 1.0 (Table 10). We evaluate the final checkpoint. The shared data and training duration are our controlled comparison protocol, not a reproduction of the original papers' complete training corpora or schedules. The methods also differ in trainable parameter count, so equal numbers of examples and updates do not imply equal training compute.

Efficiency comparisons. Each method constructs its own reusable states before timing and places them in CPU RAM. Online transfer, cache assembly, link-token computation, cross-attention, and any repair are included when applicable, using the TTFT boundary in Appendix A.2. Offline construction and training are excluded. CacheBlend and APE quality and efficiency must use the same mask and cache-layout variants, respectively.

D SUPPLEMENTARY ANALYSES

D.1 ALTERNATIVE REPAIR INTERVENTIONS

We first examine whether positional alignment restores effective cache use, then compare component-level interventions under a matched protocol. These Qwen3-4B diagnostics complement the task-level routing oracle in the main text.

Position alignment improves keys without guaranteeing recovery. For the four-skill CHAMP example `champ_00004`, PIC-Pos increases mean key cosine similarity to Full from 0.794 to 0.975 by independently encoding artifacts at their target prompt positions (Figure 10). Nevertheless, both PIC and PIC-Pos answer incorrectly, whereas Full answers correctly. Thus, closer key representations do not guarantee task recovery in this example: position alignment does not restore the contextual conditioning omitted by independent encoding.

The heatmaps average all eight KV heads and group at most 16 adjacent tokens within each artifact for display. PIC-Pos re-encodes artifacts at target positions rather than rotating an existing cache; its values are not assumed to be bitwise identical to those from local-position encoding.

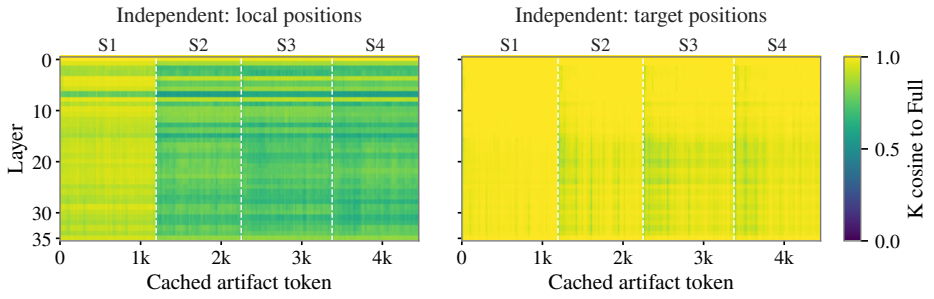


Figure 10: **Position alignment leaves a residual quality gap.** Key cosine similarity to Full for `champ_00004`, using local positions (left) or target positions (right). S1–S4 mark skill boundaries. Both panels average all KV heads and share a color scale.

Matched interventions isolate alternative recovery paths. We evaluate 32 fixed randomly sampled examples each from MuSiQue and BigCodeBench. All branches use identical prompts and the same saved Full-response prefixes, capped at 64 tokens. Starting from PIC, we separately replace artifact K, V, or both with their Full counterparts before online prefill. We also restore Full attention rows (routing) or complete attention outputs (O, before the output projection) at online prompt positions after the first artifact and subsequent response inputs. All interventions span L0–35. Routing restoration retains artifact KV and applies Full attention weights to the intervention branch's own values; online states evolve under the intervention.

Table 11: **All-layer component interventions.** Mean output KL to Full; lower is better. Full KV and Full O are separate reconstruction controls.

Intervention	MuSiQue	BCB
PIC	0.8480	0.1221
PIC-Pos	0.8536	0.0850
Full K	0.8104	0.0447
Full V	13.5594	8.4155
Full routing	**0.1059**	**0.0086**
Full KV / Full O	0.0000	0.0000

We measure $D_{\mathrm{KL}}(p^{\mathrm{Full}} \| p^{\mathrm{intervention}})$, averaging over response positions within each request and then equally across requests. This measures output-distribution recovery under shared prefixes, rather than free-generation task accuracy.

Restoring routing consistently reduces output divergence. Full routing reduces mean output KL by **87.5% on MuSiQue and 93.0% on BigCodeBench** (Table 11). Paired bootstrap 95% intervals are [81.4%, 92.8%] and [88.0%, 95.8%], respectively, using 2,000 request-level resamples. The reduction occurs in 31/32 MuSiQue examples and all 32 BigCodeBench examples (Figure 11). In contrast, replacing only K produces dataset-dependent gains, while replacing only V substantially increases KL. Full KV and Full O recover Full logits to numerical precision, serving as positive reconstruction controls.

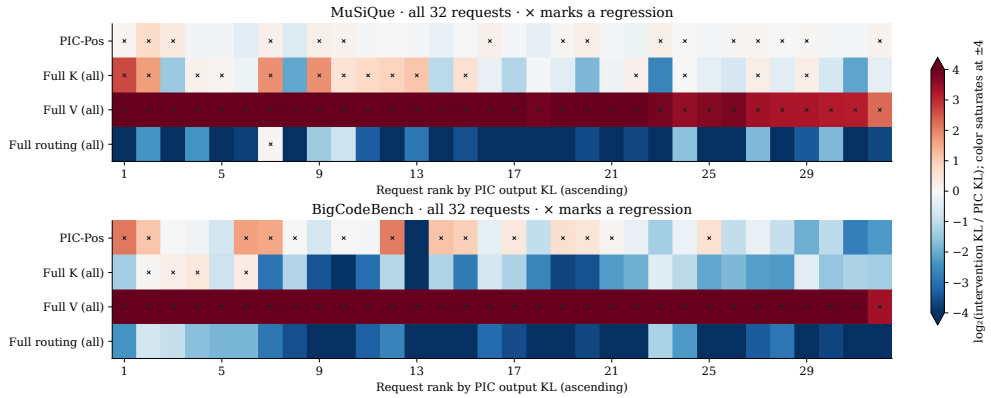


Figure 11: **Recovery across individual requests.** Columns represent examples ordered by PIC output KL within each dataset. Colors show $\log_2(D_{\mathrm{KL}}^{\mathrm{intervention}}/D_{\mathrm{KL}}^{\mathrm{PIC}})$: blue indicates improvement and red deterioration. Crosses mark regressions; colors saturate at ± 4. All component interventions span every layer.

Implications for cache-preserving recovery. Together, these results show that **restoring routing can substantially recover Full-like output distributions without replacing cached artifact values**. They do not establish that values are unimportant: the poor V-only result instead shows that inserting Full values into the remaining PIC computation can be disruptive. Nor does improved representation similarity guarantee behavioral recovery: under all-layer K replacement, 14/32 MuSiQue examples exhibit higher mean answer-position O cosine similarity but worse output KL. These diagnostics therefore support adapting how the cache is read, while the main-text routing oracle provides the corresponding task-level evidence.

D.2 Limitations and Future Directions

Our evaluation covers skills, documents, and memories as separate task settings, but does not examine requests that combine all three artifact types. We focus on PIC in models with full attention; applicability to other attention architectures remains unexplored. Our contribution concerns effective cache reading rather than cache storage or transport. Although our TTFT measurements include CPU-to-GPU cache transfer, we do not optimize cache placement, eviction, or communication. Looking ahead, our results motivate exploring cache reading as a post-training objective, enabling models to better use context prepared independently of the current request. Combining ATTUNER with KV compression and retrieval-based artifact selection is a promising direction for extending cache reuse to larger contexts and tighter memory budgets.

E Detailed Results by Evaluation Domain

Tables 12 and 13 reproduce the in-domain and out-of-domain portions of Table 2 separately for readability. All scores, timings, aggregate values, and rankings are unchanged. In-domain tasks are BigCodeBench (BCB), MuSiQue (MSQ), and PersonaMem-v2 (PMem); out-of-domain tasks are TheoremQA (TQA) and CHAMP (CHP), neither of which is included in adapter training. The domain distinction concerns training coverage, not artifact count: TheoremQA uses one artifact, while CHAMP contains both single- and multiple-artifact requests.

Table 12: **In-domain quality and efficiency.** Scores are pass@1 for BCB, F1 for MSQ, and accuracy for PMem (all in percent). TTFT is median warm-cache latency in milliseconds. Avg is the arithmetic mean of scores; Δ is its change from Full in percentage points. Geo. is the geometric mean of TTFT, and speedup is relative to Full. Full and PIC are excluded from ranking; bold and underlined entries mark the best and second-best results among the other methods, including ties.

Method	BCB		MSQ		PMem		Quality		Efficiency	
	Score ↑	TTFT ↓	Score ↑	TTFT ↓	Score ↑	TTFT ↓	Avg ↑	Δ	Geo. ↓	Speedup ↑
Qwen3-4B										
Full	44.53	420.77	61.01	39.06	53.28	62.38	52.94	–	100.83	1.00×
PIC	39.58	99.60	48.69	35.51	50.34	47.56	46.20	−6.74	55.20	1.83×
Training-free										
APE	39.58	114.59	41.02	39.17	43.72	51.92	41.44	−11.50	61.54	1.64×
EPIC	44.08	100.57	54.21	50.76	50.80	67.29	49.70	−3.24	70.03	1.44×
CacheBlend	**45.13**	194.81	54.82	52.43	**53.92**	69.73	51.29	−1.65	89.30	1.13×
AgentKVShift	42.13	196.54	52.62	68.78	47.42	90.25	47.39	−5.55	106.85	0.94×
Training-based										
KVLink	44.98	165.04	**59.93**	98.70	52.54	109.94	**52.48**	−0.46	121.44	0.83×
COMB	35.68	**38.29**	12.70	**28.68**	32.22	**43.49**	26.87	−26.07	**36.28**	**2.78**×
ATTUNER	44.08	101.27	58.06	36.71	51.86	47.75	51.33	−1.61	56.20	1.79×
Qwen3-8B										
Full	50.22	660.41	63.28	62.89	43.90	99.14	52.47	–	160.28	1.00×
PIC	47.08	113.82	50.00	35.33	35.50	68.60	44.19	−8.27	65.10	2.46×
Training-free										
APE	46.63	123.16	44.94	39.60	44.86	73.10	45.48	−6.99	70.91	2.26×
EPIC	48.58	110.50	56.71	50.79	42.40	88.41	49.23	−3.24	79.17	2.02×
CacheBlend	48.43	249.41	58.12	54.92	42.78	96.36	49.78	−2.69	109.69	1.46×
AgentKVShift	47.53	246.76	55.58	68.88	40.92	106.07	48.01	−4.46	121.71	1.32×
Training-based										
KVLink	50.22	182.91	61.20	102.70	**46.56**	143.02	**52.66**	+0.19	139.02	1.15×
COMB	42.13	**48.17**	14.90	**31.02**	31.62	**67.51**	29.55	−22.92	**46.55**	**3.44**×
ATTUNER	**50.52**	113.19	**61.22**	36.33	44.14	68.10	51.96	−0.51	65.42	2.45×

Table 13: **Out-of-domain quality and efficiency.** Scores are accuracy (%) for TQA and CHP. TTFT is median warm-cache latency in milliseconds. Avg is the arithmetic mean of scores; Δ is its change from Full in percentage points. Geo. is the geometric mean of TTFT, and speedup is relative to Full. Full and PIC are excluded from ranking; bold and underlined entries mark the best and second-best results among the other methods, including ties.

Method	TQA		CHP		Quality		Efficiency	
	Score ↑	TTFT ↓	Score ↑	TTFT ↓	Avg ↑	Δ	Geo. ↓	Speedup ↑
Qwen3-4B								
Full	71.22	89.08	69.06	139.01	70.14	–	111.28	1.00×
PIC	66.80	38.14	64.13	46.70	65.47	−4.68	42.20	2.64×
Training-free								
APE	59.57	42.88	60.54	49.83	60.06	−10.09	46.22	2.41×
EPIC	67.87	46.73	66.82	55.23	67.35	−2.80	50.80	2.19×
CacheBlend	<u>68.14</u>	50.95	**68.61**	75.92	<u>68.38</u>	−1.77	62.19	1.79×
AgentKVShift	67.07	61.59	65.47	79.94	66.27	−3.87	70.17	1.59×
Training-based								
KVLink	66.93	68.96	<u>68.16</u>	78.02	67.55	−2.60	73.35	1.52×
COMB	47.52	**30.78**	54.26	**31.18**	50.89	−19.25	**30.98**	**3.59×**
ATTUNER	**68.81**	<u>39.80</u>	<u>68.16</u>	50.62	**68.49**	−1.66	<u>44.89</u>	<u>2.48×</u>
Qwen3-8B								
Full	63.05	145.85	62.78	227.68	62.92	–	182.23	1.00×
PIC	55.15	38.89	63.23	59.17	59.19	−3.73	47.97	3.80×
Training-free								
APE	52.34	45.10	51.12	60.80	51.73	−11.19	52.36	3.48×
EPIC	58.23	58.89	61.88	67.53	60.06	−2.86	63.06	2.89×
CacheBlend	57.56	66.42	42.60	101.32	50.08	−12.84	82.03	2.22×
AgentKVShift	**61.58**	80.25	64.13	105.39	**62.86**	−0.06	91.96	1.98×
Training-based								
KVLink	58.50	70.52	**66.37**	92.46	62.44	−0.48	80.75	2.26×
COMB	42.84	<u>42.24</u>	51.57	**44.33**	47.21	−15.71	**43.27**	**4.21×**
ATTUNER	<u>60.37</u>	**39.80**	<u>65.02</u>	<u>59.98</u>	<u>62.70</u>	−0.22	<u>48.86</u>	<u>3.73×</u>

F DATASET EXAMPLES

We show one relevant-only example for each artifact type: skills from BigCodeBench, documents from MuSiQue, and memory messages from PersonaMem-v2. Artifacts are cached independently, while current questions and task instructions remain online. All skill, document, and memory contents are shown in full. Only line wrapping and typographic punctuation are normalized. Block headings, reference answers, reference code, and tests are provided for illustration and are not added to the evaluated prompts.

F.1 BIGCODEBENCH: REUSABLE SKILLS

Example `bigcodebench_00186` supplies two annotated skills: `geopy` for geographic distances and `Folium` for map construction. Each complete skill is a separate cached artifact; both are reproduced below without omissions. The two skill blocks precede the task in a single user message, introduced by `Relevant Skill:` and separated by `---`.

Cached artifact 1: geopy skill (full text).

````
# geopy -- Distance Calculation and Geocoding
## Geodesic Distance
`geodesic` takes coordinate tuples `(lat, lon)` -- never dicts or
    lists with named keys:
```python
from geopy.distance import geodesic
dist = geodesic((48.8566, 2.3522), (51.5074, -0.1278)).kilometers
# .miles and .meters are also available
```
## Pairwise Distances Between Locations
When the input is a dict like `{'City': {'Lat': ..., 'Lon': ...},
    ...}`, extract tuples before calling `geodesic`. The canonical
    pattern uses a flat list of `(name, lat, lon)` tuples:
```python
def compute_distances(dic):
    if not dic:
        raise ValueError("Input dictionary is empty.")
    locations = [(k, v['Lat'], v['Lon']) for k, v in dic.items()]
    distances = {}
    for i in range(len(locations)):
        for j in range(i + 1, len(locations)):
            d = geodesic(
                (locations[i][1], locations[i][2]),
                (locations[j][1], locations[j][2])
            ).kilometers
            distances[(locations[i][0], locations[j][0])] = d
    return distances
```
## Geocoding with Photon
`Photon` converts address strings to coordinates. Always pass a
    `user_agent` string:
```python
from geopy.geocoders import Photon
geolocator = Photon(user_agent="geoapiExercises")
result = geolocator.geocode("Berlin, Germany")
lat, lon = result.latitude, result.longitude
```
## Mixed Input (coordinates or address strings)
When input values are either coordinate dicts `{'Lat': ..., 'Lon':
    ...}` or plain address strings, use `isinstance` to distinguish
    them. Instantiate the geolocator **once** outside the loop, and
    call `geocode()` only for string values -- not for dict values:
```python
def preprocess_locations(dic):
    geolocator = Photon(user_agent="geoapiExercises")
    locations = []
````
````

```
    for name, value in dic.items():
        if isinstance(value, dict) and 'Lat' in value and 'Lon' in
    value:
            lat, lon = value['Lat'], value['Lon']
        elif isinstance(value, str):
            result = geolocator.geocode(value)
            lat, lon = result.latitude, result.longitude
        else:
            raise ValueError(f"Invalid location format for '{name}':
    {value}")
        locations.append({'Location': name, 'Lat': lat, 'Lon': lon})
    return locations
```
Calling `geocode()` on a dict value instead of only on string values
    causes the geocoder to be invoked the wrong number of times.

Cached artifact 2: Folium skill (full text).

```
# Folium -- Interactive Map Creation
Folium creates Leaflet.js maps in Python. Core pattern: create
    `folium.Map`, add `folium.Marker` objects, return the map.
## Creating a Map and Adding Markers
`folium.Map(location=...)` and `folium.Marker(location=...)` both
    require a sequence `[lat, lon]` -- **not** a dict. When data uses
    `{'Lat': ..., 'Lon': ...}` keys, extract them explicitly:
```python
import folium
m = folium.Map(location=[lat, lon], zoom_start=4)
for name, coords in dic.items():
    folium.Marker(
        location=[coords['Lat'], coords['Lon']],
        popup=name
    ).add_to(m)
return m
```
Building a DataFrame then Mapping Cities
When coordinates are generated with a seeded RNG, generate lon and lat
    per city in a loop -- one `np.random.uniform` for longitude
    then one for latitude. Vectorized bulk generation consumes the RNG
    in a different order:
```python
import pandas as pd
import numpy as np
import folium
def make_city_map(dic, cities):
    lon_min, lon_max = dic['Lon']
    lat_min, lat_max = dic['Lat']
    data = {'City': [], 'Longitude': [], 'Latitude': []}
    for city in cities:
        data['City'].append(city)
        data['Longitude'].append(np.random.uniform(lon_min, lon_max))
        data['Latitude'].append(np.random.uniform(lat_min, lat_max))
    df = pd.DataFrame(data)
    m = folium.Map(location=[df['Latitude'].iloc[0],
    df['Longitude'].iloc[0]], zoom_start=2)
    for _, row in df.iterrows():
        folium.Marker([row['Latitude'], row['Longitude']],
    popup=row['City']).add_to(m)
    return m, df
```
Mixing Dict Coordinates and Address Strings
When input values can be `{'Lat': ..., 'Lon': ...}` dicts or plain
    address strings, use `isinstance` to distinguish them. Call
```

````
    `geocode()` only for string values -- not for dict values.
    Instantiate the geolocator once outside the loop:
```python
from geopy.geocoders import Photon
def map_locations(dic):
    geolocator = Photon(user_agent="geoapiExercises")
    preprocessed = []
    for name, value in dic.items():
        if isinstance(value, dict) and 'Lat' in value and 'Lon' in
    value:
            lat, lon = value['Lat'], value['Lon']
        elif isinstance(value, str):
            result = geolocator.geocode(value)
            lat, lon = result.latitude, result.longitude
        else:
            raise ValueError(f"Invalid location format for '{name}':
    {value}")
        preprocessed.append({'Location': name, 'Lat': lat, 'Lon': lon})
    locations_df = pd.DataFrame(preprocessed)
    first = locations_df.iloc[0]
    m = folium.Map(location=[first['Lat'], first['Lon']], zoom_start=4)
    for _, row in locations_df.iterrows():
        folium.Marker([row['Lat'], row['Lon']],
    popup=row['Location']).add_to(m)
    return m
```
## Computing Pairwise Geodesic Distances
Extract `(lat, lon)` tuples before calling `geodesic`. The canonical
    approach uses a flat list of tuples indexed by position:
```python
from geopy.distance import geodesic
locations = [(name, v['Lat'], v['Lon']) for name, v in dic.items()]
distances = {}
for i in range(len(locations)):
    for j in range(i + 1, len(locations)):
        d = geodesic(
            (locations[i][1], locations[i][2]),
            (locations[j][1], locations[j][2])
        ).kilometers
        distances[(locations[i][0], locations[j][0])] = d
```
`geodesic()` takes `(lat, lon)` tuples -- never a dict.
## Map Object Reference
`m.location` returns the center as `[lat, lon]`. The object type is
    `folium.folium.Map`.
````

Online task (same user message).

```
Generates a Folium map with markers for specified locations and
    calculates the geodesic distances between each pair of locations.
The function should raise the exception for: ValueError: If the input
    dictionary is empty.
The function should output with:
    tuple: A tuple containing a Folium map object and a dictionary
    with pairs of location
    names as keys and their distances in kilometers as values.
You should write self-contained code starting with:
```
from geopy.distance import geodesic
import folium
def task_func(dic):
```
```

Evaluation target. The generated `task_func` implementation is evaluated against the task's unit tests. It must return the map and pairwise-distance dictionary and handle the specified empty-input error. The reference implementation and tests are not included in the prompt.

Reference implementation (evaluation only). This is the prepared dataset's reference code, not a generated model response. It is constructed from the saved code prompt and solution.

```
from geopy.distance import geodesic
import folium
def task_func(dic):
    if not dic:
        raise ValueError("Input dictionary is empty.")
    locations = [(k, v['Lat'], v['Lon']) for k, v in dic.items()]
    distances = {}
    folium_map = folium.Map(location=[locations[0][1],
    locations[0][2]], zoom_start=4)
    for i in range(len(locations)):
        folium.Marker([locations[i][1], locations[i][2]],
    popup=locations[i][0]).add_to(folium_map)
        for j in range(i + 1, len(locations)):
            distance = geodesic((locations[i][1], locations[i][2]),
    (locations[j][1], locations[j][2])).kilometers
            distances[(locations[i][0], locations[j][0])] = distance
    return folium_map, distances
```

Complete task-specific unit tests (evaluation only). The following is the full saved test suite. During scoring, the generated function is made available to these tests by the benchmark evaluator; the suite is not part of the model's request.

```
import unittest
from unittest.mock import patch
import folium  # Assuming the function task_func and folium are
    imported or defined appropriately.
class TestCases(unittest.TestCase):
    def test_return_type(self):
        """Test that the function returns a tuple with a map and a
    dictionary."""
        result = task_func({'Loc1': {'Lat': 0, 'Lon': 0}, 'Loc2':
    {'Lat': 1, 'Lon': 1}})
        self.assertIsInstance(result, tuple)
        self.assertIsInstance(result[0], folium.folium.Map)
        self.assertIsInstance(result[1], dict)
    def test_distances_calculation(self):
        """Test the accuracy of the distance calculation. Assumes the
    distance is reasonable for nearby points."""
        _, distances = task_func({'Loc1': {'Lat': 0, 'Lon': 0},
    'Loc2': {'Lat': 0, 'Lon': 1}})
        self.assertTrue(0 < distances[('Loc1', 'Loc2')] < 200)  #
    Rough check for distance in kilometers
    def test_multiple_locations(self):
        """Test functionality with multiple locations."""
        _, distances = task_func({'Loc1': {'Lat': 0, 'Lon': 0},
    'Loc2': {'Lat': 0, 'Lon': 1}, 'Loc3': {'Lat': 1, 'Lon': 1}})
        self.assertEqual(len(distances), 3)  # Expecting 3 pairs of
    locations
    def test_marker_addition(self):
        """Test that markers are correctly added to the map. Assumes 1
    TileLayer present."""
        folium_map, _ = task_func({'Loc1': {'Lat': 0, 'Lon': 0}})
        self.assertEqual(len(folium_map._children), 2)  # One for
    TileLayer and one for Marker
```

Unit tests (continued).

```
    @patch('geopy.distance.geodesic')
    def test_distance_dict_structure(self, mock_geodesic):
        """Ensure the distance dictionary has the correct key-value
    structure."""
        mock_geodesic.return_value.kilometers = 100  # Mock distance
    as 100 km
        _, distances = task_func({'Loc1': {'Lat': 0, 'Lon': 0},
    'Loc2': {'Lat': 0, 'Lon': 1}})
        self.assertTrue(all(isinstance(key, tuple) and
    isinstance(value, float) for key, value in distances.items()))
    def test_empty_input(self):
        """Test function behavior with an empty dictionary input
    raises ValueError."""
        with self.assertRaises(ValueError):
            task_func({})
    def test_single_location(self):
        """Test handling of a single location input."""
        folium_map, distances = task_func({'Loc1': {'Lat': 0, 'Lon':
    0}})
        self.assertEqual(len(distances), 0)  # No distances calculated
        self.assertEqual(len(folium_map._children), 2)  # One for
    TileLayer and one for Marker
    def test_negative_lat_lon(self):
        """Test handling of negative latitude and longitude values."""
        _, distances = task_func({'Loc1': {'Lat': -34, 'Lon': -58},
    'Loc2': {'Lat': -33, 'Lon': -70}})
        self.assertTrue(all(value >= 0 for value in
    distances.values()))  # Distance should be positive
    def test_large_distance_calculation(self):
        """Test accuracy for large distances, e.g., antipodal
    points."""
        _, distances = task_func({'Loc1': {'Lat': 0, 'Lon': 0},
    'Loc2': {'Lat': 0, 'Lon': 180}})
        self.assertTrue(distances[('Loc1', 'Loc2')] > 10000)  #
    Expecting a large distance
```

What the example illustrates. The task combines two reusable library skills: coordinate tuples feed geodesic-distance calculations, while Folium creates the map and markers. Correctness is determined by the task-specific tests, rather than by textual agreement with the reference implementation.

F.2 MuSiQue: Supporting Documents

Example `2hop__460946_294723` contains two annotated supporting documents. Each cached artifact includes its document index, title, and paragraph. The original indices and relative order are retained; the other candidate documents are excluded. The system instruction, `Documents:` heading, question, and answer-format instruction are outside the cached document spans.

System prompt.

```
You answer multi-hop questions using only the provided documents.
```

Cached artifact 1: supporting document.

```
[Document 5]
Title: Miquette Giraudy
Miquette Giraudy (born 9 February 1953, Nice, France) is a keyboard
    player and vocalist, best known for her work in Gong and with her
    partner Steve Hillage. She and Hillage currently form the core of
    the ambient band System 7. In addition to her performances in
    music, she has also worked as an actress, film editor and writer.
    In each role, she has used different stage names.
```

Cached artifact 2: supporting document.

```
[Document 10]
Title: Green (Steve Hillage album)
Green is the fourth studio album by British progressive rock musician
    Steve Hillage. Written in spring 1977 at the same time as his
    previous album, the funk-inflected "Motivation Radio" (1977),
    "Green" was originally going to be released as "The Green Album" as
    a companion to "The Red Album" (the originally intended name for
    "Motivation Radio"). However, this plan was dropped and after a US
    tour in late 1977, "Green" was recorded alone, primarily in
    Dorking, Surrey, and in London.
```

Online question and answer-format instruction.

```
Question: Who is the spouse of the Green performer?

Give a concise answer. Put the final answer on its own last line
    exactly as 'ANSWER: <answer>'.
```

Reference answer (shown in the requested output format).

```
ANSWER: Miquette Giraudy
```

Evidence linking. The Green document identifies Steve Hillage as the performer; the other document names Miquette Giraudy as his partner. The dataset reference answer therefore requires linking information across the two documents.

F.3 PERSONAMEM-V2: RELEVANT MEMORY MESSAGES

Example `personamem-v2:689:04277` supplies two messages from the annotated related conversation snippet. Each complete historical message, including its role, forms a separate artifact. We preserve their order and the saved answer-option mapping; unannotated history and persona background are not added. The current question and subsequent system message remain online.

Cached artifact 1: historical user message.

```
Can you help me make this sound a bit smoother?
```

Cached artifact 2: historical assistant message.

```
It's been raining all afternoon, so I spent the day indoors working on
    new drum fills and playing a few 2K matches. It's been a nice,
    laid-back day.
```

Online user question.

```
What are some cozy and relaxing activities I could do at home when I
    have a free afternoon? Please recall my related preferences from
    our conversation history to give personalized responses.
```

Online system message: options and answer-format instruction.

```
Please choose the best answer from the following options:

A. On a free afternoon, you could make your favorite warm drink, put
    on some calming music, and work on a personal hobby. Later, you
    could read a good book or watch a movie to relax before dinner.
B. On a quiet afternoon, you could brew a fresh cup of masala chai,
    listen to some Bollywood classics, and help your mom with a
    home-cooked dinner. Later, you could chat with family about cricket
    matches and end the day watching a classic Shah Rukh Khan movie.
C. On a rainy afternoon, you could make some chai, put on your
    favorite jazz playlist, and sketch some new sneaker designs while
    listening to the rain. Later, you could practice a few new drum
    fills and then wind down by watching NBA highlights or an inspiring
    drumming tutorial.
D. On a free afternoon, you could make some iced lemonade, play around
    with a new pasta recipe, and then spend time reading travel blogs.
    Later, you could try learning a bit of Italian on a language app
    and finish the day with a feel-good movie.

Think step by step about which answer best fits the user's query and
    conversation context. Provide your reasoning first, then give your
    final answer as 'Final Answer: [Letter]'
```

Reference option (shown in the requested output format).

```
Final Answer: C
```

Memory-dependent choice. The historical assistant message mentions drum practice and basketball video games. The annotated answer is option C, which includes drum practice; the prepared option order and label are retained exactly. This is a multiple-choice reference, not a model-generated response.