DMC-5: DUAL-MODEL COLLABORATION WITH CACHE-AWARE CONTEXT COMPRESSION FOR LONG-HORIZON AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon agents repeatedly feed an ever-growing interaction trajectory to a specialized agent model. Prefix caching reuses computation over repeated history, but cannot prevent context growth or remove information irrelevant to the current decision. This is especially costly when the specialized agent lacks the cached-input discounts available to general-purpose models, making trajectory compression a natural alternative. However, existing compression methods on selective context risk losing global information, recursively rewriting summary leads to semantic drift, while repeatedly recompressing the full history shifts the growing cost to the compressor. Therefore, we introduce **DMC-5** (Dual-Model Collaboration with CaChe-aware Context Compression), an asymmetric framework that assigns full-history integration to a general compression model and action generation to a specialized agent model. DMC-5 maintains the raw trajectory as an Append-only Canonical View, preserving global access while allowing the compressor to reuse a stable historical prefix instead of recursively rewriting or recomputing prior context. The agent operates only on a compact Decision View with recent raw interactions, reducing repeated input while retaining the information needed for the next action. DMC-5 further incorporates **YAMATO** (integritY-Aware Memory Abstractions for Tool Operations) to preserve execution-critical information during lossy compression. Experiments on AppWorld, OfficeBench, and 8-objective QA show that DMC-5 reduces agent model input by 38.5$\sim$57.0% and end-to-end cost by 14.5$\sim$28.0% with 78.0$\sim$85.0% compressor cache-hit rates, while improving task performance by 1.0$\sim$5.8% over the uncompressed baseline.

1 INTRODUCTION

Long-horizon, tool-using agents repeatedly ask a specialized decision model to reread an ever-growing trajectory of reasoning, actions, and observations (Yao et al., 2023; Packer et al., 2023). As shown in Fig. 1(a), an LLM agent repeatedly rereads a growing history, although only a small fraction is relevant to current decision, increasing cost and distracting the model (Laban et al., 2026). Prefix caching reuses stable prefixes (OpenAI, 2026; Song, 2026) to reduce computation, but it neither shortens the context length nor removes irrelevant content. This cost is especially pronounced for fine-tuned or customized agent endpoints, for which cached-input discounts are often unavailable or limited than those offered for general models. (OpenAI, 2026; Google Cloud, 2026; Microsoft Azure, 2026; Alibaba Cloud, 2026). To address this issue, trajectory compression shifts history integration to a lower-cost compressor, reducing the context repeatedly processed by the agent model.

Existing compressors reduce agent-side context through three update patterns in Fig. 1(b), but each incurs extra end-to-end cost. **Selective Context** (Xiao et al., 2026; Ning et al., 2026; Hu et al., 2025; Ye et al., 2026; Verma, 2026) keeps raw trajectory in sliding window and compresses context outside, reducing compression cost at the risk of missing global information and triggering repeated steps. **Recursive Summary** (Zhou et al., 2026; Yu et al., 2026; Wu et al., 2025) updates a previous summary with new interactions to avoid full-history rereading, but rewritten summary can accumulate semantic drift (Kang et al., 2026; Shi et al., 2026) and limit prefix-cache reuse. **Full-history Recompression** (Li et al., 2023; Jiang et al., 2024; Xu et al., 2024) rebuilds context from retained raw history, preserving global access but repeatedly processing the growing trajectory. Overall, reducing the agent-side context alone does not guarantee lower end-to-end cost, as compressor cache misses and extra agent steps caused by information loss can offset the savings. Taken together, these

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.

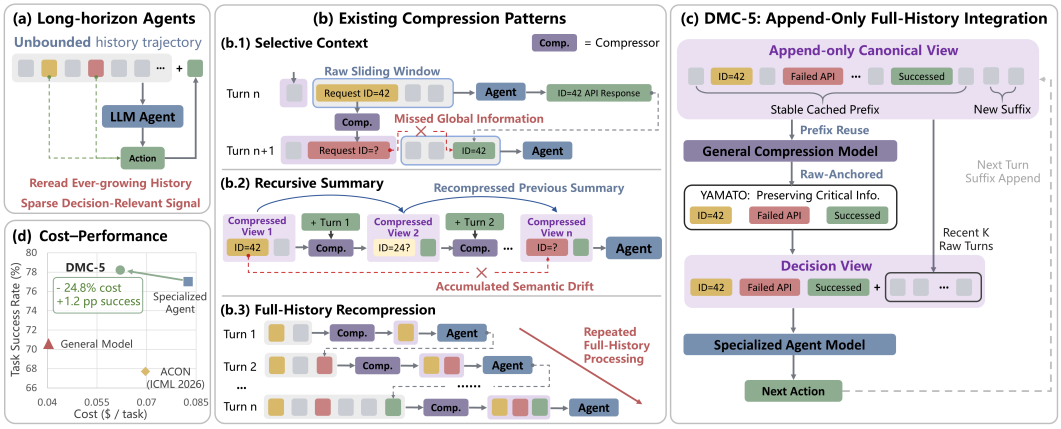

Figure 1: Motivation and overview of DMC-5. (a) Long-horizon agents repeatedly reread a growing trajectory, increasing cost while decision-relevant information is sparse. (b) Existing compression patterns trade off global information, semantic stability, and repeated compressor processing. (c) DMC-5 separates full-history integration from action generation: an Append-only Canonical View enables stable prefix reuse for the general compressor, while a compact Decision View retains recent and execution-critical information for the specialized agent. (d) DMC-5 improves the cost–performance trade-off over the uncompressed specialized agent and existing compression baselines. trade-offs show that token compression ratio alone does not determine end-to-end cost, as compressor cache misses and extra agent steps from information loss may offset agent-side token savings.

Therefore, a practical compression framework must control both compression overhead and the interaction overhead introduced by information loss. This requires the compressor and the agent to consume different views of the trajectory. An ideal compressor should retain access to the full trajectory so that previously observed information remains recoverable, while extracting the context most relevant to the current task. This full-history role can be assigned to a general model with strong cross-task generalization and discounted cached-input pricing, using an append-only trajectory view to preserve prior information and maintain stable prefix reuse across compression rounds. The agent, in contrast, should specialize in the downstream task and action space, often through fine-tuning, while consuming only a compact Decision View that keeps information critical for the next action, especially when its deployment provides no or limited cached-input discount. Together, this asymmetric design assigns full-history integration to a general compression model and compact action generation to specialized agent model, reducing both compression and interaction overhead.

Motivated by this, we introduce **DMC-5** (Dual-Model Collaboration with CaChe-Aware Context Compression), an asymmetric framework that assigns full-history integration to a general compression model and action generation to a specialized agent model. Its compressor reads an **Append-only Canonical View** containing the full-history raw trajectory, so each compression integrates the full history without recursively rewriting previous summaries, while the unchanged prefix remains reusable across compression rounds. The agent receives a **Decision View** composed of compact context and recent raw interactions, reducing repeated input while preserving critical information for the next action. To protect execution-critical state during lossy compression, DMC-5 incorporates **YAMATO** (integrityY-Aware Memory Abstractions for Tool Operations). YAMATO lets the compressor reference selected raw actions, tool results, and work-product updates through pinned pointers, and deterministically restores the information from the raw records rather than being rewritten by the model. It also maintains a persistent failure ledger and provenance tags across compression. Therefore, DMC-5 preserves full-history access during compression, keeps the agent context compact, and protects state whose corruption or omission could affect subsequent execution.

Across AppWorld, OfficeBench, and 8-objective QA, DMC-5 reduces agent-model input by 38.5∼57.0% and end-to-end cost by 14.5∼28.0%, while achieving 78.0∼85.0% compressor cache-hit rates. Compared with the uncompressed baseline, task performance improves by 1.0∼5.8% across the three benchmarks. In summary, we conclude our contributions as follows:

- We formulate trajectory compression as an end-to-end cost problem, jointly accounting for agent input, compressor overhead, prefix-cache reuse, and interaction length.

- We propose DMC-5, separating cache-aware context compression from specialized action generation through Append-only Canonical and compact Decision Views, with YAMATO preserving execution-critical information via deterministic raw-anchored restoration.
- We evaluate DMC-5 across three long-horizon agent benchmarks. It reduces agent input by 38.5∼57.0% and end-to-end cost by 14.5∼28.0%, reaches 78∼85% compressor cache hit, and improves task performance by 1.0∼5.8% over No Compression baseline.

2 METHOD

We propose **DMC-5** (Dual-Model Collaboration with CaChe-Aware Context Compression) to reduce the end-to-end cost of long-horizon agents by separating full-history integration from action generation, and **YAMATO** (integritY-Aware Memory Abstractions for Tool Operations) to preserve execution-critical information during lossy compression. We first formalize the end-to-end cost, then present the dual-model workflow, the two trajectory views, and YAMATO.

2.1 PROBLEM SETUP AND END-TO-END COST

Given a task instruction q, let the interaction trajectory after t steps be

$$\tau_t = (\xi_1, \ldots, \xi_t), \qquad \xi_i = (a_i, o_i), \tag{1}$$

where a_i is the agent action and o_i is the resulting observation. A full-history agent F_A receives

$$X_t^{\text{full}} = s_A \oplus q \oplus \tau_t, \qquad a_{t+1} = F_A(X_t^{\text{full}}), \tag{2}$$

where s_A is the agent system prompt and $\oplus$ denotes serialization and concatenation. Because each decision conditions on all previous interactions, an interaction recorded at step i is repeatedly included in the inputs of steps $i, i+1, \ldots, T-1$. Therefore, the cumulative agent input is $\sum_{t=0}^{T-1} I(X_t^{\text{full}})$, which repeatedly charges for overlapping history as the trajectory grows.

The monetary cost of these overlapping tokens depends on prefix-cache pricing. For an endpoint m with input x and output y, let $I(x)$ and $O(y)$ denote the input and output token counts. We split the input into provider-reported cached tokens $H_m(x)$ and uncached tokens $I(x) - H_m(x)$. The billed cost of one call is

$$\mathcal{B}_m^{\text{disc}}(x, y) = (I(x) - H_m(x))p_m^{\text{miss}} + H_m(x)p_m^{\text{hit}} + O(y)p_m^{\text{out}}, \tag{3}$$

where p_m^{miss}, p_m^{hit}, and p_m^{out} are the uncached-input, cached-input, and output-token prices.

In practical settings, the agent is specialized for the downstream task through fine-tuning, while many current model providers do not expose discounted cached-input pricing for such customized endpoints (Alibaba Cloud, 2026). Therefore, we target deployments in which the specialized agent receives no cached-input discount, *i.e.*, $p_m^{\text{hit}} = p_m^{\text{miss}} = p_m^{\text{in}}$:

$$\mathcal{B}_m^{\text{flat}}(x, y) = I(x)p_m^{\text{in}} + O(y)p_m^{\text{out}}. \tag{4}$$

Thus, repeated history remains fully billed for the specialized agent, which motivates trajectory compression. For a generic trajectory compression method, let X_t^U denote the compressed input provided to the agent U at step t, and let X_r^V denote the input of compressor V at compression round $r \in \mathcal{R}$. Compared with the full-history agent, compression reduces the agent input but introduces an additional compressor cost. The end-to-end costs are

$$\mathcal{C}_{\text{full}} = \sum_{t=0}^{T_{\text{full}}-1} \mathcal{B}_U^{\text{flat}}(X_t^{\text{full}}, y_t^{\text{full}}), \qquad \mathcal{C}_{\text{cmp}} = \sum_{t=0}^{T_{\text{cmp}}-1} \mathcal{B}_U^{\text{flat}}(X_t^U, y_t^U) + \sum_{r \in \mathcal{R}} \mathcal{B}_V^{\text{disc}}(X_r^V, M_r), \tag{5}$$

where V and U denote the compressor and agent endpoints, M_r is the compressor output, and T_{full} and T_{cmp} are the total steps of agent run.

Compression is cost-effective only when the reduction in agent-side cost exceeds the added compressor cost and any interaction overhead caused by information loss. The compressor cost further depends on prefix-cache reuse: a larger $H_V(X_r^V)$ moves more input from the uncached price p_V^{miss} to the lower cached price p_V^{hit}. Meanwhile, compression may also change agent actions, thereby T_{cmp} may not equal T_{full}. We therefore measure end-to-end saving over the complete agent runs as

$$\Delta\mathcal{C} = \mathcal{C}_{\text{full}} - \mathcal{C}_{\text{cmp}}, \tag{6}$$

which accounts for agent-side token reduction, compressor processing, prefix-cache reuse, and changes in interaction length, rather than estimating compression ratio alone.

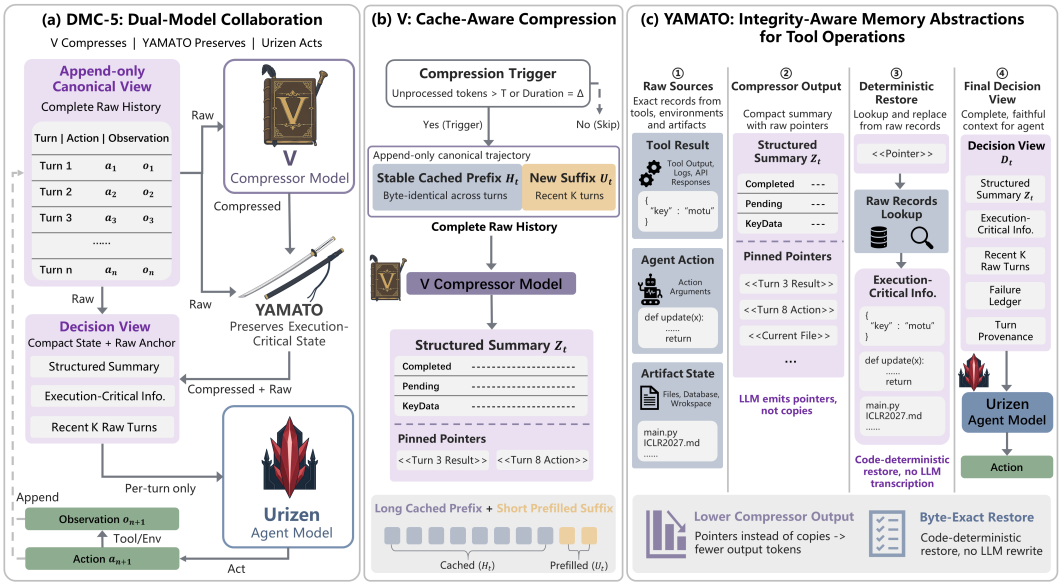

Figure 2: **Overview of DMC-5.** (a) A general compressor V integrates Append-only Canonical View, while a specialized agent Urizen acts on compact Decision View. (b) Each compression round reuses the stable historical prefix and only prefills new suffix. (c) YAMATO restores execution-critical records through pointers from raw records and combines them with recent raw turns.

2.2 DMC-5: Dual-Model Collaboration

Full-history integration and action generation impose different context requirements: the former benefits from complete historical access, while the latter should avoid repeatedly processing irrelevant history. Therefore, DMC-5 assigns full-history integration to a **general compressor** (denoted as **V**) and action generation to a **specialized agent** (denoted as **Urizen**), as shown in Fig. 2(a). Accordingly, the two models consume different views of the same trajectory: an **Append-only Canonical View** for V and a compact **Decision View** for Urizen.

The Append-only Canonical view stores the complete raw trajectory and is never replaced by compressed summary. First, at agent turn t, when compression round is triggered, DMC-5 selects the trajectory prefix $\tau_{t,K}^{pre} = (\xi_1, \ldots, \xi_{t-K})$ for compression, while retaining the most recent K turns $\tau_{t,K}^{suf} = (\xi_{t-K+1}, \ldots, \xi_t)$ verbatim in the Decision View. At each compression round, the entire eligible prefix is read directly from the canonical view and sent to the V compressor F_V:

$$X_t^V = s_V \oplus q \oplus \tau_{t,K}^{pre}, \qquad \widehat{M}_t = F_V(X_t^V), \tag{7}$$

where s_V is the compressor prompt. Thus, V constructs each compressed state $\widehat{M}_t$ directly from the raw trajectory, preserving access to earlier information while avoiding recursive summary drift.

Then, DMC-5 applies YAMATO operation $\mathcal{Y}$ to the output of V compressor to restore execution-critical state from the Append-only Canonical View:

$$M_t = \mathcal{Y}(\widehat{M}_t; \tau_{t,K}^{pre}) \oplus \tau_{t,K}^{suf}. \tag{8}$$

After that, the Decision View consists of agent prompt s_U, task instruction q and context M_t:

$$X_t^U = s_U \oplus q \oplus M_t, \qquad a_{t+1} = F_U(X_t^U) \tag{9}$$

where F_U is the Urizen agent. Thus, Urizen reads older history through M_r while retaining recent interactions verbatim. Subsequent actions a_{t+1} and observations o_{t+1} are appended only to canonical view as the suffix.

With this dual-model design, DMC-5 preserves full-history access for V while providing Urizen with only the compact state and recent raw interactions needed for action generation, reducing repeated agent-side context.

2.3 V: CACHE-AWARE CONTEXT COMPRESSION

Full-history compression preserves access to earlier information, but repeatedly processing the growing trajectory can shift the cost from the agent to the compressor. V addresses this cost through append-only prefix reuse, as shown in Fig. 2(b). Compression is triggered when the accumulated uncompressed history exceeds a token threshold T or the compression interval Δ is reached. At agent turn t, V's input is decomposed as

$$X_t^V = H_t \oplus U_t, \tag{10}$$

where H_t is the previously processed prefix and U_t contains newly eligible turns since the last compression. Because the canonical trajectory is append-only, H_t remains byte-identical and in the same order across compression events, allowing the endpoint to serve H_t at the cached-input price while only U_t is prefilled as new input. Therefore, V retains access to the full history without repeatedly paying the uncached-input cost for the same trajectory.

Once the compression is triggered, V outputs a structured summary containing completed and pending progress, key data, and pinned pointers to execution-critical raw records in canonical view. The output is passed to YAMATO and is never written back to the canonical trajectory. With the cache-aware context compression design, V retains full-history access without recursive summary, while keeping the historical prefix reusable across compression rounds.

2.4 YAMATO: INTEGRITY-AWARE MEMORY ABSTRACTIONS FOR TOOL OPERATIONS

Lossy compression reduces agent input, but corrupted or omitted tool operation state can cause incorrect or repeated actions. Tool operation often depends on exact identifiers, arguments, returned values, and artifact states. As shown in Fig. 2(c), YAMATO preserves such execution-critical information through deterministic restoration rather than model transcription, maintaining information integrity while reducing output tokens.

Specifically, YAMATO treats records whose values may affect subsequent action (*e.g.*, identifiers, arguments, returned values, and work-product states) as execution-critical, and lets compressor reference them through pointers. In particular, V outputs a structured summary Z_t and a set of pinned pointers Π_t:

$$\widehat{M}_t = (Z_t, \Pi_t). \tag{11}$$

YAMATO $\mathcal{Y}$ then restores the referenced canonical records deterministically, maintains persistent failure state, and attaches turn provenance to the semantic summary:

$$M_t = \mathcal{Y}(\widehat{M}_t; \tau_{t,K}^{pre}) \oplus \tau_{t,K}^{suf} = \overline{Z}_t \oplus \mathcal{R}(\Pi_t; \tau_{t,K}^{pre}) \oplus L_t \oplus \tau_{t,K}^{suf}, \tag{12}$$

where $\overline{Z}_t$ denotes the provenance-aware semantic summary, $\mathcal{R}$ denotes deterministic restoration operation, L_t retains failed actions across compression rounds. Detailed preservation rules, pointer schemas, and provenance handling are provided in Appendix 6.1.

In conclusion, YAMATO keeps the semantic state compact while restoring selected execution-critical information byte-for-byte from the canonical trajectory. It limits information loss that could otherwise increase interaction steps, while pointers reduce compressor output tokens and cost.

Overall, each component of DMC-5 corresponds directly to the end-to-end cost in Eq. 5. The Decision View reduces the agent input X^U, append-only compression increases cached reuse within the compressor input X^V, and YAMATO limits additional interaction steps T_{cmp} caused by information loss while reducing compressor output M through pinned pointers. DMC-5 therefore targets both terms of $\mathcal{C}_{\text{cmp}}$ rather than optimizing the agent-side compression ratio alone.

3 EXPERIMENTS

We evaluate four aspects of DMC-5: 1) end-to-end quality–cost trade-off, 2) cache-aware context compression, 3) asymmetric dual-model collaboration, 4) execution-critical information integrity.

3.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate DMC-5 on three long-horizon agent benchmarks. **AppWorld** (Trivedi et al., 2024) contains 168 executable multi-step application tasks, while **OfficeBench** (Wang et al., 2024b) contains 192 shorter office-oriented tool-use tasks; we report accuracy on both. **8-objective QA** contains 200 multi-objective tasks with substantially longer accumulated context, following (Kang et al., 2026) and report EM/F1. Together, they cover complex executable interaction,

Table 1: Overall comparison across three agent benchmarks. Score changes are absolute differences from No Compression; input and cost changes are relative. Agent Input is averaged per task, and Cache Hit reports compressor prefix-cache reuse rate.

Benchmark	Method	Metric	Score↑	Agent Input ↓		Compressor	Total Cost ↓	
				Tokens	Change	Cache Hit ↑	Total	Change
Agent Model: Fine-tuned Qwen3.5-27B			**Compressor Model:** DeepSeek-V4-flash					
AppWorld	No Compression	Acc.	77.0	258,996	–	–	13.71	–
	FIFO (Yang et al., 2024)	Acc.	36.3 (−40.7)	155,335	-40.02%	–	**8.98**	**-34.50%**
	AgentDiet (Xiao et al., 2026)	Acc.	64.8 (−12.2)	181,082	-30.08%	0.0	11.93	-12.98%
	ACON (Kang et al., 2026)	Acc.	67.3 (−9.7)	160,684	-37.96%	0.0	11.57	-15.61%
	DMC-5	Acc.	**78.2** (+1.2)	**138,431**	**-46.55%**	**80.3%**	10.31	-24.80%
OfficeBench	No Compression	Acc.	87.0	76,868	–	–	5.25	–
	FIFO (Yang et al., 2024)	Acc.	59.3 (−27.7)	53,281	-30.6%	–	**3.95**	**-24.80%**
	AgentDiet (Xiao et al., 2026)	Acc.	83.8 (−3.2)	55,328	-28.02%	0.0	4.13	-21.33%
	ACON (Kang et al., 2026)	Acc.	83.3 (−3.7)	49,447	-35.67%	0.0	4.19	-20.19%
	DMC-5	Acc.	**88.0** (+1.0)	**47,307**	**-38.46%**	**78.0%**	4.49	-14.48%
8-objective QA		EM	39.1					
	No Compression	F1	47.6	560,556	–	–	35.40	–
		EM	35.5 (−3.6)					
	AgentDiet (Xiao et al., 2026)	F1	43.6 (−4.0)	225,041	-59.85%	0.0	25.88	-26.89%
		EM	34.2 (−4.9)					
	ACON (Kang et al., 2026)	F1	41.5 (−6.1)	**224,119**	**-60.02%**	0.0	25.94	-26.72%
		EM	**44.9** (+5.8)					
	DMC-5	F1	**54.0** (+6.4)	240,937	-57.02%	**85.0%**	**25.50**	**-27.97%**

shorter workflows, and long-context information integration. Detailed task splits are provided in Appendix 6.2. All results are averaged over three independent runs, reported in Appendix 6.3.1

Models and baselines. For fair comparison, we use the same fine-tuned Qwen3.5-27B as the agent and DeepSeek-V4-flash as the compressor, comparing against NO COMPRESSION, FIFO (Yang et al., 2024), AGENTDIET (Xiao et al., 2026), and ACON (Kang et al., 2026). ACON is evaluated under official implementation, while AgentDiet is reproduced since no official code is available, reported in Appendix 6.2.4. Other model configurations are specified in the corresponding ablations.

Cost billing. We compute billed cost from observed uncached input, cached input, and output tokens using Eq.(5). All reported costs come from total tasks run and include both agent and compressor calls. Cache Hit denotes the provider-reported cached tokens. Details please refer to Appendix 6.2.2

Hyperparameters. We use $K = 1$, $\Delta = 3$, and $T = 6000$ by default. The compressor uses temperature 0 with thinking disabled, while the agent follows benchmark-default decoding. Please see Appendix 6.2.3 for more details.

3.2 COMPARISON WITH EXISTING COMPRESSION METHODS

Overall comparison. Table 1 compares DMC-5 with full-history prompting and three representative compressors across workloads of different lengths. AppWorld emphasizes multi-step executable interaction, OfficeBench contains shorter tool-use trajectories, and 8-objective QA has the largest accumulated input. Across all three settings, DMC-5 reduces agent input by 38.46∼57.02% and end-to-end cost by 14.48∼27.97%, while improving task performance by 1.0∼5.8% over No Compression. The cost reduction is smallest on the shorter OfficeBench, where compression has less opportunity to amortize its overhead, and largest on 8-objective QA with longer trajectories.

The baselines expose the trade-offs discussed in Section 1. FIFO achieves low cost by retaining only recent context, but loses 27.7∼40.7% accuracy on AppWorld and OfficeBench, and fails on 8-objective QA due to execution-critical information loss and non-terminating interaction loops. AgentDiet selectively removes locally redundant context, reducing agent input by 28.02∼59.85% and end-to-end cost by 12.98∼26.89%, while slightly outperforming ACON on OfficeBench and 8-objective QA. ACON achieves stronger agent-input reduction on AppWorld and OfficeBench through global compression, but its recursive summaries provide no reusable prefix cache and still reduce task performance by 3.7∼9.7%. Benefiting from the Append-only Canonical View, DMC-5 retains full-history access and 78.0∼85.0% cache-hit rate while restricting the agent to a compact Decision View, reaching 78.2% on AppWorld, 88.0% on OfficeBench, and 44.9/54.0% EM/F1 on 8-objective QA, showing better cost saving and task performance for long-horizon agent.

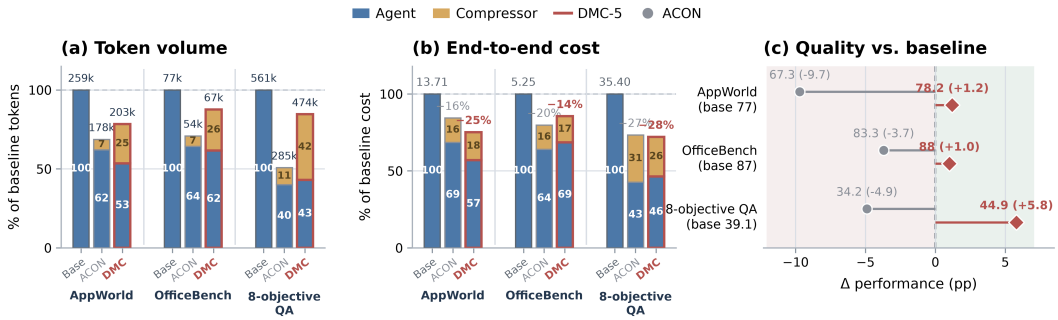

Figure 3: Comparison on token, cost and quality across three benchmarks. (a) Total token volume, decomposed into agent and compressor input. (b) Billed cost under the same decomposition. (c) Change in the task performance relative to No Compression.

Table 2: Comparison of representative recursive-summary ACON and append-only DMC-5.

Benchmark	Strategy	Cached Comp. Tokens	Uncached Comp. Tokens ↓	Comp. Cost ↓	End-to-End Cost ↓	Cost per Success ↓	Metric Score ↑
AppWorld	Recursive Summary (ACON)	0	17,269	**2.17**	11.57	0.1024	67.3
	Append-only (DMC-5)	**51,837**	**12,717**	2.50	**10.31**	**0.0781**	**78.2**
OfficeBench	Recursive Summary (ACON)	0	5,052	**0.83**	**4.19**	**0.0262**	83.3
	Append-only (DMC-5)	**15,635**	**4,410**	0.89	4.49	0.0266	**88.0**
8-objective QA	Recursive Summary (ACON)	0	60,441	10.84	25.94	0.0474	34.2
	Append-only (DMC-5)	**198,150**	**34,844**	**9.08**	**25.50**	**0.0355**	**44.9**

Note. Recursive Summary is implemented follows ACON under the same prompting and dual-model structure of DMC-5.

Token Volume vs. End-to-End Cost. Fig. 3 compares total token volume, end-to-end cost and task quality. Although DMC-5 processes more compressor input than ACON because its Canonical View retains full-history access, 78∼85% of this input is served from cache. As a result, the larger token volume does not translate into higher compressor cost, and DMC-5 achieves end-to-end cost comparable (+5.7%) to or even lower (-1.25∼-9.19%) than ACON across three benchmarks. At the same time, DMC-5's Decision View preserves more information for action generation and enhances task performance by 1.0∼5.8%, showing a better quality–cost trade-off by dual-model collaboration with cache-aware context compression.

3.3 Analysis on Cache-Aware Context Compression

Append-only vs. Recursive Summary. Table 2 shows that compared with ACON-style recursive summary, append-only DMC-5 configuration reuses 15.6K∼198.2K cached tokens per task and reduces uncached compressor input by 12.7∼42.3% across the three benchmarks, while improving task scores by 4.7∼10.9%. Despite processing more total compressor tokens, DMC-5's compressor cost remains comparable to recursive compression, with 23.7% and 25.1% lower cost per success on AppWorld and 8-objective QA, respectively. Fig. 4 further shows that the cache-hit rate of append-only input increases from 0 to 96% by the last compression, whereas recursive compression records no cache hits. Therefore, DMC-5 retains full-history access while leveraging low-cost prefix caching to handle repetitive history, achieving higher task performance and lower end-to-end cost.

Longer trajectories benefit more from DMC-5. Fig. 5 shows how the cost structure changes as an AppWorld trajectory grows. Without compression, per-step cost continues to increase with the accumulated history. After compression, DMC-5 reduces the agent-side cost and eventually drives it downward, indicating that the Decision View removes substantial redundant history, while the Append-only Canonical View keeps the added compressor cost growing slowly through prefix reuse. Table 3 shows the same trend that tasks compressed only once cost 10.7% more than No Compression, whereas tasks with at least two compressions save 16.6∼26.5% and improve accuracy by 3.7∼9.7%. As these per-step savings accumulate, Fig. 6 shows that DMC-5 reaches break-even around step 16 and achieves 25% net cost saving by step 50. Easy tasks realize savings earlier, while hard tasks gain more as the trajectory grows, reaching 30% cumulative saving. Overall, longer tasks trigger compression more often and provide more opportunities to reuse both the compact Decision View and the cached prefix of Canonical View, producing larger net savings.

3.4 Analysis on Asymmetric Dual-Model Collaboration

Table 4 evaluates whether history integration and action generation benefit from different model roles. Using the same compressor, replacing the general agent with the specialized fine-tuned

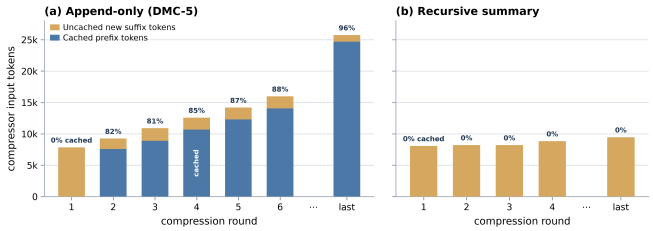

Figure 4: Compressor cache structure under append-only and recursive summary across compression rounds.

Table 3: Effect of compression frequency on OfficeBench.

Comp. Count	Task Count	Cost Change	Acc. Change
1	75	+10.7%	0
2	27	-26.5%	+3.7%
3–4	31	-20.5%	+9.7%
≥5	40	-16.6%	+7.5%

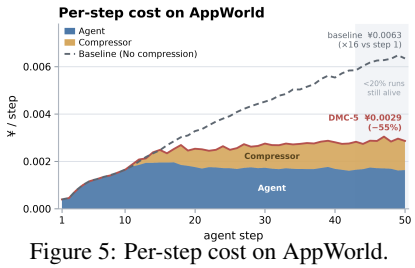

Figure 5: Per-step cost on AppWorld.

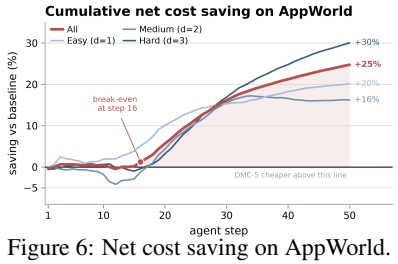

Figure 6: Net cost saving on AppWorld.

Table 4: Effect of model-role allocation on AppWorld.

Setting	Success ↑	Total Cost ↓	Cost / Success↓
Specialized model: Fine-tuned Qwen3.5-27B			
Specialized only	77.0%	13.71	0.1051
Specialized → Specialized	77.4%	21.49	0.1634
General model: DeepSeek-V4-flash			
General only	70.6%	**6.93**	**0.0591**
General → General	67.9%	9.07	0.0796
General → Specialized(DMC-5)	**78.2%**	10.31	0.0781
General model: Qwen3.7-flash			
General only	67.4%	**4.40**	**0.0389**
General → General	61.3%	7.81	0.0758
General → Specialized(DMC-5)	**73.8%**	9.50	0.0766

Note. $A \rightarrow B$ applies A as compressor and B as agent.

Qwen3.5-27B improves AppWorld success from 67.9% to 78.2% with DeepSeek-V4-flash, and from 61.3% to 73.8% with Qwen3.7-flash. This shows that the specialized model remains important for downstream action generation even when it no longer processes the full trajectory.

At the same time, moving full-history integration to a general compressor reduces the cost of specialized calls. Using the specialized model itself for compression raises total cost to 21.49 with only 77.4% success, whereas DeepSeek-V4-flash→Specialized achieves 78.2% at 10.31. Qwen3.7-flash further lowers the cost to 9.50, but its lower 73.8% success shows that the compressor must still preserve an accurate Decision View. Overall, the results support the asymmetric design: general models provide cheaper full-history integration, while specialized agents preserve action quality.

3.5 Analysis on Execution-Critical Information Integrity

YAMATO preserves execution-critical information. Table 5 isolates YAMATO from history organization by applying the same integrity layer to both recursive ACON and append-only DMC-5. On ACON, YAMATO reduces compressions containing errors from 4.2% to 1.0% and corrupted execution-critical items from 1.23% to 0.35%, improving recall of execution-critical items by 23.8%. On DMC-5, the corresponding rates drop from 3.0% to 0.04% and from 0.33% to 0.01%, while recall of execution-critical items reaches 98.3%. The consistent reductions under various compression mechanisms show that deterministic restoration protects execution-sensitive state.

Fewer integrity errors lead to fewer repeated failures. YAMATO also reduces repeated failures and improves task completion. Adding YAMATO to DMC-5 raises task accuracy from 69.1% to 78.2%, while runs with at least five repeated failed actions fall from 1.6% to 0.2% and the maximum same-action repetition drops from 29 to 6. ACON shows the same trend after adding YAMATO, with success increasing from 67.3% to 72.0% and the maximum repetition chain decreasing from 39 to 12. For DMC-5, the resulting reduction in repeated interaction also lowers end-to-end cost from 13.31 to 10.31. These results show that YAMATO not only preserves compressed state, but also prevents integrity errors from propagating into repeated actions and additional interaction cost.

Besides, Appendix 6.3 provides extensive empirical analysis of DMC-5, covering robustness across runs and hyperparameters, YAMATO integrity and overhead, task difficulty, pricing and latency trade-offs, as well as representative success and failure trajectories.

Table 5: Analysis of YAMATO over quality and integrity analysis on AppWorld.

Metric	No Compression	ACON (Kang et al., 2026)	ACON w/ YAMATO	DMC-5 w/o YAMATO	DMC-5 w/ YAMATO
Overall outcome					
Task Accuracy ↑	77.0%	67.3%	72.0%	69.1%	**78.2%**
End-to-End Cost ↓	13.71	11.57	12.11	13.31	**10.31**
Cost / Success ↓	0.1051	0.1024	0.1001	0.1147	**0.0781**
Integrity of compression					
Compression with Errors ↓	–	4.2% (30/711)	1.0% (9/894)	3.0% (77/2581)	**0.04%** (1/2,460)
Corrupted Execution-Critical Items ↓	–	1.23% (48/3,893)	0.35% (16/4,607)	0.33% (82/24,771)	**0.01%** (3/22,551)
Recall of Execution-Critical Items ↑	–	63.3% (279/441)	87.1% (608/698)	77.7% (839/1079)	98.3% (1008/1025)
Behavior of repeated failed actions					
Runs with Repeat $\geq$5 ↓	1.2%	1.6%	0.6%	1.6%	**0.2%**
Max Same-Action Repeats ↓	9	39	12	29	**6**

Note. Corrupted items are execution-critical items in the Decision View whose values disagree with the corresponding historical records. *Recall* is the fraction of correct, action-relevant execution-critical items that remain available in the Decision View. *Repeated failures* are subsequent identical or highly similar actions following an explicit execution error.

4 Related Work

Agent context compression and memory. Prompt compression shortens an LLM's input by token selection or summaries. LLMLingua and LongLLMLingua use smaller models to identify useful content, while RECOMP compresses retrieved evidence (Jiang et al., 2023b; 2024; Xu et al., 2024). Agent trajectories require repeated compression because they grow after every action. MemGPT manages context as tiered memory, and HiAgent groups interactions by subgoal (Packer et al., 2023; Hu et al., 2025). ACON optimizes compression guidelines for histories and observations and distills them into smaller compressors; AgentDiet removes useless, repeated, or expired content; AdaCoM trains an external context manager; and Focus lets an agent trigger consolidation (Kang et al., 2026; Xiao et al., 2026; Yi et al., 2026; Verma, 2026). MemAgent streams long inputs through a fixed-size memory trained with reinforcement learning (Yu et al., 2026). These methods shorten the acting model's context by selecting, rewriting, or externally managing prior state. Instead, DMC-5 keeps the raw trajectory append-only for repeated full-history integration, preserving a stable cached prefix for the compressor while providing the specialized agent with a separate compact Decision View.

Cache-aware LLM inference. Serving systems reduce repeated-prefix cost by managing KV caches. KVFlow schedules cache retention for agent workflows, CacheBlend reuses non-prefix caches through selective recomputation, and ThinKV compresses reasoning-model caches with adaptive quantization and eviction (Pan et al., 2025b; Yao et al., 2025; Ramachandran et al., 2026). LRAgent supports cache sharing across LoRA-specialized agents by separating shared and adapter-dependent states (Jeon et al., 2026). CAPC is particularly relevant to our setting, as it shows that query-dependent prompt compression can disrupt prefix reuse and explicitly optimizes compression for cache compatibility (Song, 2026). However, CAPC focuses on cache-aware compression within individual prompts, whereas DMC-5 considers repeated trajectory compression across agent turns. By maintaining an append-only full-history view, DMC-5 enables trajectory-level history integration to continuously reuse existing prefix caches and benefit from serving-side cache optimizations.

Multi-model collaboration and agent efficiency. Model cascades and routers reduce cost by assigning requests to models with different capabilities. FrugalGPT learns API cascades, and RouteLLM selects between stronger and weaker models from preference data (Chen et al., 2024; Ong et al., 2025). DUET uses one model to produce a short reasoning signal that another turns into an answer (Chen et al., 2026). Research on agent evaluation also argues that quality and resource use should be measured together (Kapoor et al., 2025). Unlike routing or cascade methods that decide which model should answer a request, DMC-5 keeps both models active with fixed asymmetric roles. The general model integrates full history into a compact Decision View for specialized model to generate action, reducing end-to-end cost while keeping task performance of long-horizon agents.

5 Conclusion

We presented **DMC-5** (Dual-Model Collaboration with CaChe-Aware Context Compression), a dual-model framework for reducing end-to-end cost of long-horizon agents by assigning full-history integration to general compressor and action generation to specialized agent. DMC-5 uses Append-only Canonical View to preserve full-history access and prefix-cache reuse, compact Decision View to reduce repeated agent input, and **YAMATO** (integritY-Aware Memory Abstractions for Tool Operations) to protect execution-critical information during lossy compression. Across AppWorld, OfficeBench, and 8-objective QA, DMC-5 reduces agent input by 38.5$\sim$57.0% and end-to-end cost by 14.5$\sim$28.0%, while improving task performance by 1.0$\sim$5.8% over uncompressed baseline.

AI USE STATEMENT

Generative AI tools were used to assist with language polishing, code development, and visualization. The research ideas, methodology, experimental design, and scientific claims were developed by the human authors, and all cited references were manually read and selected by the authors.

AI-assisted code was written under human direction and subsequently reviewed and verified by the authors. Experimental results were collected and aggregated using deterministic scripts rather than generative AI. Figures were generated from verified experimental data using plotting scripts partially assisted by AI tools, and all reported values and visualizations were manually checked.

All AI-assisted text and code were reviewed by the authors, who take full responsibility for the final content of this work.

ETHICS STATEMENT

This work uses established research benchmarks and does not involve human subjects, private or sensitive data, or other specific ethical concerns.

REPRODUCIBILITY STATEMENT.

We provide the implementation of DMC-5 and executable evaluation script for AppWorld in the supplementary materials submitted through OpenReview. Detailed benchmark settings, model configurations, prompts, compression hyperparameters, pricing coefficients, and evaluation protocols are documented in Appendix 6.2. These materials are intended to enable reproduction of the main experimental results.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. AutoMix: Automatically mixing language models. In *NeurIPS*, pp. 131000–131034, 2024.

Alibaba Cloud. Context cache. `https://help.aliyun.com/en/model-studio/context-cache`, 2026. Accessed 2026-09-17.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *TMLR*, pp. 1–10, 2024.

Yilei Chen, Sharut Gupta, Yannis Paschalidis, Ayush Sekhari, and Aldo Pacchiano. When less is enough: Efficient inference via collaborative reasoning. *arXiv:2605.01111*, pp. 1–10, 2026.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *EMNLP*, pp. 3829–3846, 2023.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed H. Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *ICLR*, pp. 41348–41366, 2024.

Tao Ge, Hu Jing, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *ICLR*, pp. 2591–2607, 2024.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *MLSys*, pp. 325–338, 2024.

Google Cloud. Context caching for fine-tuned Gemini models. `https://docs.cloud.google.com/gemini-enterprise-agent-platform/models/context-cache/context-cache-for-tuned-gemini`, 2026. Accessed 2026-09-17.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. HiAgent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *ACL*, pp. 32779–32798, 2025.

Hyesung Jeon, Hyeongju Ha, and Jae-Joon Kim. LRAgent: Efficient KV cache sharing for multi-LoRA LLM agents. In *ICML*, pp. 1–10, 2026.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In *ACL*, pp. 14165–14178, 2023a.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *EMNLP*, pp. 13358–13376, 2023b.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *ACL*, pp. 1658–1677, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *ICLR*, pp. 54107–54157, 2024.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing context compression for long-horizon LLM agents. In *ICML*, pp. 1–10, 2026.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *TMLR*, pp. 1–10, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *SOSP*, pp. 611–626, 2023.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation. In *ICLR*, pp. 54738–54778, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *ICML*, pp. 19274–19286, 2023.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In *EMNLP*, pp. 6342–6353, 2023.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *NeurIPS*, pp. 22947–22970, 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *ICLR*, pp. 52989–53046, 2024a.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. CacheGen: KV cache compression and streaming for fast large language model serving. In *ACM SIGCOMM*, pp. 38–56, 2024b.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *NeurIPS*, pp. 52342–52364, 2023.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *NeurIPS*, pp. 74325–74362, 2024.

Microsoft Azure. Deploy a fine-tuned model for inferencing. https://learn.microsoft.com/en-us/azure/foundry/openai/how-to/fine-tuning-deploy, 2026. Accessed 2026-09-17.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *NeurIPS*, pp. 19327–19352, 2023.

Yansong Ning, Jun Fang, Naiqiang Tan, and Hao Liu. Agent-Omit: Adaptive context omission for efficient LLM agents. In *ICML*, pp. 1–10, 2026.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *ICLR*, pp. 34433–34448, 2025.

OpenAI. Prompt caching in the API. https://openai.com/index/api-prompt-caching/, 2026. Accessed 2026-09-17.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv:2310.08560*, pp. 1–10, 2023.

Zaifeng Pan, Yitong Ding, Yue Guan, Zheng Wang, Zhongkai Yu, Xulong Tang, Yida Wang, and Yufei Ding. FastTree: Optimizing attention kernel and runtime for tree-structured LLM inference. In *MLSys*, pp. 1–10, 2025a.

Zaifeng Pan, Ajjkumar Dahyalal Patel, Yipeng Shen, Zhengding Hu, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. KVFlow: Efficient prefix caching for accelerating LLM-based multi-agent workflows. In *NeurIPS*, pp. 139912–139931, 2025b.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of ACL*, pp. 963–981, 2024.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *UIST*, pp. 1–22, 2023.

Guanghui Qin, Corby Rosset, Ethan C. Chau, Nikhil Rao, and Benjamin Van Durme. Dodo: Dynamic contextual compression for decoder-only LMs. In *ACL*, pp. 9961–9975, 2024.

Akshat Ramachandran, Marina Neseem, Charbel Sakr, Rangharajan Venkatesan, Brucek Khailany, and Tushar Krishna. ThinKV: Thought-adaptive KV cache compression for efficient reasoning models. In *ICLR*, pp. 110072–110106, 2026.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *ICLR*, pp. 32628–32649, 2024.

Yaorui Shi, Yuxin Chen, Siyuan Wang, Sihang Li, Hengxing Cai, Qi Gu, Xiang Wang, and An Zhang. Look back to reason forward: Revisitable memory for long-context LLM agents. In *ICLR*, pp. 78055–78082, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, pp. 8634–8652, 2023.

Yan Song. Cache-aware prompt compression: A two-tier cost model for LLM API caching. *arXiv:2607.15516*, pp. 1–10, 2026.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *ICML*, pp. 47901–47911, 2024.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *ACL*, pp. 16022–16076, 2024.

Nikhil Verma. Active context compression: Autonomous memory management in LLM agents. *arXiv:2601.07190*, pp. 1–10, 2026.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Y. Zou. Mixture-of-agents enhances large language model capabilities. In *ICLR*, pp. 33944–33963, 2025.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. MEMORYLLM: Towards self-updatable large language models. In *ICML*, pp. 50453–50466, 2024a.

Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. OfficeBench: Benchmarking language agents across multiple applications for office automation. *arXiv:2407.19056*, pp. 1–10, 2024b.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. ReSum: Unlocking long-horizon search intelligence via context summarization. *arXiv:2509.13313*, pp. 1–10, 2025.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proc. ACM Softw. Eng.*, 2:801–824, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *ICLR*, pp. 21875–21895, 2024.

Yuan-An Xiao, Pengfei Gao, Chao Peng, and Yingfei Xiong. Reducing cost of LLM agents with trajectory reduction. *Proc. ACM Softw. Eng.*, 3:1241–1263, 2026.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. RECOMP: Improving retrieval-augmented LMs with context compression and selective augmentation. In *ICLR*, pp. 43478–43502, 2024.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *NeurIPS*, pp. 50528–50652, 2024.

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 *EuroSys*, pp. 94–109, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, pp. 1–10, 2023.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Wenbiao Yin, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Jingren Zhou, Siheng Chen, and Yong Jiang. AgentFold: Long-horizon web agents with proactive context folding. In *ICLR*, pp. 154754–154776, 2026.

Lu Yi, Runlin Lei, Liuyi Yao, Yuexiang Xie, Yuyang Li, Wenhao Zhang, Zhewei Wei, Yaliang Li, and Jian-Yun Nie. Learning agent-compatible context management for long-horizon tasks. *arXiv:2605.30785*, pp. 1–10, 2026.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. MemAgent: Reshaping long-context LLM with multi-conv RL-based memory agent. In *ICLR*, pp. 39458–39486, 2026.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *NeurIPS*, pp. 34661–34710, 2023.

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 *NeurIPS*, pp. 62557–62583, 2024.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *ICLR*, pp. 15585–15606, 2024.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *ICLR*, pp. 58413–58438, 2026.

6 APPENDIX

This appendix specifies the evaluation protocol, implementation, and accounting used in DMC-5. It also reports uncertainty estimates, mechanism-level ablations, efficiency analyses, and representative trajectory cases.

6.1 YAMATO IMPLEMENTATION DETAILS

YAMATO augments lossy semantic compression with execution-critical information that should not be reconstructed from model-generated text. It introduces no additional LLM calls: all model-generated fields are produced within the existing compressor call, while pointer restoration, work-product retention, failure tracking, and provenance checks are implemented deterministically in code.

At compression round r, the compressor V reads the task instruction and the eligible prefix of the Append-only Canonical View, and outputs

$$\widehat{M}_r = (Z_r, \Pi_r), \tag{13}$$

where Z_r is a structured semantic summary and Π_r contains references to canonical records that should be restored exactly. YAMATO organizes the compressed state through three complementary mechanisms: (i) a structured summary with turn provenance, (ii) exact restoration of selected canonical records, and (iii) a persistent failure ledger. The resulting compact state is then combined with recent uncompressed turns to form the Decision View.

6.1.1 CANONICAL RECORDS AND PRESERVATION CANDIDATES

Each canonical turn $\xi_i = (a_i, o_i)$ contains the agent action a_i and its observation o_i, both of which remain unchanged in the Append-only Canonical View. YAMATO maintains a turn registry that maps each turn index i to its raw action and observation strings.

Specifically, each new action–observation turn is serialized once when it is appended to the Canonical View. Deterministic rules add a type marker to the turn header: `(API DOC)` and `(FILE CONTENT)` denote literal tool outputs, `(WORK PRODUCT)` denotes artifact-producing or artifact-updating actions, and `(FAILED)` denotes unsuccessful operations. Once appended, earlier turns are neither regenerated nor relabeled, preserving the byte-identical historical prefix required for cache reuse.

For example, an API-documentation turn may be serialized as:

```
--- Turn 7 (API DOC) ---
[ASSISTANT ACTION]
get_api_documentation("calendar.create_event")

[TOOL RESULT]
create_event(calendar_id, title, start_time)
```

These markers guide the compressor in identifying candidate records for exact preservation. When a marked record is needed in the Decision View, the compressor can reference it with a turn pointer instead of copying the full raw content into the semantic summary. For example, it may emit `<<TURN_7_RESULT>>`, which YAMATO resolves deterministically to the original observation stored in the Canonical View.

6.1.2 STRUCTURED SUMMARY WITH TURN PROVENANCE

The semantic summary adopts an ACON-style (Kang et al., 2026) structured history format, with its fields adapted to support DMC-5's provenance tracking, exact-state restoration, and persistent failure state. Together with the Append-only Canonical View, it forms the semantic component of DMC-5's state representation.

In particular, the semantic component Z_r compresses the task state into five fixed fields: REASONING, COMPLETED, PENDING, FAILED ATTEMPTS, and KEY DATA. They respectively record the

current task state, completed progress, remaining steps, previous unsuccessful attempts, and compact information that may be needed by later actions. A simplified format is shown below:

```
### REASONING
- ...

### COMPLETED
- ...

### PENDING
- ...

### FAILED ATTEMPTS
- ...

### KEY DATA
- ...
```

To distinguish environment-grounded information from agent-side derivations, the compressor attaches turn-level provenance tags to summarized claims. Table 6 summarizes the tags.

Table 6: Turn-provenance tags used in the structured YAMATO summary.

Tag	Source	Meaning
`[O@i]`	Observation o_i	The summarized information is grounded in a tool or environment observation at turn i.
`[D@i]`	Agent-side state at turn i	The information is an agent-derived plan, inference, or intermediate conclusion associated with turn i, rather than a directly observed result.
`[UNSOURCED]`	No explicit turn reference	A KEY DATA entry has no recognized source reference in the compressed summary.

The `[O@i]` and `[D@i]` tags are generated as part of the same compressor output. The compressor is instructed to preserve the distinction between information observed from the environment and information derived by the agent when rewriting historical state.

For KEY DATA, YAMATO additionally applies a lightweight deterministic format check. If an entry contains no recognized turn-level source reference, the code marks it as `[UNSOURCED]`. This check does not determine whether the claim is factually correct and does not modify an existing `[O@i]` or `[D@i]` tag. Its purpose is only to make missing provenance explicit to the acting agent.

We denote the resulting provenance-aware semantic summary as

$$\overline{Z}_r = \mathrm{Prov}(Z_r), \tag{14}$$

where $\mathrm{Prov}(\cdot)$ denotes the deterministic provenance-format check described above.

Turn provenance complements exact raw-record restoration: $\overline{Z}_r$ preserves compact semantic state together with its historical source, while pinned pointers are used when execution-critical content should be copied directly from the Canonical View.

6.1.3 PINNED RECORDS AND WORK-PRODUCT RESTORATION

YAMATO protects three types of execution-critical information: *tool observations*, *agent actions*, and *work products*. For these records, the compressor emits short pointers to the corresponding canonical turns rather than reproducing their contents in the semantic summary. YAMATO then resolves the pointers deterministically.

Let

$$\Pi_r = \Pi_r^{\mathrm{act}} \cup \Pi_r^{\mathrm{obs}} \cup \Pi_r^{\mathrm{work}} \tag{15}$$

Table 7: YAMATO pointer interfaces for restoring execution-critical state from the Append-only Canonical View.

Pointer	Canonical target	Restored content	Handling
`<<TURN_i_RESULT>>`	Observation o_i	Exact tool or environment output, such as API responses, logs, documentation, returned text, or file contents.	Direct lookup and restoration.
`<<TURN_i_ACTION>>`	Action a_i	Exact tool call, command, arguments, or write/edit operation.	Direct lookup and restoration.
`<<TURN_i_WORK>>`	Work-product action at turn i	Exact artifact-related operation associated with a file.	Lookup followed by code-side matching and ordered aggregation of operations on the same file.

denote the action, observation, and work-product pointers emitted at compression round r. Table 7 summarizes their semantics.

For action and observation pointers, YAMATO directly retrieves the referenced raw records:

$$E_r^{\text{act}} = \mathcal{R}\left(\Pi_r^{\text{act}}; \tau_{b_r}\right), \tag{16}$$

$$E_r^{\text{obs}} = \mathcal{R}\left(\Pi_r^{\text{obs}}; \tau_{b_r}\right), \tag{17}$$

where $\mathcal{R}$ denotes deterministic lookup from the Canonical View and b_r is the last canonical turn included in compression round r. The referenced action or observation is copied verbatim into the Decision View.

Work-product pointers use the same lookup mechanism, but require one additional step because an artifact may evolve across multiple turns. YAMATO first resolves the referenced work-product records:

$$\widetilde{E}_r^{\text{work}} = \mathcal{R}\left(\Pi_r^{\text{work}}; \tau_{b_r}\right). \tag{18}$$

For each resolved work-product record, YAMATO identifies its artifact path and matches other work-product operations on the same file in the Canonical View. These operations are then concatenated in their original execution order:

$$E_r^{\text{work}} = \mathcal{G}\left(\widetilde{E}_r^{\text{work}}; \tau_{b_r}\right), \tag{19}$$

where $\mathcal{G}$ denotes deterministic path matching and ordering over the eligible Canonical View. This preserves the sequence of writes and edits that defines the current work product without asking the compressor to reconstruct the artifact from text.

The restored execution-critical context is therefore

$$E_r = E_r^{\text{act}} \oplus E_r^{\text{obs}} \oplus E_r^{\text{work}}. \tag{20}$$

6.1.4 Persistent Failure Ledger

Lossy compression may discard previous failed attempts, causing the agent to repeat the same ineffective action. YAMATO therefore maintains a persistent failure ledger L_r across compression rounds:

$$L_r = \text{Tail}_N\left(L_{r-1} \oplus L_r^{\text{new}}\right), \tag{21}$$

where L_r^{new} contains failures observed since the previous compression round and $\text{Tail}_N(\cdot)$ retains only the most recent N entries. We set $N = 40$ by default. The ledger serves as persistent context for avoiding repeated mistakes, but does not prevent the agent from retrying an operation when later context suggests a corrected action.

6.1.5 DECISION-VIEW CONSTRUCTION

After provenance processing, exact-record restoration, and failure-state update, YAMATO assembles the compact state at compression round r as

$$C_r = \overline{Z}_r \oplus E_r \oplus L_r, \tag{22}$$

where $\overline{Z}_r$ is the provenance-aware semantic summary, E_r contains the restored execution-critical records, and L_r is the persistent failure ledger.

Suppose compression round r is triggered at agent turn t_r. The most recent K turns are excluded from the compressed prefix, so

$$b_r = t_r - K, \tag{23}$$

where b_r is the last canonical turn integrated into the compressed state. Let $r(t)$ denote the latest completed compression round at agent turn t. The Decision View is then

$$M_t = C_{r(t)} \oplus \tau^{\text{raw}}_{b_{r(t)}+1:t}, \tag{24}$$

$$X^U_t = s_U \oplus q \oplus M_t, \tag{25}$$

where $\tau^{\text{raw}}_{b_{r(t)}+1:t}$ contains the recent raw turns that were excluded from compression together with any turns generated afterward.

Thus, C_r remains fixed between two compression rounds, while new actions and observations are appended unchanged to the raw suffix until the next refresh. The resulting Decision View combines four complementary information sources: (i) a compact semantic summary for task progress and long-range state, (ii) exact restored records for execution-critical information, (iii) a persistent ledger of previous failures, and (iv) recent raw interactions. YAMATO therefore does not make the complete trajectory lossless; instead, it uses semantic compression for ordinary history while protecting information whose corruption or omission may affect subsequent tool execution.

Illustrative example. Listing 1 illustrates the Decision View for an OfficeBench task. Assume that compression round r is triggered after turn 9 with $K = 1$. The compressed prefix therefore covers turns 1–8 ($b_r = 8$), while turn 9 remains in the recent raw suffix. Lines beginning with `%%` are annotations for illustration and are not visible to the agent.

Listing 1: Illustrative Decision View for an OfficeBench task.

```
%% C_r = Zbar_r + E_r + L_r

<HISTORY_SUMMARY>

### COMPLETED
- Located the Q2 revenue value [O@2]
- Created Q2_Sales_Report.docx [O@8]

### PENDING
- Export the report document to PDF [D@8]

### FAILED ATTEMPTS
- T4: write_file failed because "path" was missing

### KEY DATA
- net revenue: 182400.00 [O@2]
- report path:
  /workspace/Q2_Sales_Report.docx [O@8]

### PINNED RECORD
%% Restored from <<TURN_2_RESULT>>

File: quarterly_sales.xlsx
Sheet: Q2
A6: Net Revenue
B6: 182400.00

### WORK PRODUCT
%% Restored from the canonical work-product history

{"action":"write_file",
 "path":"/workspace/Q2_Sales_Report.docx",
 "content":"Q2 Sales Report\nNet Revenue: 182400.00"}
```

```
</HISTORY_SUMMARY>


%% tau_raw_{9:9}

[ASSISTANT ACTION, T9]
{"action":"convert_to_pdf",
 "input_path":"/workspace/Q2_Sales_Report.docx",
 "output_path":"/workspace/Q2_Sales_Report.pdf"}

[TOOL RESULT, T9]
Conversion completed:
/workspace/Q2_Sales_Report.pdf
```

The upper block corresponds to C_r. Its semantic fields come from $\overline{Z}_r$, the spreadsheet record and work-product state are restored through E_r, and the failed `write_file` attempt is retained in L_r. For example, `[O@2]` marks information grounded in the tool observation at turn 2, whereas `[D@8]` denotes an agent-derived pending decision.

Turn 9 is excluded from the compressed prefix as the recent raw turn and therefore belongs to $\tau_{9:9}^{\text{raw}}$ rather than C_r. The agent receives both parts in the same Decision View. As further turns are generated, they are appended to this raw suffix until the next compression round constructs a new compact state.

6.2 Evaluation Details

6.2.1 Benchmark and task configurations.

Table 8 summarizes our evaluation sets. We evaluate AppWorld on all 168 tasks in `test_normal_split`, OfficeBench on 192 subtask instances from the combined training and testing partitions, and 8-objective QA on all 200 tasks from its training and test partitions. DMC-5 requires no model parameter updates. We tune only the compression prompt on the AppWorld training split, then freeze it and apply the same prompt unchanged to all three benchmarks. Since no training is performed on OfficeBench or 8-objective QA, we combine their available training and test splits to maximize evaluation coverage. For a controlled comparison, all methods use the same task sets and a uniform budget of at most 50 agent interaction steps per task. All reported results are averaged over three independent runs.

Table 8: Benchmark overview.

Benchmark	Evaluated split	#Tasks	Task grouping	Max steps	Metric
AppWorld	test_normal_split	168	57 / 48 / 63 Easy / Medium / Hard	50	Accuracy
OfficeBench	Training + testing	192	86 / 51 / 55 1-app / 2-app / 3-app	50	Accuracy
8-objective QA	Train + test	200	–	50	EM / F1

AppWorld (Trivedi et al., 2024) provides a stateful execution environment spanning nine everyday applications, including Spotify, Gmail, and SimpleNote, with 457 APIs and approximately 100 simulated users (Kang et al., 2026). An agent completes user requests through executable actions that retrieve information and modify application state, often requiring coordination across applications. Our evaluation uses 168 `test_normal_split` tasks: 57 easy, 48 medium, and 63 hard. Correctness is measured using the benchmark-defined task goal completion score, reported as accuracy (%). This benchmark tests whether compressed context preserves the information and execution state needed to complete multi-step application workflows.

OfficeBench (Wang et al., 2024b) evaluates office automation workflows involving applications such as Word, Excel, PDF, Calendar, and Email, together with utilities such as Shell and Calculator. Tasks are grouped into 86 1-app, 51 2-app, and 55 3-app categories according to the number of applications involved. We evaluate 192 subtask instances from the combined training and testing

partitions used in our implementation. Correctness is assessed using the benchmark-provided accuracy functions, which evaluate the outcomes of the requested office operations. We report accuracy (%) aggregated over the evaluated instances. These workflows test whether the agent retains relevant document content, intermediate results, and task progress across successive tool interactions.

8-objective QA (Zhou et al., 2026; Kang et al., 2026) groups eight distinct questions from NaturalQuestions into a single task. The agent interacts with a search tool and must return a consolidated set of answers to all eight questions. Unlike single-question multi-hop QA, this setting requires tracking multiple question–answer pairs and unresolved objectives within the same interaction history. We evaluate all 100 training tasks and 100 test tasks, giving 200 composite tasks and 1,600 question instances. Long context accumulates through successive search queries, retrieved evidence, and intermediate reasoning, while answers obtained earlier must remain available for the final response. Following Kang et al. (2026), we report exact match (EM) and F1 separately, averaging each metric over the eight questions in a task and then over all evaluated tasks.

Table 9: Representative tasks from the evaluated benchmarks.

Benchmark	Category	Example Task
AppWorld	Easy	Remove all the songs that I have liked from my Spotify queue, and then start the player.
	Medium	Reset friends on venmo to be the same as my friends in my phone. Befriend and unfriend as needed.
	Hard	My roommates and I went for a dinner yesterday. I paid for the taxi back and forth (total $60) and Nancy paid for everyone's food (total $128). Both food and commute are supposed to be shared equally among all. Make necessary payment requests with a note "For Taxi", and a payment to Nancy with a note "For Food", on venmo.
OfficeBench	1-app	Add a meeting to Bob's calendar at 5/17/2024 10:30 a.m to 11:00 a.m.
	2-app	Schedule a team training session for all participants from excel file, create calendar events for each member in their name.
	3-app	Convert Excel spreadsheet containing project tasks to task.pdf and email Alice with contents from the pdf file for review.
8-objective QA	Multi-question	What category was hurricane charley when it hit florida?; where in the bible is the book of revelation?; who played skeletor in the movie masters of the universe?; who has the most votes for the all star game?; when was the first mad max movie release?; the cast of don't tell mom the babysitter's dead?; what does the adrenal gland produce that is necessary for the sympathetic nervous system to function?; who played ice queen in chronicles of narnia?

6.2.2 Models, Endpoints, and Pricing

All models are accessed through API endpoints on Alibaba Cloud Model Studio (Bailian).[1] The specialized agent is based on Qwen3.5-27B and deployed as a LoRA-adapted endpoint after fine-tuning on private agent data. We verified that no evaluation instances from the three benchmarks are included in the fine-tuning data. DeepSeek-V4-flash serves as the default general compressor, while Qwen3.7-flash is used in the model-role ablation. Unless otherwise stated, the same specialized agent is used across all compression methods.

Table 10 summarizes the pricing used for end-to-end cost accounting. We use the Alibaba Cloud Model Studio (Bailian) pricing in effect at the time of evaluation. Model prices may change over time, and all reported costs are computed using this fixed pricing snapshot.

[1] https://www.alibabacloud.com/en

Table 10: Model roles and pricing used for end-to-end cost accounting. Prices are reported per one million tokens and correspond to the pricing in effect at the time of evaluation.

Role	Model	Uncached input	Cached input	Output	Deployment
Specialized agent	Qwen3.5-27B (LoRA)	0.30	0.30	2.40	Fine-tuned endpoint
Primary compressor	DeepSeek-V4-flash	0.50	0.10	1.00	API endpoint
Alternative compressor	Qwen3.7-flash	0.10	0.02	0.40	API endpoint

6.2.3 Inference Configuration, Prompts, and Compression Settings

Inference configuration. All compressor calls use greedy decoding with temperature 0 to minimize stochastic variation in the compressed state. We disable the model's internal thinking mode (`enable_thinking=false`), since trajectory compression is formulated as structured information extraction and state preservation rather than open-ended reasoning. For the specialized agent, we retain the decoding configuration recommended by the deployed API endpoint, including its default temperature and benchmark-consistent thinking setting. We do not fix generation seeds. Instead, all reported results are obtained from three independent runs to capture the stochasticity of API-based agent execution.

Hyperparameters. Unless otherwise specified, DMC-5 uses the same compression configuration across all three benchmarks. The most recent $K = 1$ interaction turn is always retained verbatim in the Decision View. A new compression round is triggered when either (i) three new interaction turns have accumulated since the previous compression, or (ii) the newly accumulated uncompressed trajectory exceeds $6{,}000$ tokens. Thus, with compression interval $\Delta = 3$ and token threshold $T = 6{,}000$, the trigger condition is

$$\mathrm{Compress}(t) = \mathbb{I}\left[\Delta_{\mathrm{turn}}(t) \geq 3 \ \vee \ \Delta_{\mathrm{token}}(t) \geq 6{,}000\right]. \tag{26}$$

At each compression round, the compressor reads the complete eligible prefix from the Append-only Canonical View, while the most recent turn remains uncompressed.

Table 11: Default hyperparameters.

Setting	Value
Compressor temperature	0 (greedy)
Compressor thinking	Disabled
Recent raw turns K	1
Compression interval Δ	3 turns
Token trigger T	6,000 new tokens
Agent decoding	Endpoint / benchmark default

Prompts and input construction. The specialized agent retains the original benchmark-specific agent prompt without modification. Across benchmarks, DMC-5 follows the same compressor-prompt design and input organization, while only the task- and tool-specific content varies. The compressor summarizes task progress and pending work, preserves execution-critical state and provenance, and emits raw-record pointers for information that should remain byte-exact. A representative compressor prompt is shown in Listing 2.

Listing 2: An example compressor prompt used for trajectory compression.

```
You are a context compression agent for a multi-step, tool-using AI agent.

You will receive a task instruction followed by the agent's full interaction
trajectory: a sequence of turns, each containing the agent's action
([ASSISTANT ACTION], a tool call like `tool({...arguments...})`) and the
corresponding tool result / observation ([TOOL RESULT]). Turns marked with
(FILE CONTENT) contain raw source material in their tool result: the exact
contents of a file read from disk, or the readable text of a fetched web
page. Turns marked with (WORK PRODUCT) contain file contents the agent
```

```
wrote or edited itself (e.g. an analysis script) -- the artifact the next
steps keep modifying. Turns marked with (FAILED) produced an error or an
empty result.

Produce a structured summary of the trajectory that preserves all critical
decision-making context for the agent's next step. You must NEVER add
information that was not present in the original trajectory.

PROVENANCE RULES -- you are a scribe, not a judge. The trust level of any
information may only be preserved or downgraded by summarisation, never
upgraded:
1. Tag statements with their source class (shorthand):
   - [O@i] -- observed directly from the tool result at turn i.
   - [D@i] -- a plan, rule, or conclusion the AGENT itself formed at turn i;
     UNVERIFIED, may be wrong; never present it as established fact.
   You yourself must NOT introduce any new conclusion.
2. NEVER restate task requirements in your own words -- write
   "per task instruction" instead of paraphrasing (paraphrase is how
   constraints get silently mutated).
3. Uncertainty must survive: keep hedges ("probably", "only page 0
   fetched"); never state anything more confidently than your source.

BREVITY RULES -- output tokens are expensive; every character must earn
its place:
4. Telegraphic note form. No full sentences, no articles, no pleasantries.
   e.g. "csv parsed, 250 rows, 2024 monthly [O@5]" not "I have successfully
   parsed the CSV file which contains 250 rows of monthly data."
5. Single home per value: exact values (totals, dates, names, URLs, IDs)
   live ONLY in KEY DATA. Other sections refer to them by name, never
   repeat them.
6. Do NOT list turns marked (FAILED) in FAILED ATTEMPTS -- they are
   recorded automatically by the system. Only list dead ends that are NOT
   marked (FAILED), e.g. a search that returned results which turned out
   irrelevant.

VERBATIM PRESERVATION RULES:
- Exact formats matter. Numbers, column names, dates, URLs, and quoted
  facts must NEVER be paraphrased or rounded. Copy them character-by-
  character, or use a placeholder.
- To preserve an entire tool result verbatim, do NOT copy its text.
  Instead output a placeholder on its own line: <<TURN_i_RESULT>>
  (i = turn number). It is auto-replaced with the original text --
  cheaper and always exact.
- In PINNED CONTENT, pin (via placeholder) every (FILE CONTENT) turn whose
  material is still needed for the remaining steps in PENDING -- especially
  files whose values must be cited in the final answer/report, and fetched
  pages whose facts have NOT yet been extracted into KEY DATA. Do NOT pin
  files already fully processed, nor pages whose relevant facts are fully
  captured in KEY DATA -- pinning processed sources only bloats the view.
- If the trajectory ends with a "=== CODE-CONSUMED SOURCES ===" note, the
  listed files have already been processed by code (a later script reads
  them from disk; subsequent steps work from script output, never from the
  raw text). Do NOT pin those turns -- their value now lives in the script
  outputs and in KEY DATA.
- (WORK PRODUCT) turns: the LATEST one is auto-pinned by the system -- do
  NOT pin it again. An OLDER work product still needed (e.g. a script that
  will be edited again while a newer one is also pinned) may be pinned via
  <<TURN_i_ACTION>> on its own line.
- When the pinned work product is an edit chain (a write plus subsequent
  edits to the same file), the system pins the WHOLE chain -- base write
  plus patches in order -- and appends it after your summary. That chain IS
  the current file state: never list "re-read the script/file" as a
  PENDING step and never claim the file content was lost.
- This is critical for FAILED shell/python turns: paraphrasing a failed
  command ("openpyxl import failed") does NOT tell the agent WHICH package
  or flag broke -- the agent will retry near-identical variants and burn
  steps. The exact previous command is the only reliable signal.

Respond in EXACTLY the following format, starting with the line
"# History Summary":

# History Summary

### REASONING
Max 3 telegraphic lines: where the task stands, with provenance tags.

### COMPLETED
One terse line per completed subtask, tagged [O@i]. No values (rule 5).

### PENDING
Remaining steps + known error info, terse. Plans/decision rules tagged
```

```
[D@i], stating what they were based on (e.g. "based on agent's reading of
task instruction at T26") -- state provenance only, do NOT add
verification directives or commands.
For EVERY enumeration over a collection, record completeness state:
- "[O@4] web search: first page only, NOT confirmed exhaustive"
- "[O@7] csv rows: read to end of file, COMPLETE"
Never present a partial collection as complete.

### FAILED ATTEMPTS
Only dead ends NOT marked (FAILED) (see rule 6), one terse line each:
tried what -> outcome -> RULED OUT / retry-with-what-change.
Never drop a previously valid entry unless the same action later
succeeded. If none, write "None".

### KEY DATA
ONLY [O@i] values: figures, dates, names, URLs, IDs, lists -- exact
copies, each with source tag. Derived datasets (e.g. "rows to exclude")
go to PENDING as [D@i], not here.

### PINNED CONTENT
One placeholder per line (see pinning rules), e.g.:
<<TURN_3_RESULT>>
If none needed, write "None".

Do NOT repeat the original trajectory. Return ONLY the formatted summary.
```

Listing 3 illustrates an example of the input to the specialized agent at decision step t. The same input organization is used across benchmarks. The system prompt and original task remain fixed throughout an episode, while the HISTORY_SUMMARY block is updated only when compression is triggered. Its semantic fields are generated by the compressor, whereas the regions annotated as YAMATO-RESTORE correspond to execution-critical records restored verbatim from the Append-only Canonical View. These annotations are included only for exposition; at runtime, the restored raw content is inserted directly without such markers. The most recent K interaction turns bypass compression and are retained verbatim, while the current raw observation and benchmark-specific action or tool schema are appended for the next decision.

Listing 3: Input to the specialized agent in OfficeBench at decision step t. The notation is explanatory and is not included in the runtime prompt.

```
messages = [

  # [SESSION-FIXED]
  # Instantiated once for the current OfficeBench episode.
  {
    "role": "system",
    "content":
      "You are an AI assistant operating an office automation
       environment. Respond with a single JSON action per step.
       User: <username>
       Date and time: <date, weekday, time>
       Available applications: <application descriptions>
       Workspace: <testbed path>
       ..."
  },

  # [TASK-FIXED + REFRESH-r]
  # The original task remains unchanged. The memory block is replaced
  # whenever compression refresh r completes.
  {
    "role": "user",
    "content":
      "Task: <original OfficeBench instruction>
       Available apps: <initial application list>
       ...

       <HISTORY_SUMMARY>
       [FIXED INTERPRETATION TEMPLATE]
       [O@n] = observation from tool result at turn n;
       [D@n] = unverified agent derivation at turn n;
       [UNSOURCED] = no source recorded.
       This memory records earlier interactions but does not override
       the original task.

       [REFRESH-r: COMPRESSOR-GENERATED SEMANTIC STATE]
       # History Summary

       ### REASONING
```

```
        <current progress and decision context>

        ### COMPLETED
        <completed subtasks>

        ### PENDING
        <remaining operations and constraints>

        ### FAILED ATTEMPTS
        <persistent failure ledger maintained by YAMATO>

        ### KEY DATA
        <values with turn-level provenance>

        ### PINNED FILES
        [YAMATO-RESTORE RESULT(i)]
        <exact tool result copied from canonical turn i>

        ### PINNED CONTENT (WORK PRODUCT)
        [YAMATO-RESTORE CURRENT-ARTIFACT]
        <latest write action and subsequent edits, in order>

        </HISTORY_SUMMARY>"
  },

  # [TURN-DYNAMIC: RECENT RAW WINDOW]
  # The latest K turns bypass compression and retain their native form.
  {
    "role": "assistant",
    "content":
      "<think>...</think>
       <action>
       {\"app\": \"excel\",
        \"action\": \"set_cell\",
        \"sheet\": \"<sheet>\",
        \"cell\": \"<cell>\",
        \"value\": \"<value>\"}
       </action>"
  },

  # [STEP-t: CURRENT RAW OBSERVATION]
  {
    "role": "user",
    "content":
      "<verbatim result of the previous OfficeBench action>

       ## Current app: <active application>
       ## Instruction:
       <actions and JSON schemas available in the active app>
       <switch-app and finish-task instructions>
       ## Command:"
  }
]
```

6.2.4 BASELINE IMPLEMENTATIONS

We compare DMC-5 against four representative context-management settings. Unless otherwise specified, all methods use the same fine-tuned Qwen3.5-27B agent, the same 50-step interaction cap, and the same cost-accounting rules. For methods that require an auxiliary compression model, we use DeepSeek-V4-flash unless stated otherwise. Method-specific trigger and update policies are preserved by design.

No Compression. The agent receives the complete interaction trajectory at every step without any history reduction or auxiliary compressor. This setting serves as the full-history reference for both task performance and end-to-end cost.

FIFO. FIFO retains only the most recent five interaction turns and discards all earlier turns (Yang et al., 2024). It introduces no auxiliary model call and therefore represents a simple window-based context reduction baseline. We use the same five-turn window as in prior long-horizon agent compression evaluations.

AgentDiet. AgentDiet (Xiao et al., 2026) performs inference-time trajectory reduction by revisiting an earlier interaction after a fixed lag and replacing it with a shorter version when redundant, expired, or otherwise unnecessary content can be removed. The reduction is produced by a lightweight reflection model conditioned on the target interaction and its surrounding trajectory context. Due to no official implementation was available, we reproduced the method following the algorithm and prompting strategy described in the paper. The hyperparameters used are the same as those for DMC-5 or follow the default settings from the original paper.

ACON. ACON (Kang et al., 2026) uses generative history compression in which the current compressed memory is recursively updated with newly accumulated interactions. We use the official implementation, with only the adaptations required to interface with our agent/compressor endpoints and evaluation configuration. The agent model, compressor model, maximum interaction steps, and cost accounting are the same as those for DMC-5.

6.3 Extended Experiments

6.3.1 Run-to-run stability

Table 12: Run-to-run stability across three independent evaluations. Values are reported as mean $\pm$ standard deviation.

Benchmark	Method	Task Quality ↑	Cost ↓
AppWorld	No Compression	77.0 ± 2.4	13.71 ± 0.58
	DMC-5	$\mathbf{78.2 \pm 0.9}$	$\mathbf{10.31 \pm 0.35}$
OfficeBench	No Compression	87.0 ± 1.9	5.25 ± 0.30
	DMC-5	$\mathbf{88.0 \pm 1.0}$	$\mathbf{4.49 \pm 0.14}$
8-objective QA	No Compression	EM 39.1 ± 1.8 / F1 47.6 ± 3.1	35.40 ± 1.45
	DMC-5	EM $\mathbf{44.9 \pm 0.3}$ / F1$\mathbf{54.0 \pm 1.0}$	$\mathbf{25.50 \pm 0.53}$

All main results are averaged over three independent runs. To quantify stochastic variation, we report the mean and standard deviation of task quality and end-to-end cost across runs. As shown in Table 12, DMC-5 consistently improves average task quality while reducing cost on all three benchmarks. Moreover, its run-to-run standard deviation is consistently smaller than that of No Compression, indicating improved empirical stability, showing the effectiveness of DMC-5's design that combines full-history access with a compact, integrity-preserving Decision View to reduce sensitivity to irrelevant or corrupted context.

6.3.2 Hyperparameter Sensitivity

DMC-5 has three main context-control hyperparameters: the recent raw-turn window K, compression interval Δ, and token threshold T. Our default configuration uses $K = 1$, $N = 3$, and $T = 6000$. We vary one hyperparameter at a time while keeping the other two fixed.

The results show that DMC-5 is reasonably stable around the default configuration. For the compression threshold, an overly small $T = 3000$ degrades both accuracy and cost efficiency, while increasing T beyond 6000 brings no further quality gain and slightly increases cost. Varying the recent raw-turn window has only a minor effect on accuracy, but larger K consistently increases cost by retaining more uncompressed context. The compression interval exhibits the clearest trade-off: compressing every turn ($N = 1$) substantially increases cost and also hurts accuracy, whereas a larger interval ($N = 5$) slightly reduces task quality without further cost benefit. Overall, the default setting $T = 6000$, $K = 1$, and $N = 3$ provides the best quality–cost balance among the evaluated configurations.

6.3.3 YAMATO Integrity

Integrity Metrics. We evaluate whether compression preserves execution-critical historical information using the same integrity metrics as in the main paper. Execution-critical information refers

Table 13: Hyperparameter sensitivity on AppWorld. We vary one hyperparameter at a time while keeping the others fixed. Gray rows denote the default configuration used in the main experiments.

(a) Threshold T (K=1, N=3)

T	Acc. ↑	Cost ↓
3000	72.4	15.28
6000	78.2	10.31
8000	78.0	10.96
10000	77.0	11.58

(b) Recent K raw(T=6000, N=3)

K	Acc. ↑	Cost ↓
1	78.2	10.31
2	77.8	10.85
3	78.0	11.23

(c) Interval Δ (T=6000, K=1)

N	Acc. ↑	Cost ↓
1	70.6	19.82
3	78.2	10.31
5	76.9	10.68

to exact historical values that may affect subsequent tool actions, including email addresses, monetary values, paths or filenames, UUID/hexadecimal identifiers, access tokens, and URLs. Table 14 summarizes the metric definitions.

Table 14: Definitions of execution-critical information and integrity metrics.

Metric	Definition
Execution-critical information	Historical values that may affect subsequent tool actions, including email addresses, monetary values, paths/filenames, UUID/hex IDs, access tokens, and URLs.
Compression with Errors	Fraction of compression events that contain at least one corrupted execution-critical item.
Corrupted Execution-Critical Items	Fraction of all execution-critical items in the compressed state whose values are inconsistent with the corresponding historical records.
Recall of Execution-Critical Items	Among execution-critical items that are both relevant to the current action and correct in the historical trajectory, the fraction that are present in the Decision View when the action is generated.

Execution-critical items are extracted automatically using deterministic pattern-based rules over the six value types defined above. For corruption, each retained item is compared with its corresponding historical record, and ambiguous matches are marked as *uncertain* and excluded from the error statistics. For recall, an item is considered action-relevant when a later action reuses the exact historical value or re-invokes the operation that originally produced it across at least one compression boundary. The recall denominator therefore contains action-relevant execution-critical items that are correct in the historical trajectory, while the numerator further requires that the item remains present in the Decision View when the action is generated.

$$\text{Recall} = \frac{\#\{\text{action-relevant, correct critical items present in Decision View}\}}{\#\{\text{action-relevant, correct critical items}\}}. \tag{27}$$

$$\text{Corruption Rate} = \frac{\#\{\text{incorrect critical items}\}}{\#\{\text{critical items}\}}. \tag{28}$$

Table 15: Detailed integrity analysis on AppWorld. We compare recursive and append-only history organization with and without YAMATO. Uncertain cases are excluded from the corruption rate.

Metric	ACON	ACON w/ YAMATO	DMC-5 w/o YAMATO	DMC-5 w/ YAMATO
Compression events	711	894	2,581	2,460
Compression with Errors	30/711 (4.2%)	9/894 (1.0%)	77/2,581 (3.0%)	**1/2,460 (0.04%)**
Execution-Critical Items	3,893	4,607	24,771	22,551
Corrupted Execution-Critical Items	48	16	82	**3**
Corruption rate	1.23%	0.35%	0.33%	**0.013%**
Recall of Execution-Critical Items	63.3% (279/441)	87.1% (608/698)	77.7% (839/1,079)	**98.3%** (1,008/1,025)

Integrity Results with and without YAMATO. As shown in Table 15, YAMATO consistently improves integrity under both history organizations. For recursive ACON, it reduces the critical-state error rate from 1.23% to 0.35% and raises recall from 63.3% to 87.1%. For append-only DMC-5, the corresponding error rate drops from 0.33% to 0.013%, while recall increases from 77.7% to 98.3%. These results indicate that append-only history organization and deterministic state restoration are complementary. The former avoids recursive sematic drift, whereas YAMATO protects exact execution-critical values from lossy transcription.

It worth note that the denominators differ across methods because they are defined over action-relevant critical items encountered in each method's own trajectory, which varies with the generated action sequence and interaction length.

Table 16: Validation of pointer restoration and provenance consistency on AppWorld.

Metric	Result
Pointer restoration	
Pinned references	5,181
Valid references	5,181 / 5,181 (100%)
Successfully resolved	5,181 / 5,181 (100%)
Byte-exact restoration	5,169 / 5,181 (99.77%)
Non-byte-exact restoration	12 / 5,181 (0.23%)
Non-byte-exact cases marked `[UNSOURCED]`	12 / 12 (100%)
Silent non-byte-exact restoration	0

Pointer Restoration Fidelity. We first verify whether execution-critical information referenced by pinned pointers is restored faithfully from the Append-only Canonical View. Across 168×3 AppWorld runs, YAMATO emits 5,181 pinned references. All references are valid and successfully resolved to the corresponding raw records. All 5,181 pinned references are successfully resolved to their original canonical records, confirming exact deterministic restoration. Of these, 5,169 remain byte-identical in the final Decision View (99.77%). The remaining 12 cases are also restored correctly, but are subsequently augmented with an explicit `[UNSOURCED]` marker by the provenance guard because the associated claims lack a recognized historical source. Therefore, these byte-level differences reflect deliberate post-restoration annotation rather than any corruption of the restored raw content.

Table 17: Provenance consistency analysis on AppWorld.

Metric	Count	Rate
Audited claim lines	49,700	–
Claims with provenance tags	38,325	77.1%
Correctly grounded tagged claims	37,991 / 38,325	99.13%
Correctly downgraded `[UNSOURCED]` items	12 / 12	100%
Incorrect `[O@i]` assignments	1 / 35,800	0.003%
Agent derivations incorrectly upgraded to `[O@i]`	1 / 2,525	0.040%

Provenance Consistency. We audit 49,700 claim lines in total. Among them, 38,325 factual claims carry explicit provenance tags, while the remaining 11,375 lines are mainly structural continuations in multi-line fields rather than standalone factual claims. Of the tagged claims, 37,991 are supported by the referenced tool output, action, or task instruction, yielding 99.13% provenance consistency. The remaining 334 mismatches are dominated by formatting artifacts, naming normalization, and imprecise turn indices rather than unsupported factual content.

We then inspect the three provenance failure modes most relevant to information integrity. First, all 12 items that lack a valid raw source are correctly downgraded to `[UNSOURCED]`, showing that unsupported information is not silently treated as grounded state. Second, among 35,800 claims labeled with observation provenance `[O@i]`, only one is incorrectly assigned to a non-existent turn-level observation, caused by context truncation. Third, among 2,525 agent-derived claims, only one is incorrectly upgraded from a derivation to observation provenance. These results show

that YAMATO not only preserves pinned values, but also maintains a reliable distinction between observed, derived, and unsupported information.

Table 18: Detailed analysis of repeated failed actions on AppWorld.

Metric	No Compression	ACON	DMC-5
Runs with Repeat ≥ 1	31.3%	**28.3%**	32.3%
Runs with Same-Action Repeat ≥ 3	4.0%	6.2%	**3.2%**
Runs with Same-Action Repeat ≥ 5	1.2%	1.6%	**0.2%**
Maximum Same-Action Repeats	9	39	**6**

Repeated Failed-Action Analysis. We further analyze how compression affects repeated actions after explicit execution failures. Repeated failed actions are detected by normalizing action strings and counting subsequent actions that are identical or have a similarity of at least 0.9 after an explicit execution error. As shown in Table 18, DMC-5 does not reduce the occurrence of ordinary retries, with the rate of runs containing at least one repeated failed action remaining comparable to the baselines. However, it substantially suppresses more severe repetition. The fraction of runs with the same failed action repeated at least five times drops to 0.2%, compared with 1.2% for No Compression and 1.6% for ACON. The same trend appears in the tail, where DMC-5 reduces the maximum same-action repetition to 6, while ACON exhibits an extreme loop with 39 repeats of the same failed action. These results suggest that YAMATO mainly limits the escalation of failed actions into persistent repetition loops rather than eliminating ordinary retry behavior.

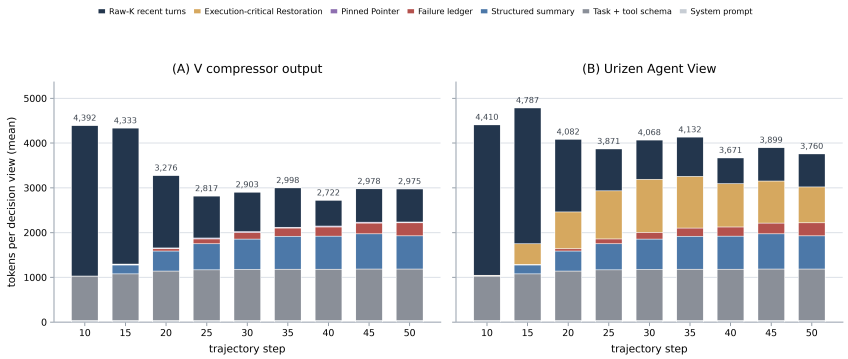

Figure 7: Composition of the compressed context over trajectory steps on AppWorld. (A) Token composition of V compressor output. (B) Token composition of Urizen Decision View after YAMATO restores execution-critical records from pinned pointers.

Decision View Composition. Fig. 7 decomposes both the compressor output and the final Urizen Decision View along the trajectory. Overall, neither view grows with the full interaction history. After compression is triggered, their total sizes remain within a relatively stable range despite the continuously growing canonical trajectory, showing that DMC-5 converts an unbounded interaction history into a bounded working context for both models.

For the compressor output in Fig. 7(A), the structured summary remains compact as the trajectory grows, while the failure ledger stays bounded. The pinned-pointer component is consistently small because execution-critical raw records are referenced rather than regenerated by the compressor. This keeps compressor output nearly stable even when more historical state becomes available in the canonical trajectory.

Fig. 7(B) further shows how YAMATO adapts the agent-facing context over time. The amount of restored execution-critical information first increases as the agent accumulates intermediate states and dependencies, but decreases again in later stages as completed subgoals no longer require the same historical records. Thus, DMC-5 does not permanently retain every previously critical item. Instead, it restores the execution-critical state that remains relevant to the current decision, while completed or no-longer-needed information is absorbed into the compact summary. Together with

the bounded failure ledger and recent raw-turn window, this dynamic restoration keeps the Decision View compact without discarding the exact state required for subsequent tool execution.

YAMATO Overhead. YAMATO consistently improves task performance across both compression architectures, with only modest additional cost under recursive compression and a substantial cost reduction when combined with append-only DMC-5. As shown in Table 5, on recursive ACON, it improves accuracy from 67.3% to 72.0% while slightly increasing total cost from 11.57 to 12.11, although cost per success still decreases from 0.1024 to 0.1001. On append-only DMC-5, YAMATO improves success from 69.1% to 78.2% and simultaneously reduces total cost from 13.31 to 10.31 and cost per success from 0.1147 to 0.0781. This indicates that integrity preservation is most cost-effective when combined with append-only history organization, where compact pointers and selective restoration avoid repeatedly regenerating or carrying the same historical state.

6.3.4 Performance across Task Difficulty

Table 19: AppWorld results stratified by task difficulty. Agent Input denotes the average cumulative agent input per task, and Input Slope measures the average growth of agent input with each interaction step.

Metric	Method	Easy	Medium	Hard
Success	No Compression	91.2%	87.5%	57.1%
	DMC-5 w/ DeepSeek-V4-flash	91.2%	83.3%	**63.5%**
	DMC-5 w/ Qwen3.7-flash	91.2%	70.8%	58.7%
Steps	No Compression	19.9	29.1	38.5
	DMC-5 w/ DeepSeek-V4-flash	19.7	31.3	39.1
	DMC-5 w/ Qwen3.7-flash	19.9	31.3	39.4
Agent Input	No Compression	133K	233K	393K
	DMC-5 w/ DeepSeek-V4-flash	**85K**	**138K**	**187K**
	DMC-5 w/ Qwen3.7-flash	88K	142K	194K
Input Slope	No Compression	500	461	464
	DMC-5 w/ DeepSeek-V4-flash	**232**	**112**	**98**
	DMC-5 w/ Qwen3.7-flash	239	126	118
Cost / Step	No Compression	0.00216	0.00256	0.00324
	DMC-5 w/ DeepSeek-V4-flash	0.00174	0.00199	0.00223
	DMC-5 w/ Qwen3.7-flash	**0.00165**	**0.00180**	**0.00203**
Total Cost	No Compression	2.23	3.57	7.91
	DMC-5 w/ DeepSeek-V4-flash	1.82	2.99	5.50
	DMC-5 w/ Qwen3.7-flash	**1.71**	**2.70**	**5.09**
Cost / Success	No Compression	0.0429	0.0850	0.2197
	DMC-5 w/ DeepSeek-V4-flash	0.0350	**0.0748**	**0.1375**
	DMC-5 w/ Qwen3.7-flash	**0.0328**	0.0794	0.1376

Table 19 analyzes DMC-5 across the official AppWorld difficulty levels. The efficiency gain becomes more pronounced as task difficulty increases. With DeepSeek-V4-flash, cumulative agent input is reduced from 133K to 85K on easy tasks, from 233K to 138K on medium tasks, and from 393K to 187K on hard tasks. The corresponding input-growth slopes decrease from 500/461/464 to 232/112/98, showing that DMC-5 increasingly suppresses context growth on longer and more complex trajectories.

These savings do not come from shortening the interaction. The number of agent steps remains similar to No Compression across all three difficulty levels, while cost per step consistently decreases. Consequently, total cost falls from 2.23/3.57/7.91 to 1.82/2.99/5.50 with DeepSeek-V4-flash. The reduction is largest on hard tasks, where more accumulated history provides greater opportunity for compression and prefix reuse.

Task quality exhibits a complementary trend. DMC-5 preserves the 91.2% success rate on easy tasks and incurs a moderate decrease on medium tasks, while improving hard-task success from 57.1%

to 63.5% with DeepSeek-V4-flash. Replacing DeepSeek-V4-flash with the cheaper Qwen3.7-flash further reduces cost across all difficulty levels, but causes a larger quality drop on medium tasks. This indicates that the efficiency benefit of the dual-view architecture is consistent across compressors, whereas preserving task quality, particularly on more demanding trajectories, depends on the compressor's ability to construct an accurate Decision View.

6.3.5 Sensitivity to Compressor Pricing

We further examine whether the cost advantage of DMC-5 depends on a particular compressor pricing configuration. Fig. 8 recomputes the end-to-end cost on fixed AppWorld trajectories while varying the compressor input price, cached-input discount, and output price. The agent trajectories and token counts are kept unchanged, so the analysis isolates pricing effects without introducing behavioral variation.

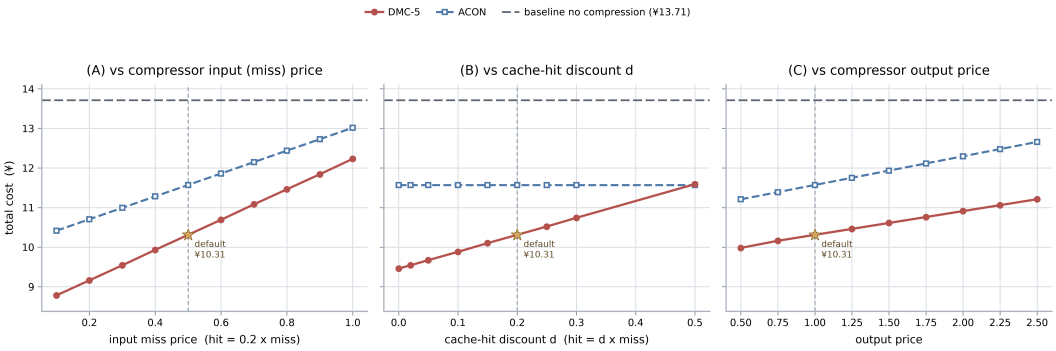

Figure 8: Pricing sensitivity on AppWorld using fixed trajectories. We recompute billed cost while varying one compressor pricing coefficient at a time and keeping all observed token counts and agent behaviors unchanged. (A) varies the compressor uncached-input price while keeping the cached-input price at $0.2\times$ the uncached-input price. (B) varies the cached-input price ratio d, where $p^{\text{hit}} = d\,p^{\text{miss}}$. (C) varies the compressor output price. Vertical dashed lines denote the default configuration, and the horizontal dashed line denotes the No Compression cost.

Fig 8(A) examines the compressor uncached-input price. As the input price increases, the costs of both compression methods increase, but DMC-5 remains consistently cheaper than ACON and No Compression throughout the evaluated range. Notably, DMC-5 still provides a net saving even when the compressor uncached-input price exceeds that of the specialized agent. This is because most historical compressor input is served through the cheaper cached-input path, so a relatively expensive compressor does not necessarily translate into a high end-to-end cost.

Fig 8(B) isolates the effect of the cached-input price ratio. ACON is almost unaffected because its recursively rewritten summaries provide little reusable prefix, whereas DMC-5 becomes progressively cheaper as the cached-input price decreases. Even when cached input is priced at 50% of uncached input, DMC-5 remains below the No Compression baseline and approximately matches ACON. The gain therefore does not require an extreme cache discount, while stronger discounts directly translate the high prefix-reuse rate of DMC-5 into larger cost savings.

Fig 8(C) varies the compressor output price. DMC-5 remains cheaper than both ACON and No Compression over the entire range, including settings where compressor output is priced comparably to or even higher than specialized-agent output. This robustness follows from the compact compressor output enabled by structured summaries and pinned pointers, which limits the amount of expensive generated content.

Together, these results show that DMC-5 is not tied to a narrow pricing configuration. Its cost advantage persists across substantially different input prices, cache discounts, and output prices, and is consistently stronger than recursive compression.

6.3.6 WALL-CLOCK LATENCY

DMC-5 optimizes billed inference cost rather than minimum response latency. Because each compression event introduces an additional model invocation, the monetary savings are accompanied by increased wall-clock time. Table 20 reports the measured average end-to-end latency per task.

Table 20: Average wall-clock latency per task (seconds).

Method	AppWorld	OfficeBench	8-objective QA
No Compression	69	38	122
ACON	78	42	149
DMC-5	94	44	186

The overhead is smallest on the shorter OfficeBench workload and larger on longer trajectories that invoke the compressor more frequently. Thus, DMC-5 exposes a clear cost–latency trade-off: it substantially reduces billed inference cost, but is not designed to minimize serial response time. Parallel or asynchronous compression is orthogonal to the proposed context representation and remains an interesting direction for deployment-oriented optimization.

6.3.7 REPRESENTATIVE TRAJECTORIES

We qualitatively inspect representative AppWorld trajectories to illustrate how compression errors affect downstream actions. Table 21 summarizes three cases in which recursive compression loses or distorts task state while DMC-5 succeeds, together with one DMC-5 failure case that illustrates the boundary of context preservation.

Table 21: Representative AppWorld trajectories. The first three cases illustrate compression-induced failures avoided by DMC-5, while the last case shows a failure outside the scope of YAMATO.

Task	Outcome	Failure mode	DMC-5 behavior
`986aa4e_1`	ACON fails at 50 steps; DMC-5 passes 3/3	An access token disappears across recursive summaries, forcing re-login and causing a previously failed login action to be repeated.	YAMATO preserves the exact token in KEY DATA with provenance, avoiding re-login and repeated failure.
`09b0ee6_2`	ACON fails at 50 steps; DMC-5 passes 3/3	A recursive summary regresses from "all 83 songs processed" to "83 song IDs ready for lookup", discarding accumulated aggregation state.	Append-only reconstruction avoids recursive state regression, while the final aggregate result remains grounded in KEY DATA.
`634f342_1`	ACON fails at 50 steps; DMC-5 passes 2/3	API documentation, access tokens, and completed file-processing state are lost or distorted, causing repeated documentation lookup, re-login, and re-execution.	Pinned raw records and grounded state preserve API results, credentials, and completed progress across compression rounds.
`9016950_1`	DMC-5 fails at 24 steps	All required historical information is preserved, but the agent incorrectly resolves the scope of "them" and performs an unnecessary action.	The failure originates from task interpretation rather than context loss, illustrating the boundary of YAMATO.

Case 1 Preserving exact execution state. Task `986aa4e_1` requires the agent to modify a Spotify playlist according to comments in Todoist, leave a confirmation comment, and mark the task as completed. ACON reaches the 50-step interaction limit, whereas DMC-5 succeeds in all three runs.

The failure begins when recursive compression retains the statement that the agent is already logged in but drops the exact Spotify access token. The downstream agent therefore knows that authentication has occurred but no longer has the credential required for subsequent API calls.

`ACON summary`

```
- Access token from earlier login is stored in session; no need to re-login.

Agent
[27] According to the history summary, I logged in successfully ...
[28] I need the access_token ... but I don't have the access token
[29] I need to log in to Spotify to get the access token ...
```

Re-authentication also causes the agent to repeat a previously encountered argument error.

```
[14] login(email=...)        -> Unexpected parameter "email"
[15] login(username=...)     -> success
...
[30] login(email=...)        -> the same error appears again
[32] login(username=...)     -> success
```

DMC-5 instead preserves the exact credentials and task identifiers in KEY DATA, together with their observation provenance.

```
### KEY DATA
- Todoist access_token: eyJ... [O@7]
- Spotify access_token: eyJ... [O@15]
- Project id: 280 [O@10]
- Task id: 6342 [O@11]
- Playlist id: 654 [O@16]
- Songs added: 248, 160, 65, 140, 115 [O@19-23]
- Songs removed: 260, 49 [O@16, O@33-34]
```

The agent therefore reuses the original token instead of re-authenticating, avoids repeating the failed login pattern, and completes the remaining playlist operations and task update. This case illustrates how preserving exact execution state prevents a local compression error from propagating into repeated interactions.

Case 2 Preventing recursive state regression. Task `09b0ee6_2` asks the agent to identify the four artists with the fewest unique songs across all of the user's Spotify playlists. The answer requires aggregating metadata from 83 unique songs. ACON reaches the interaction limit without producing the answer, while DMC-5 succeeds in all three runs.

Before the fifth compression, the recursive summary correctly records that all 83 songs have already been retrieved and that only aggregation remains.

```
Earlier ACON summary
- Added Marcus Lane ..., Hazel Winter ..., Lucas Grey ...
- All 83 songs fetched. Ready to aggregate artist counts ...
```

After another recursive update, however, the same task state regresses to an earlier stage.

```
Later ACON summary
- Compiled 83 unique song IDs ready for artist lookup.
```

The accumulated artist statistics are no longer available, so the agent begins retrieving the song metadata again and eventually exhausts the 50-step budget.

DMC-5 avoids this failure because each compressed state is reconstructed from the append-only raw trajectory rather than from the previous summary. Once the aggregation has been completed, the resulting state is explicitly grounded and retained.

```
### KEY DATA
- 4 least-song artists (1 song each):
  Grace Harrison, Luna Starlight, Evelyn Rose, Liam Palmer [O@21]

### COMPLETED
- Collected 83 unique song IDs across all playlists [O@19]
- Counted songs per artist (28 artists) [O@20]
- Identified 4 artists with least songs [O@21]

### PENDING
- Submit answer:
  "Grace Harrison, Luna Starlight, Evelyn Rose, Liam Palmer" [D@21]
```

The agent subsequently submits the result without repeating the enumeration. This case demonstrates that append-only reconstruction prevents completed progress from being recursively rewritten into an earlier task state.

Case 3 Preserving tool records and completed progress. Task `634f342_1` requires the agent to read a file containing 14 old songs, locate them across Spotify playlists, move them into a new "Archived Songs" playlist, and remove them from their original playlists. ACON reaches the 50-step limit, whereas DMC-5 succeeds in two of three runs.

Under recursive compression, several forms of operational state disappear. The agent repeatedly reloads API documentation that had already been observed.

```
[13] show_api_descriptions(app_name="spotify")
[17] show_api_descriptions(app_name="spotify")
...
[37] show_api_descriptions(app_name="spotify")
[38] show_api_doc(app_name="spotify", ...)
```

The exact access token is also lost, leading the agent to reconstruct or reacquire it.

```
[35] spotify_token = "BQDh2JjK8vN5mR7pL3xW9..."
     -> unterminated / truncated string
...
[40] The spotify_token variable was lost.
     I need to get it again by logging in.
```

More importantly, an operation that has already succeeded is later rewritten as unfinished.

```
Earlier state
- old_songs.txt successfully read and parsed into 14 songs

Later recursive summary
- Attempted to read old_songs.txt but failed initially ...
- Need to retry reading old_songs.txt with the access token.
```

DMC-5 retains exact credentials and completed state in grounded KEY DATA, while large tool records can be referenced through pinned raw results.

```
### KEY DATA
- file_system access_token: eyJ... [O@12]
- spotify access_token: eyJ... [O@16]
- old_songs.txt content: 14 songs listed [O@14]
- Matched song IDs: [29, 43, 74, ..., 311] [O@29]
- Archived Songs playlist ID: 654 [O@34]

### PINNED DOCS
<<TURN_40_RESULT>>
```

The agent therefore avoids repeatedly reconstructing already observed tool state and can continue from the preserved task progress. This case illustrates why exact raw restoration is particularly useful for operational records that are difficult to reproduce reliably through natural language summaries.

Case 4 Failure outside the scope of context preservation. Task `9016950_1` asks the agent to create a Venmo account for each parent who does not already have one and then notify them by text message. DMC-5 terminates normally after 24 steps but passes only 6 of 7 checks.

The agent correctly identifies that Chad does not have a Venmo account while Sabrina already has one, and therefore creates an account only for Chad. The subsequent failure occurs when the agent incorrectly expands the reference of "them" to both parents.

```
[19] search_users(Chad)      -> no Venmo account
[20] search_users(Sabrina)   -> existing Venmo account
[21] signup(Chad, ...)       -> success

[22] "Now I need to send text messages to both parents."
     send_text_message(Chad, ...)

[23] send_text_message(Sabrina, ...)   <- unnecessary action
[24] complete_task()
```

This failure is not caused by missing compressed state. The contact information, account status, message content, and authentication credentials remain available throughout the trajectory, and the agent does not repeat any login or information-retrieval operation. Instead, the error arises from incorrect referential reasoning over the task instruction.

The case therefore illustrates a clear boundary of DMC-5. YAMATO protects historical information from omission or corruption during compression, but it cannot correct semantic reasoning errors made by the underlying agent when the required information is already available.

Across the three successful cases, recursive compression fails through different manifestations of information loss, including the loss of exact execution-critical values, regression of completed task state, and omission of previously observed tool records. DMC-5 mitigates these failures through the Append-only Canonical View, the compact Decision View, and YAMATO's deterministic restoration of execution-critical information. The final case further illustrates the scope of the method. YAMATO preserves information integrity during lossy compression, but reasoning failures caused by limitations of the underlying agent model are outside the scope of DMC-5 and do not directly benefit from these mechanisms.

6.4 Limitation and Future Work

Our experiments show that DMC-5 remains effective across different compressor models, task difficulties, and a broad range of pricing configurations, while YAMATO substantially mitigates compression-induced information loss and repeated failures. Nevertheless, several broader settings remain beyond the current design scope.

First, DMC-5 is designed primarily to reduce billed inference cost by separating full-history integration from specialized action generation and exploiting prefix reuse in the compressor. The current implementation already achieves substantial cost savings, but compression is performed synchronously, so the additional compressor calls can increase wall-clock latency. Further reducing this latency overhead remains an open problem beyond the current scope of DMC-5. Future work could therefore investigate asynchronous or parallel compression, speculative refresh, and overlap between compression and agent execution.

Second, YAMATO effectively protects execution-critical information from corruption or omission during compression. However, when the required information is already available, the underlying agent may still make task-interpretation or reasoning errors. Handling such reasoning errors remains beyond the current scope of DMC-5 and may benefit from stronger planning, verification, or self-correction mechanisms.

Third, DMC-5 currently uses a fixed compression policy with predefined thresholds and a fixed division of roles between the general compressor and specialized agent. Our sensitivity experiments indicate that the default policy is robust across the evaluated settings, but different workloads may favor different compression timing, context allocation, or compressor choices. Adapting compression behavior to workload characteristics remains an open direction beyond the current scope of DMC-5. Future work could explore adaptive compression triggers, dynamic model selection, and tighter integration with serving-side cache management, as well as broader evaluation on longer-horizon and more heterogeneous agent workloads.

6.5 Related Work

Section 4 provides a compact comparison in the main paper. Here we further distinguish related methods by the state they operate on, how that state is updated over time, and where computational savings arise.

6.5.1 Agent Context Compression and Memory

Prompt and context compression. Prompt-compression methods primarily reduce the representation presented to a downstream model. LLMLingua scores and removes less informative tokens, while LongLLMLingua introduces query-aware compression for long inputs (Jiang et al., 2023b; 2024). LLMLingua-2 learns a bidirectional token classifier from distilled compression data, Dodo performs input-dependent contextual compression for decoder-only models, and RECOMP selects or summarizes retrieved evidence before generation (Pan et al., 2024; Qin et al., 2024; Xu et al., 2024). These methods mainly optimize the length or information density of a given model input.

A complementary line compresses text into learned latent representations. Gisting encodes reusable information into special tokens, AutoCompressors recursively construct summary vectors, and ICAE

maps long inputs into compact memory slots (Mu et al., 2023; Chevalier et al., 2023; Ge et al., 2024). Latent representations can achieve high compression density, but generally require a compatible downstream model. DMC-5 instead produces a textual Decision View that can be consumed by an independently deployed specialized agent. More importantly, DMC-5 focuses on repeated trajectory integration rather than compressing a single static input.

Persistent agent memory. Agent-memory systems focus on organizing historical information for later access. MemGPT moves information across memory tiers, HiAgent groups interactions by subgoal, Generative Agents retrieves experiences and forms higher-level reflections, and Reflexion stores verbal feedback from previous failures (Packer et al., 2023; Hu et al., 2025; Park et al., 2023; Shinn et al., 2023). MEMORYLLM maintains an updatable latent memory pool, while RAPTOR organizes recursive summaries in a retrieval hierarchy (Wang et al., 2024a; Sarthi et al., 2024). These methods primarily address how historical information is stored, organized, or retrieved. DMC-5 addresses a different problem: how a growing interaction trajectory is repeatedly integrated into the next decision while controlling both agent-side input and compressor-side cost.

Trajectory-update strategies. Recent agent context-management methods can be characterized by their dominant history-update behavior. These categories describe update patterns rather than mutually exclusive method families.

Selective-update methods retain a recent window or selected trajectory elements while removing or folding less relevant content (Xiao et al., 2026; Ning et al., 2026; Hu et al., 2025; Ye et al., 2026; Verma, 2026). They reduce the amount of history processed at each step, but information removed from the active state may be unavailable when it becomes relevant again.

Recursive-update methods maintain a bounded state by combining the previous compressed memory with newly generated interactions (Zhou et al., 2026; Yu et al., 2026; Wu et al., 2025). This avoids rereading the complete trajectory, but repeatedly rewriting the same state can accumulate omissions or altered details (Kang et al., 2026; Shi et al., 2026). It also changes the prefix presented to the compressor across rounds, limiting prefix-cache reuse.

Full-history recompression instead reconstructs a compact representation from retained raw context, preserving access to earlier information but repeatedly processing an expanding history (Li et al., 2023; Jiang et al., 2024; Xu et al., 2024).

DMC-5 combines full-history access with a different update protocol. Its Append-only Canonical View preserves raw trajectory records and a stable historical prefix for repeated compressor calls, while a separate compact Decision View is regenerated for the specialized agent. YAMATO further restores selected execution-critical records directly from the Canonical View rather than relying on model rewriting. The design therefore jointly targets history recoverability, compression integrity, prefix-cache reuse, and agent-side context cost.

6.5.2 Cache-Aware LLM Inference

Prefix reuse and serving-side cache organization. LLM serving systems reduce repeated computation by organizing reusable KV states. PagedAttention manages KV blocks using virtual-memory techniques, SGLang organizes shared prefixes in a radix tree, Prompt Cache exposes reusable prompt modules, and FastTree optimizes attention execution for prefix-sharing request trees (Kwon et al., 2023; Zheng et al., 2024; Gim et al., 2024; Pan et al., 2025a). These systems optimize how reusable states are stored or scheduled once shared prefixes exist.

DMC-5 operates at a different layer. It does not modify the serving runtime or directly manipulate KV tensors. Instead, it structures repeated compressor requests so that the historical portion remains byte-identical across rounds. The application therefore creates a stable reusable prefix that standard provider-side prefix caching can exploit.

KV-cache selection and compression. Another line reduces the memory or bandwidth footprint of cached attention states. H_2O retains recent tokens and attention heavy hitters, Scissorhands exploits persistent token importance, StreamingLLM combines attention sinks with a recent window, SnapKV selects head-specific KV positions, QUEST loads query-relevant KV pages, and ThinKV

combines reasoning-aware retention with adaptive quantization and eviction (Zhang et al., 2023; Liu et al., 2023; Xiao et al., 2024; Li et al., 2024; Tang et al., 2024; Ramachandran et al., 2026).

These methods compress or select internal model state. They can reduce inference memory or attention cost without necessarily reducing the prompt tokens billed to an API endpoint. DMC-5 instead reduces the textual context repeatedly processed by the acting model while organizing compressor input for prefix reuse. The two optimization layers are therefore complementary.

Cache reuse across requests and workflows. CacheGen encodes and transfers KV states across machines, CacheBlend reuses non-prefix cached chunks with selective recomputation, KVFlow schedules prefix retention across agent workflows, and LRAgent shares base-model cache states across LoRA-specialized agents (Liu et al., 2024b; Yao et al., 2025; Pan et al., 2025b; Jeon et al., 2026). These methods improve how reusable cache states are transferred, scheduled, or shared across requests.

CAPC is particularly related to DMC-5 because it explicitly studies the conflict between prompt compression and prefix reuse (Song, 2026). CAPC optimizes compression within an individual prompt so that shortening the input does not unnecessarily destroy reusable cached prefixes. DMC-5 addresses the complementary multi-turn problem. It organizes a growing trajectory across repeated compressor calls so that the historical prefix remains reusable, while the specialized agent receives a separate Decision View optimized for action generation. Thus, cache management and semantic compression remain separate mechanisms, but DMC-5 connects them through the trajectory-update protocol.

6.5.3 Multi-Model Collaboration and Agent Efficiency

Routing and cascades. Model routers and cascades reduce inference cost by deciding which model should handle a request. FrugalGPT learns cascades across model APIs, RouteLLM learns routing decisions from preference data, Hybrid LLM predicts query difficulty, and AutoMix starts with a cheaper model and escalates uncertain requests (Chen et al., 2024; Ong et al., 2025; Ding et al., 2024; Aggarwal et al., 2024). Their central decision is which model should produce the answer.

DMC-5 does not route requests between interchangeable models. Both models remain active and have fixed asymmetric roles. The general compressor performs full-history integration, while the specialized agent performs downstream action generation. This functional decomposition also allows each model to receive a context designed specifically for its role.

Collaborative generation. Other multi-model systems collaborate by exchanging generated outputs. LLM-Blender ranks and fuses candidate responses, Mixture-of-Agents propagates responses through layered model groups, DUET transfers a compact reasoning signal between models, and speculative decoding uses a smaller model to draft tokens that a larger model verifies (Jiang et al., 2023a; Wang et al., 2025; Chen et al., 2026; Leviathan et al., 2023). These methods collaborate over answers, reasoning signals, or token proposals.

DMC-5 instead collaborates over persistent agent state. The compressor constructs the Decision View from the evolving trajectory, and the specialized agent acts from that state. The collaboration therefore persists across interaction turns rather than forming a one-shot generation pipeline.

Agent evaluation and end-to-end efficiency. Long-horizon agent benchmarks expose interaction costs that are largely absent from single-turn evaluation. AgentBench spans multiple interactive environments, AgentBoard reports intermediate progress, WebArena evaluates agents on reproducible websites, and SWE-bench evaluates repository-level software-engineering workflows (Liu et al., 2024a; Ma et al., 2024; Zhou et al., 2024; Jimenez et al., 2024). AppWorld and OfficeBench extend this setting to executable API workflows and office automation (Trivedi et al., 2024; Wang et al., 2024b).

Final task success alone does not capture the cost of additional model calls, retries, or longer trajectories. Agentless demonstrates that simpler pipelines can outperform more elaborate agents at lower cost, while broader work on agent evaluation argues for reporting resource consumption together with task quality (Xia et al., 2025; Kapoor et al., 2025). DMC-5 follows this perspective by jointly

measuring task performance, agent-side input, compressor cache reuse, and end-to-end billed cost, thereby capturing both compression savings and interaction overhead caused by lost state.