BRANCHMEM: BRANCHING REASONING WITH PERSISTENT LATENT MEMORY FOR LANGUAGE MODEL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent memory enables LLM-based agents to improve by accumulating and reusing past experience. Latent memory provides a more compact representation of past experience than raw trajectories, but existing approaches often either absorb experience into shared trainable components or require additional optimization for each explicit latent memory. We introduce BRANCHMEM, a framework that uses a lightweight compressor to transform successful experiences into persistent latent memories. Each successful experience is encoded once into a memory that can later be retrieved and reused for subsequent tasks, without updating the underlying reasoner or optimizing each memory separately. We further find that different latent memories often alter generation only at a few critical points, yet these differences can redirect the entire reasoning process. Based on this observation, BRANCHMEM introduces Action Branching, which allows different latent memories to guide alternative reasoning paths at critical points while preserving the reasoning already performed, enabling more effective use of past experience for solving new problems. We evaluate BRANCHMEM across mathematical reasoning, code generation, and information-seeking tasks, demonstrating its ability to leverage past experience for new problems and showing how Action Branching further improves reasoning by exploiting alternative paths induced by latent memories.

1 INTRODUCTION

Large language model (LLM) agents can improve by accumulating and reusing successful experience. Existing memory systems commonly preserve past trajectories, summaries, or procedural rules in text (Park et al., 2023; Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2025; Fang et al., 2026). Latent memory provides a different interface: it compresses past experience into continuous representations that directly condition generation (Zhang et al., 2026; Wu et al., 2026), providing a compact mechanism for retaining and reusing experience while leaving the underlying Reasoner unchanged.

Recent work has explored different forms of latent memory. TokMem preserves experience as individually addressable latent memory units but requires explicit optimization of the corresponding memories, whereas MemGen produces latent guidance through a shared Memory Weaver (Wu et al., 2026; Zhang et al., 2026). These studies show that past experience can be represented in latent form and reused during generation. However, existing work has primarily focused on whether latent memory improves final outputs; **less is known about when and how different memories induce distinct reasoning behaviors.**

Given different latent memories, how do they change the Reasoner's reasoning process, and what regularities characterize these effects?

To study this question, we introduce BRANCHMEM (Figure 1), which uses a shared compressor to encode successful experiences into persistent latent memories while preserving each memory as an independently retrievable and reusable unit. Each successful trajectory is encoded once into a latent value and stored together with its retrieval key; adding a new memory requires neither updating the frozen Reasoner nor separately optimizing the resulting memory. More importantly,

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.

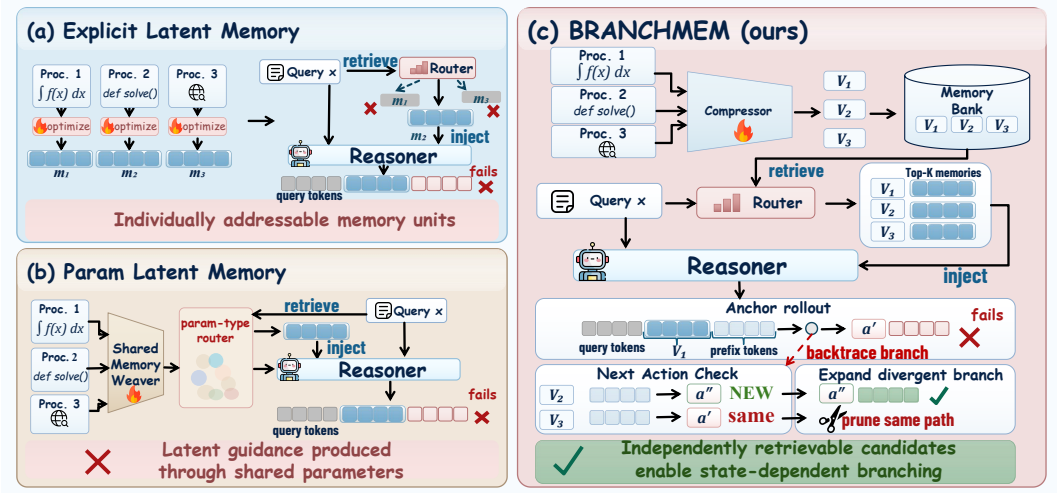


Figure 1: Comparison of latent-memory interfaces. BRANCHMEM preserves experiences as independently retrievable memories and exposes memory-induced alternatives during reasoning.

this independently retrievable representation allows different memories to be substituted under the **same Reasoner and the same reasoning context**, enabling their behavioral effects to be compared directly.

Concretely, we first use one retrieved memory to generate an anchor trajectory, then hold the reasoning prefix fixed and replace only the latent memory through teacher-forced next-action scoring. This aligned comparison reveals a consistent behavioral pattern: **different latent memories remain aligned over most of the reasoning process and become behaviorally distinct primarily at a small number of critical decisions.** Across GSM8K and KodCode, candidate variation at the first divergence point is 53.9–$959.0\times$ its shared-prefix level, and these divergences tend to occur where the frozen Reasoner has a smaller local Top-1/Top-2 margin. Rather than steering the Reasoner toward independent trajectories from the outset, different memories therefore more often act as **local interventions on a largely shared reasoning process**.

This behavior also exposes a limitation of **query-level retrieval**. Retrieval recalls and ranks memories using only the original query, whereas behavioral differences between memories often emerge only under later reasoning states. As a result, initial similarity or retrieval rank provides limited information about which retrieved candidate will produce a useful alternative during subsequent reasoning. Retrieval is therefore better suited to **narrowing the candidate memory set** than to determining, once and for all, which memory should guide the remainder of the reasoning process. As the reasoning prefix changes, previously aligned candidates may become behaviorally distinct, causing useful alternatives within the retrieved set to emerge progressively during reasoning.

Based on this observation, we introduce **Action Branching**. Given a fixed retrieved memory set, **BRANCHMEM** first generates an anchor trajectory with the Top-1 memory and creates a branch only when another memory proposes a different action under the current prefix. The reasoning prefix before the divergence is reused, while the remaining memories in the selected divergence group are re-evaluated under the newly reached reasoning state; other groups remain pending at the parent node. Recursively applying this procedure progressively exposes state-dependent alternatives and turns a static retrieved memory set into a reasoning tree induced by memory-dependent behavioral differences, without treating every memory as an independent full trajectory from the outset.

We evaluate BRANCHMEM with two frozen Reasoners across five benchmarks spanning mathematical reasoning, code generation, and information seeking. The localized intervention pattern remains consistent across models, tasks, and training objectives. Moreover, compared with the corresponding Top-1 variants, Action Branching improves five-task average solution coverage by 4.01–7.84 **percentage points**, showing that the same retrieved memory set contains useful reasoning alternatives beyond the initially selected Top-1 memory.

Our contributions are threefold. First, we introduce a persistent latent-memory interface that encodes successful experiences once while preserving them as independently retrievable and reusable memories for a frozen Reasoner. Second, we characterize a consistent behavioral structure of latent memory: different memories share most of the reasoning process and become distinct primarily at sparse critical decisions, while query-level retrieval alone provides limited information about which retrieved candidate will yield a useful alternative under later reasoning states. Third, based on this structure, we introduce Action Branching, which reuses shared reasoning and recursively expands memory-induced alternatives during inference, and validate the resulting framework across two Reasoners and five benchmarks.

2 Related Work

Experience Memory for LLM Agents. LLM agents can improve across interactions by preserving and reusing past experience. Existing approaches commonly represent experience as explicit textual or executable units. Generative Agents and MemoryBank retrieve past records, Reflexion stores verbal feedback, and Voyager preserves executable skills (Park et al., 2023; Zhong et al., 2024; Shinn et al., 2023; Wang et al., 2023a). ExpeL distills trajectories into reusable rules, AWM induces workflows from prior interactions, and MemP organizes experience as retrievable procedural memory (Zhao et al., 2024; Wang et al., 2025; Fang et al., 2026). These methods reuse experience through textual or executable interfaces. BranchMem instead represents successful experience as persistent continuous signals that directly condition a frozen Reasoner.

Latent Memory and Continuous Representations. Prior work has explored continuous representations for context compression, reasoning, and memory (Mu et al., 2023; Chevalier et al., 2023; Ge et al., 2024; Cheng et al., 2024; Hao et al., 2025; Xu et al., 2025a; Fu et al., 2026; Feng et al., 2026). More closely related to our setting, Memory Tokens and TokMem preserve sequence- or procedure-specific latent representations and explicitly optimize the corresponding latent units (Sastre & Rosá, 2025; Wu et al., 2026), while MemGen generates latent guidance through a shared Memory Weaver (Zhang et al., 2026). BranchMem adopts a shared parameterized memory module while preserving each encoded experience as an independently retrievable latent memory. Our focus is complementary to prior work on learning latent representations: we study how retrieved memories differ in their effects under the same reasoning context and how these differences can be utilized during reasoning.

Multi-path Reasoning and Search. Exploring alternative reasoning paths is a common strategy for improving autoregressive reasoning. Self-Consistency independently samples multiple complete chains and aggregates their answers, while Tree of Thoughts, Graph of Thoughts, and LATS explicitly search over intermediate reasoning states (Wang et al., 2023b; Yao et al., 2023; Besta et al., 2024; Zhou et al., 2024). BranchMem differs primarily in the **source and structure of its alternatives**. Rather than generating alternatives through independent sampling or a separate search proposal mechanism, Action Branching derives them from differences induced by retrieved latent memories under the current reasoning state. It introduces a branch only when memories propose distinct local decisions and recursively re-evaluates their effects as the reasoning state changes.

3 Method

BranchMem represents successful past experience as persistent latent memories that can be independently retrieved and applied to a frozen Reasoner. We first learn a shared latent-memory interface that converts successful trajectories into reusable memory values and stores them with retrieval keys. We then compare retrieved memories under aligned reasoning contexts, and finally use Action Branching to recursively expand the alternatives induced by these memories as the reasoning state changes. Figure 2 provides an overview.

3.1 Problem Setup and Latent Conditioning

Let π_θ denote a frozen LLM Reasoner and x a task input. Given x, the Reasoner generates a trajectory $\tau = (u_1, \ldots, u_T)$, where u_t is the token generated at position t; throughout the paper, we refer

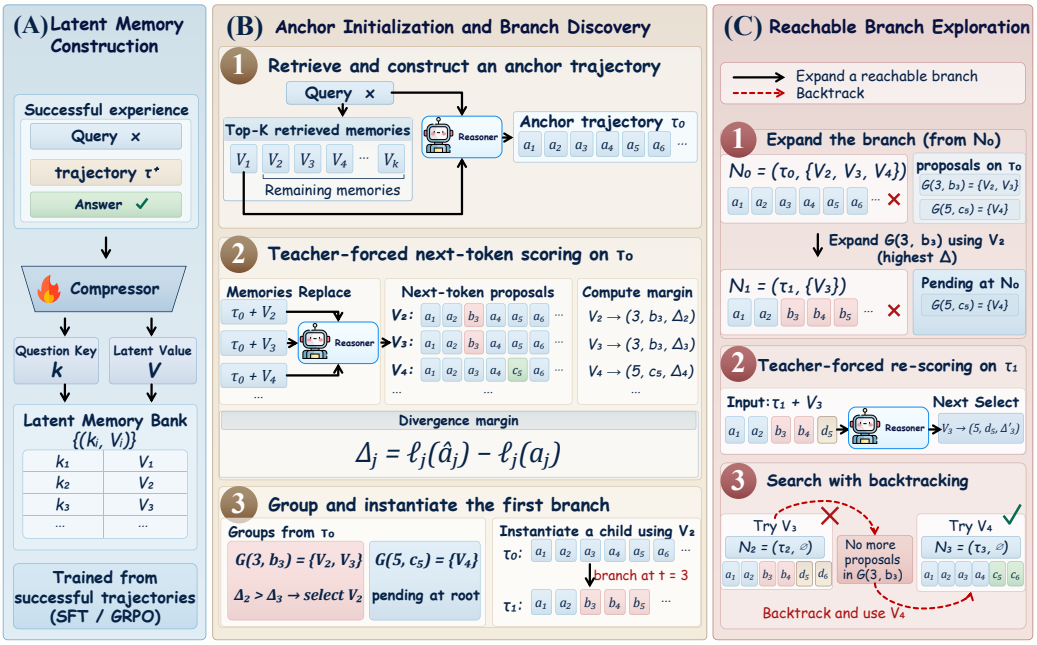


Figure 2: Overview of BRANCHMEM. Persistent memory construction, aligned branch discovery, and recursive Action Branching.

to each next-token decision as an **action**. We consider a collection of successful past experiences $\mathcal{H} = \{(x_i, \tau_i^+)\}_{i=1}^N$, where τ_i^+ is a successful trajectory for task x_i.

Each experience is represented by a fixed-length latent value $V = [v_1, \ldots, v_S] \in \mathbb{R}^{S \times d_m}$. A learned projection P_{out} maps V into the Reasoner embedding space, and generation is conditioned as $\tau \sim \pi_\theta(\cdot \mid x, P_{\text{out}}(V))$. The projected latent slots are inserted after the query tokens and before response generation.

These latent positions participate in the frozen Reasoner through standard causal self-attention. Suppressing the attention-head index and output projection for clarity, at layer ℓ, their contribution to response position t is

$$a_{t,\text{mem}}^{(\ell)} = \sum_{s=1}^{S} \alpha_{t,s}^{(\ell)} W_V^{(\ell)} h_{s,\text{mem}}^{(\ell-1)}, \tag{1}$$

where $\alpha_{t,s}^{(\ell)}$ is the causal attention weight from response position t to latent-memory position s. Since the Reasoner parameters θ remain frozen, replacing V changes the memory-conditioned hidden states and next-token distribution while keeping the underlying model fixed. This allows different latent memories to be compared under the same Reasoner and the same reasoning prefix.

3.2 PERSISTENT LATENT-MEMORY INTERFACE

Given a successful experience $(x_i, \tau_i^+) \in \mathcal{H}$, a shared compressor C_ϕ maps the successful trajectory to a fixed-length latent value $V_i = C_\phi(\tau_i^+) \in \mathbb{R}^{S \times d_m}$, while the corresponding task input x_i provides a query-side key K_i. The resulting pairs form a persistent memory bank

$$\mathcal{M} = \{(K_i, V_i)\}_{i=1}^N. \tag{2}$$

Each V_i is stored as an independently retrievable and reusable memory. Because the compressor is shared across experiences, once the interface is trained, a new successful trajectory can be added to the memory bank with a single encoding pass, without updating the frozen Reasoner or performing additional per-memory optimization.

Interface learning. The latent values have no direct representation-level supervision; instead, the shared memory interface is trained through their effect on the frozen Reasoner. Let ϕ denote the trainable memory-interface parameters, including the compressor adapters and output projection. We optimize

$$\max_{\phi} \ \mathbb{E}_{(x_i,\tau_i^+)\sim\mathcal{H}} \left[\mathcal{J}_{\text{train}}(\phi; x_i, \tau_i^+)\right], \tag{3}$$

while keeping the Reasoner parameters θ fixed. Under SFT, $\mathcal{J}_{\text{train}}$ is the autoregressive likelihood of τ_i^+. Under GRPO (Shao et al., 2024), it is the group-relative objective computed from rollouts conditioned on $C_\phi(\tau_i^+)$. The two training regimes therefore share the same memory construction and conditioning mechanism and differ only in the supervision signal. Full training details are provided in Appendix A.2.

Retrieval. Each stored memory contains a paired query key and latent value (K_i, V_i). For a new query x, we project its query representation K_x toward the latent-value space using a lightweight projection P_{kv}, obtaining $\widehat{V}(x) = P_{\text{kv}}(K_x)$. Memories are ranked by cosine similarity between $\widehat{V}(x)$ and the stored values, yielding

$$\mathcal{R}_K(x) = \text{TopK}_{V_i:(K_i,V_i)\in\mathcal{M}} \cos\Big(\text{vec}(\widehat{V}(x)), \text{vec}(V_i)\Big). \tag{4}$$

Retrieval reduces the full memory bank to a fixed candidate set $\mathcal{R}_K(x)$. All retrieved memories are then applied through the same frozen Reasoner and conditioning interface defined in Section 3.1, allowing their effects to be compared under aligned reasoning contexts. The fitting procedure for P_{kv} is provided in Appendix A.4.

3.3 Aligned Analysis of Latent-Memory Effects

We next examine how different retrieved memories alter the Reasoner under the **same reasoning context**. Let $\mathcal{R}_K(x) = \{V_1, \ldots, V_K\}$ denote the retrieved memories ordered by retrieval score. We use the highest-ranked memory $V_A = V_1$ to generate an anchor trajectory $\tau_A = (u_1, \ldots, u_T)$.

For each anchor prefix $u_{<t}$, we keep the query, Reasoner, and generated prefix fixed and replace only the latent memory. Under candidate V_j, the teacher-forced next-action proposal and its first behavioral divergence from the anchor are

$$\tilde{u}_{j,t} = \arg\max_{u} \pi_\theta(u \mid x, u_{<t}, P_{\text{out}}(V_j)), \qquad p_j = \min\{t : \tilde{u}_{j,t} \neq u_t\}. \tag{5}$$

If no such position exists, V_j remains behaviorally aligned with the anchor over the observed trajectory. Thus, $(p_j, \tilde{u}_{j,p_j})$ identifies the earliest reasoning state at which candidate V_j induces an action different from the anchor.

The divergence position tells us where two memories first become behaviorally distinct, but not how strongly memory replacement perturbs the current prediction or how decisive the underlying Reasoner is at that state. Let $p_t(V) = \pi_\theta(u_t \mid x, u_{<t}, P_{\text{out}}(V))$, with $p_t(\varnothing)$ denoting the corresponding probability without latent conditioning. We characterize the aligned state using

$$\begin{aligned} D_{j,t} &= |\log p_t(V_j) - \log p_t(V_A)|, \\ I_t &= |\log p_t(V_A) - \log p_t(\varnothing)|, \qquad m_t^0 = \ell_{(1),t}^0 - \ell_{(2),t}^0, \end{aligned} \tag{6}$$

where $\ell_{(1),t}^0$ and $\ell_{(2),t}^0$ are the two largest no-memory next-token logits. $D_{j,t}$ measures the effect of replacing the anchor memory, I_t measures the effect of latent conditioning relative to the no-memory Reasoner, and m_t^0 measures the separation between the Reasoner's top two next-token logits at the same state.

The aligned comparison reveals a consistent behavioral structure: **different latent memories remain aligned over most of a reasoning trajectory and become behaviorally distinct only at a small number of positions**. Retrieved memories therefore do not typically define independent reasoning trajectories from the outset; instead, they more often behave as local interventions on an otherwise largely shared reasoning process. Section 4 provides quantitative evidence for this localized pattern across models, tasks, and training objectives.

Once an alternative action is taken at p_j, the subsequent reasoning prefix changes, and a memory that was previously aligned with the current trajectory may become behaviorally distinct under the newly

reached context. Memory-induced alternatives can therefore emerge progressively as the reasoning state evolves. This observation motivates the recursive procedure below.

3.4 Action Branching: Constructing Memory-Induced Reasoning Trees

Because each divergence changes the subsequent reasoning state, memory-induced alternatives must be reconsidered under the newly reached context. We therefore introduce **Action Branching**. Starting from an anchor generated with the Top-1 memory, it creates a branch only when another memory proposes a different action under the current prefix. After entering the new branch, the remaining memories in the selected divergence group are re-evaluated under the updated reasoning state, while other groups remain pending at the parent node. This allows further reachable alternatives to emerge recursively as the reasoning state changes.

For any current trajectory $\tau = (u_1, \ldots, u_T)$, we apply the aligned comparison from Section 3.3 to the candidate memories associated with that node. Memories that first diverge at the same position and propose the same alternative action define one immediate reachable branch:

$$G_{t,a} = \{V_j \mid p_j = t,\ \tilde{u}_{j,t} = a,\ a \neq u_t\}\,. \tag{7}$$

Each non-empty $G_{t,a}$ therefore corresponds to a distinct outgoing alternative from the current trajectory. We consider groups in increasing order of divergence position, so earlier reachable alternatives are instantiated first.

Multiple memories may induce the same immediate transition and therefore belong to the same group. We use the divergence advantage only to select a representative memory for instantiating that branch:

$$\Delta_{j,t} = \ell_{j,t}(a) - \ell_{j,t}(u_t), \qquad V^\star = \arg\max_{V_j \in G_{t,a}} \Delta_{j,t}, \tag{8}$$

where $\ell_{j,t}$ denotes the next-token logits conditioned on V_j.

The selected memory $V^\star$ instantiates the child trajectory by reusing the reasoning prefix before divergence, taking the alternative action a, and generating only the new suffix:

$$\tau' = (u_{<t}, a, u'_{>t}), \qquad u'_{>t} \sim \pi_\theta\left(\cdot \mid x, u_{<t}, a, P_{\text{out}}(V^\star)\right). \tag{9}$$

Reasoning before the divergence is therefore preserved rather than regenerated.

The remaining memories in $G_{t,a} \setminus \{V^\star\}$ are then re-probed under the child trajectory τ'. Because the reasoning state has changed, a memory that agreed with the representative at the parent may induce a later divergence under the child and create a descendant branch. Other groups defined at the parent remain pending sibling alternatives.

We recursively explore the selected child before returning to pending siblings. Repeating this process until no candidate memory induces a further reachable alternative yields the memory-induced reasoning tree $\mathcal{T}(x; \mathcal{R}_K)$. Tree construction depends only on the next-action differences induced by the retrieved memories under the frozen Reasoner; benchmark-specific evaluation is applied to the generated trajectories only after construction. The complete procedure is given in Algorithm 1.

4 Experiments

We evaluate BranchMem across mathematical reasoning, code generation, and information-seeking tasks. Our experiments examine its overall effectiveness, the behavioral effects induced by different latent memories, the relationship between retrieval rank and downstream memory utility, and the alternatives exposed by Action Branching.

4.1 Experimental Setup

Evaluation and benchmarks. We evaluate BranchMem on five benchmarks spanning mathematical reasoning, code generation, and information seeking: GSM8K (Cobbe et al., 2021), KodCode (Xu et al., 2025b), TriviaQA (Joshi et al., 2017), PopQA (Mallen et al., 2023), and StrategyQA (Geva et al., 2021).

Table 1: Solution coverage (%) across five benchmarks under each method's generation procedure. red and blue cells mark the best and second-best results within each backbone, respectively.

Backbone	Method	KodCode	GSM8K	TriviaQA	PopQA	StrategyQA	Avg.
Llama-3.1-8B	Vanilla	60.35	78.08	63.46	37.84	60.41	60.03
	TokMem	69.80	72.78	68.96	43.45	64.04	63.81
	MemGen-SFT	68.60	73.61	64.68	37.91	69.72	62.90
	MemGen-GRPO	62.60	82.86	68.51	42.12	65.21	64.26
	BRANCHMEM-SFT$_{\text{Top-1}}$	69.75	73.69	68.34	39.80	64.77	63.27
	BRANCHMEM-SFT	**71.50**	77.93	71.16	42.74	73.07	67.28
	BRANCHMEM-GRPO$_{\text{Top-1}}$	63.00	83.62	68.64	41.27	65.93	64.49
	BRANCHMEM-GRPO	67.70	**87.71**	**75.65**	**47.16**	**83.41**	**72.33**
Llama-3.2-3B	Vanilla	46.30	77.78	60.01	32.02	57.35	54.69
	TokMem	65.75	66.94	65.58	39.87	59.38	59.50
	MemGen-SFT	64.90	66.18	60.89	39.59	61.14	58.54
	MemGen-GRPO	57.10	77.25	59.34	32.73	55.02	56.29
	BRANCHMEM-SFT$_{\text{Top-1}}$	67.25	65.57	56.19	37.35	62.29	57.73
	BRANCHMEM-SFT	**69.10**	69.90	65.28	**41.69**	**71.03**	63.40
	BRANCHMEM-GRPO$_{\text{Top-1}}$	58.60	76.72	60.67	33.14	56.47	57.12
	BRANCHMEM-GRPO	62.35	**82.25**	**69.59**	38.54	**71.03**	**64.75**

Baselines. We compare BRANCHMEM with the frozen Reasoner without external memory (**Vanilla**), TokMem (Wu et al., 2026) as an explicit latent-memory baseline, and MemGen (Zhang et al., 2026) as a generative latent-memory baseline. Where applicable, we report training-matched SFT and GRPO variants. We additionally report **BRANCHMEM-Top-1**, which uses the same memory interface and retrieval procedure but generates only from the highest-ranked retrieved memory.

Implementation and evaluation. We use Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Llama-3.2-3B-Instruct (Meta AI, 2024) as frozen Reasoners and Llama-3.2-1B as the compressor. Each Reasoner uses a separately trained memory interface, with BRANCHMEM-SFT and BRANCHMEM-GRPO corresponding to the two training objectives in Section 3.2. Unless otherwise specified, retrieval returns $K = 8$ candidate memories. We report **solution coverage**, counting a query as covered if at least one trajectory generated by the corresponding procedure is correct under the benchmark-specific evaluator. Full data, training, memory-interface, retrieval, and evaluation configurations are provided in Appendix A.

4.2 MAIN RESULTS

RQ1: How effectively does BRANCHMEM leverage persistent latent memories across tasks and Reasoners? Table 1 summarizes the overall results. BRANCHMEM-SFT/GRPO achieve five-task average solution coverages of $67.28/72.33$ on Llama-3.1-8B and $63.40/64.75$ on Llama-3.2-3B, respectively. Across individual tasks, the best BRANCHMEM configuration achieves the highest or tied-highest result under both Reasoners. Relative to the corresponding Top-1 variants, full BRANCHMEM further improves average coverage by $4.01/7.84$ **points** on Llama-3.1-8B and $5.67/7.63$ **points** on Llama-3.2-3B for SFT/GRPO, respectively. This gap shows that useful reasoning guidance remains in the retrieved set beyond the initially selected memory.

Beyond overall performance, we further examine how different retrieved memories affect the frozen Reasoner.

RQ2: Does the localized intervention pattern hold across settings? Figure 3 and Table 2 evaluate the aligned analysis from Section 3.3 across GSM8K and KodCode, two frozen Reasoners, and both SFT and GRPO training. In Table 2, subscripts b and s denote statistics averaged over first-divergence and shared-prefix positions, respectively; each ratio compares the corresponding averages. All eight settings exhibit the same structure: **different latent memories remain behaviorally close over the shared prefix, while their effects concentrate around the first divergence.** Candidate effect at branch points reaches 53.9–$959.0\times$ its shared-prefix level, while latent insertion effect increases by 34.8–$496.6\times$. Different memories therefore do not continuously steer generation toward separate

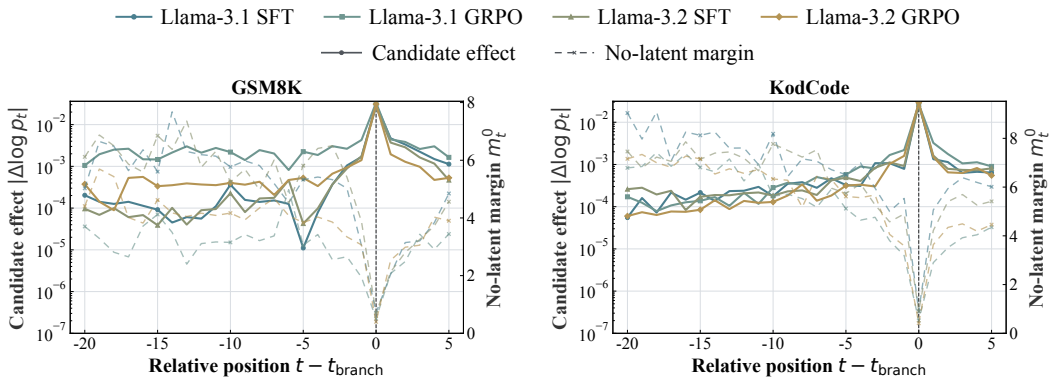


Figure 3: Branch-aligned latent-memory effects on GSM8K and KodCode.

trajectories from the outset; instead, they remain similar over most of the reasoning process and become distinct at a small number of next-action decisions.

Table 2: Branch-point statistics for aligned latent-memory interventions.

Backbone	Training	Task	D_b/D_s	I_b/I_s	m_b^0/m_s^0	Low-margin AUC
Llama-3.1-8B	SFT	GSM8K	314.9×	64.9×	12.5%	0.840
	SFT	KodCode	463.6×	352.8×	10.0%	0.912
	GRPO	GSM8K	53.9×	34.8×	10.3%	0.893
	GRPO	KodCode	542.9×	396.6×	6.5%	0.928
Llama-3.2-3B	SFT	GSM8K	371.3×	59.7×	12.5%	0.844
	SFT	KodCode	371.6×	330.1×	10.8%	0.911
	GRPO	GSM8K	216.3×	66.5×	7.5%	0.915
	GRPO	KodCode	959.0×	496.6×	5.2%	0.939

Branch points also coincide with low-margin states of the frozen Reasoner. At branch points, the no-latent Top-1/Top-2 margin falls to only 5.2–12.5% of its shared-prefix level, and low margin identifies first-divergence events with ROC–AUC 0.840–0.939. The no-latent and anchor actions disagree at only 3.5–17.2% of shared positions, compared with 51.3–69.8% of branch points. Thus, memories tend to remain aligned when the Reasoner has a large local decision margin and diverge more often when that margin is small.

The resulting pattern is both localized and context dependent: different memories share most of the reasoning process and become behaviorally distinct primarily near low-margin decisions, with their effects changing as the reasoning state evolves.

RQ3: Is memory utility determined at retrieval time or expressed during reasoning? We further examine the relationship between retrieval score and downstream memory utility using **rescue cases**, where the Top-1 memory fails but at least one candidate among ranks 2–8 produces a successful trajectory. This isolates whether the initial ranking can identify a useful candidate once it has already been recalled.

Across the eight TriviaQA and PopQA model–training settings, the conditional pairwise AUC of retrieval score for distinguishing successful from unsuccessful lower-ranked candidates is only 0.422–0.552, with a mean of 0.498 and a median of 0.507. Even when a successful candidate is already present in the retrieved Top-8 set, its initial similarity score therefore provides little reliable information about downstream success.

The important distinction is between **candidate recall** and **within-set utility**. Retrieval narrows the full memory bank to a plausible candidate set, but query-level similarity does not fully describe how a memory will behave under later reasoning states. This is consistent with the state-dependent behavior

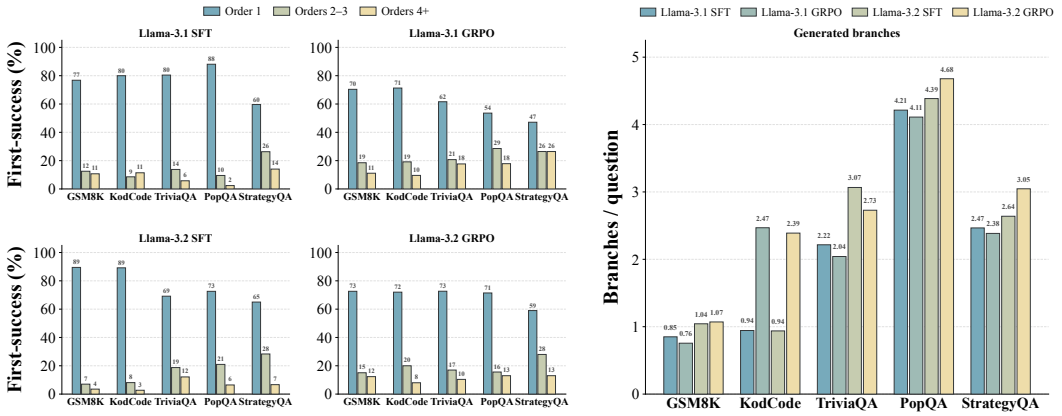


Figure 4: First-success positions in completed Action Branching trees. Left: first-success traversal order. Right: mean branches generated before first success; correct anchors count as zero.

observed in RQ2: a lower-ranked memory may remain aligned initially yet become behaviorally distinct after the prefix changes.

Retrieval is therefore useful for determining which memories should be considered, while downstream utility is expressed more directly in the reasoning context where those memories act.

RQ4: How does Action Branching exploit memory-induced alternatives? We examine where successful alternatives appear in the Action Branching tree. For each query, we keep the frozen Reasoner, memory checkpoint, and retrieved Top-8 set fixed and measure **where the first successful trajectory appears in the completed tree traversal**.

Figure 4 shows that, among observed first successes on generated branches, **Order 1 accounts for** 47–89%, while approximately 73–98% appear within the first three branches. Useful alternatives therefore often appear early in the traversal, although their locations vary across tasks. Under Llama-3.1 GRPO on StrategyQA, for example, 26% of first successes occur at Order 4 or later.

The number of generated branches preceding the first successful trajectory shows a similar task-dependent pattern. Counting an already-correct anchor as zero, the mean ranges from 0.76–1.07 on GSM8K to 4.11–4.68 on PopQA. Useful alternatives tend to appear earlier for mathematical reasoning, whereas information-seeking tasks more often require deeper exploration.

Action Branching reuses the shared prefix and generates new continuations only after memory-induced divergence, exposing useful alternatives beyond the Top-1 memory without restarting reasoning from the query.

5 Conclusion

We introduced BranchMem, which stores successful experience as independently retrievable latent memories for a frozen Reasoner. Our aligned analysis shows that different memories share most reasoning states and diverge primarily at sparse critical decisions, motivating Action Branching to reuse shared reasoning and expand memory-induced alternatives. Across two Reasoners and five benchmarks, our experiments show that persistent latent memories can guide new tasks and that Action Branching turns their localized effects into useful reasoning alternatives.

Ethics Statement

This work uses publicly available benchmarks and does not involve human subjects, private data, or safety-sensitive deployment settings.

Reproducibility Statement

Appendix A provides the full implementation, training, data, retrieval, and evaluation configurations, together with the complete Action Branching procedure in Algorithm 1. Appendix B reports additional analyses of latent-memory interventions, branching behavior, and downstream memory utility.

AI Use Statement

Generative AI tools were used to assist with language polishing and structural editing during manuscript preparation. All technical formulations, experimental design, implementation, analyses, citations, and scientific claims were independently reviewed and verified by the authors, who take full responsibility for the final content of this work.

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of Thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 17682–17690, 2024. URL https://doi.org/10.1609/aaai.v38i16.29720.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. xRAG: Extreme context compression for retrieval-augmented generation with one token. In *Advances in Neural Information Processing Systems*, volume 37, pp. 109487–109516, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/c5cf13bfd3762821ef7607e63ee90075-Abstract.html.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, 2023. URL https://doi.org/10.18653/v1/2023.emnlp-main.232.

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

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, 2026. URL https://doi.org/10.18653/v1/2026.findings-acl.866.

Tao Feng, Chongrui Ye, Tianyang Luo, Jingjun Xu, Xueqiang Xu, Haozhen Zhang, Ge Liu, and Jiaxuan You. ElasticMem: Latent memory as a learnable resource for LLM agents. *arXiv preprint arXiv:2605.30690*, 2026. URL https://arxiv.org/abs/2605.30690.

Muxin Fu, Xiangyuan Xue, Yafu Li, Zefeng He, Siyuan Huang, Xiaoye Qu, Yu Cheng, and Yang Yang. LatentMem: Customizing latent memory for multi-agent systems. *arXiv preprint arXiv:2602.03036*, 2026. URL https://arxiv.org/abs/2602.03036.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=uREj4ZuGJE.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021. URL https://doi.org/10.1162/tacl_a_00370.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Itxz7S4Ip3.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 1601–1611, 2017. URL https://doi.org/10.18653/v1/P17-1147.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 9802–9822, 2023. URL https://doi.org/10.18653/v1/2023.acl-long.546.

Meta AI. Llama 3.2: Revolutionizing edge AI and vision with open, customizable models, 2024. URL https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, volume 36, pp. 19327–19352, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/3d77c6dcc7f143aa2154e7f4d5e22d68-Abstract.html.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023. URL https://doi.org/10.1145/3586183.3606763.

Ignacio Sastre and Aiala Rosá. Memory tokens: Large language models can generate reversible sentence embeddings. In *Proceedings of the First Workshop on Large Language Model Memorization*, pp. 183–189, 2025. URL https://doi.org/10.18653/v1/2025.l2m2-1.14.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. AgentBank: Towards generalized LLM agents via fine-tuning on 50000+ interaction trajectories. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 2124–2141, 2024. URL https://doi.org/10.18653/v1/2024.findings-emnlp.116.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a. URL https://arxiv.org/abs/2305.16291.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=1PL1NIMMrw.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Zijun Wu, Yongchang Hao, and Lili Mou. TokMem: One-token procedural memory for large language models. In *International Conference on Learning Representations*, pp. 32460–32480, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/372cb7805eaccb2b7eed641271a30eec-Abstract-Conference.html.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. SoftCoT: Soft chain-of-thought for efficient reasoning with LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 23336–23351, 2025a. URL https://doi.org/10.18653/v1/2025.acl-long.1137.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6980–7008, 2025b. URL https://doi.org/10.18653/v1/2025.findings-acl.365.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract.html.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. MemGen: Weaving generative latent memory for self-evolving agents. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=vI56m4Iu4e.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 19632–19642, 2024. URL https://doi.org/10.1609/aaai.v38i17.29936.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 19724–19731, 2024. URL https://doi.org/10.1609/aaai.v38i17.29946.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

A Implementation Details

A.1 Memory Interface Settings

Each Reasoner uses a separately trained memory interface with a Llama-3.2-1B compressor. The compressor is adapted with LoRA (Hu et al., 2022), together with the latent slots, normalization and scale parameters, and cross-model projections. It produces $S = 8$ latent tokens with dimensionality $d_m = 2048$.

For each successful experience (x_i, τ_i^+), the shared compressor encodes the task input x_i into a query-side key K_i and the successful trajectory τ_i^+ into a latent value V_i. The resulting pair (K_i, V_i)

is stored in the memory bank. At inference time, the same compressor encodes the new query x into K_x, which is mapped toward the latent-value space by P_{kv} for retrieval as described in Appendix A.4.

For each query, BRANCHMEM retrieves $K = 8$ candidate memories and uses the highest-ranked candidate to initialize the anchor trajectory. The retrieved latent-memory tokens are inserted immediately after the query tokens and before response generation.

A.2 TRAINING AND DATA CONFIGURATIONS

The Reasoner remains frozen throughout training, while the memory interface is optimized using either SFT or GRPO. Both training regimes use the same memory construction and conditioning mechanism and differ only in the supervision signal.

Training objectives. Let ϕ denote the trainable memory-interface parameters and define the induced latent-conditioned policy as

$$\pi_\phi(u_t \mid x, u_{<t}; \tau^+) = \pi_\theta\big(u_t \mid x, u_{<t}, P_{\text{out},\phi}(C_\phi(\tau^+))\big)\,,$$

where the Reasoner parameters θ remain fixed. For SFT, we teacher-force the successful trajectory and minimize its negative log-likelihood,

$$\mathcal{L}_{\text{SFT}}(\phi) = -\mathbb{E}_{(x,\tau^+)\sim\mathcal{H}}\left[\sum_{t=1}^{|\tau^+|} \log \pi_\phi\big(\tau_t^+ \mid x, \tau_{<t}^+; \tau^+\big)\right].$$

For GRPO, we sample a group of latent-conditioned completions $\{o_i\}_{i=1}^G$ for the same input and compute group-relative advantages $\hat{A}_i$ from their rewards. With

$$r_{i,t}(\phi) = \frac{\pi_\phi(o_{i,t} \mid x, o_{i,<t}; \tau^+)}{\pi_{\phi_{\text{old}}}(o_{i,t} \mid x, o_{i,<t}; \tau^+)},$$

let

$$\bar{r}_{i,t}(\phi) = \text{clip}\left(r_{i,t}(\phi), 1-\epsilon, 1+\epsilon\right),$$

and maximize the clipped GRPO objective

$$\mathcal{J}_{\text{GRPO}}(\phi) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_t \min\Big(r_{i,t}(\phi)\hat{A}_i, \bar{r}_{i,t}(\phi)\hat{A}_i\Big) - \beta D_{\text{KL}}\right].$$

Here, $\hat{A}_i$ is computed relative to the other rollouts in the same group, and D_{KL} denotes the standard GRPO KL regularizer. Gradients are applied only to the memory interface; the Reasoner remains frozen.

Training hyperparameters are summarized in Table 3. Memory banks are constructed from successful trajectories in the corresponding training split, and evaluation examples are not included in the bank.

Table 4 summarizes the training sources and the numbers of evaluated queries. PopQA uses the interface trained on TriviaQA, without PopQA-specific training data; the TriviaQA and StrategyQA training trajectories come from AgentBank (Song et al., 2024).

A.3 EVALUATION PROTOCOL

Action Branching constructs the memory-induced reasoning tree from the fixed retrieved candidate set following Section 3.4. Once construction is complete, the generated trajectories are evaluated using the corresponding benchmark-specific evaluator.

We report **solution coverage**, counting a query as covered if at least one generated trajectory is correct. BRANCHMEM-Top-1 is evaluated using the single trajectory generated from the highest-ranked retrieved memory.

For analyses that compare different procedures over the retrieved set, all procedures use the same fixed Top-K candidate set for each query.

Table 3: Training configurations.

Setting	Hyperparameter
Training (SFT)	Train batch size: 4 Learning rate: 10^{-5} Epochs: 2 Warmup ratio: 0.1 Optimizer: AdamW Scheduler: cosine
Training (GRPO)	Rollout batch size: 8 Train batch size: 8 Epochs: 1 KL coefficient β: 0.05 Iterations: 1 Learning rate: 10^{-5} Warmup ratio: 0.1 Optimizer: AdamW Scheduler: cosine
LoRA	Rank r: 16 Scaling α: 32 Dropout: 0 Bias: none Task type: `CAUSAL_LM`

Table 4: Dataset sources and splits. Train size denotes the source training pool; test size denotes queries evaluated in this work.

Dataset	Train size	Test size	Note
GSM8K	7,473	1,319	Official GSM8K training split
KodCode	8,000	2,000	KodCode-Light-RL-10K
TriviaQA	4,134	7,993	AgentBank-TriviaQA training
PopQA	–	1,427	Trained on TriviaQA
StrategyQA	1,267	687	AgentBank-StrategyQA training

A.4 Retrieval Projection

Given the paired query keys and latent values (K_i, V_i) in the training memory bank, we fit a slot-wise projection P_{kv} from the query-key space to the latent-value space. For each latent slot s,

$$P_s^* = \arg\min_{P_s} \sum_{i \in \mathcal{I}_{\text{fit}}} \left\| \mu_s^V + P_s\big(K_{i,s} - \mu_s^K\big) - V_{i,s} \right\|_2^2 + \lambda \|P_s\|_F^2. \tag{10}$$

Here, μ_s^K and μ_s^V denote the corresponding means over the fitting set. The regression is solved in a low-rank PCA subspace of the query keys. The slot-wise projections jointly define P_{kv} in Equation 4.

We select the PCA rank and ridge coefficient using held-out centered cosine similarity, and then refit the selected configuration on all training memory pairs. Table 5 summarizes the retrieval-projection configuration.

A.5 Action Branching Algorithm

Algorithm 1 gives the complete Action Branching procedure described in Section 3.4. FirstDivergence applies the aligned comparison from Section 3.3 and returns the earliest position at which a candidate memory proposes an action different from the current trajectory, together with the alternative action and its divergence advantage from Equation 8, where $\Delta_j \equiv \Delta_{j,p_j}$ denotes the advantage at the first-divergence position.

Candidates that first diverge at the same position and propose the same action are grouped into one immediate alternative. Within each group, the memory with the largest divergence advantage is

Table 5: Retrieval projection configuration.

Setting	Configuration
Fitting / validation split	90% / 10% of training memory pairs
PCA rank	$\{8, 16, 32, 64, 128\}$
Ridge coefficient λ	$\{10^{-3}, 10^{-2}, 10^{-1}, 1\}$
Selection criterion	Held-out centered cosine similarity
Final fitting	Refit on all training memory pairs

Algorithm 1: Action Branching Tree Construction

Input: Query x; retrieved set $\mathcal{R}_K(x) = \{V_1, \ldots, V_K\}$; frozen Reasoner π_θ
Output: Memory-induced reasoning tree $\mathcal{T}(x; \mathcal{R}_K)$

$\tau^{(0)} \leftarrow$ GENERATE(x, V_1);
$\mathcal{T} \leftarrow \{\tau^{(0)}\}$;
Function `Explore` (τ, U) **:**
  if $U = \emptyset$ **then**
    return;
  $\mathcal{G} \leftarrow \emptyset$;
  foreach $V_j \in U$ **do**
    $(p_j, a_j, \Delta_j) \leftarrow$ FIRSTDIVERGENCE(τ, V_j);
    if *a reachable divergence exists* **then**
      add V_j to group G_{p_j, a_j} in $\mathcal{G}$;
  order groups in $\mathcal{G}$ by increasing divergence position;
  foreach $G_{t,a} \in \mathcal{G}$ **do**
    $V^* \leftarrow \arg\max_{V_j \in G_{t,a}} \Delta_j$;
    $\tau' \leftarrow$ BRANCHGENERATE$(\tau_{<t}, a, V^*)$;
    add τ' as a child of τ in $\mathcal{T}$;
    $U' \leftarrow G_{t,a} \setminus \{V^*\}$;
    `Explore`(τ', U');
`Explore`$(\tau^{(0)}, \{V_2, \ldots, V_K\})$;
return $\mathcal{T}$;

used to instantiate the branch. BRANCHGENERATE reuses the prefix before the divergence, takes the selected alternative action, and generates only the new suffix. The remaining memories in the same group are then re-evaluated under the resulting child trajectory, allowing further divergences to emerge recursively. Other groups remain sibling alternatives at the current node.

B Additional Analyses

B.1 Residual Interventions

We further examine whether memory-induced branch behavior can be explained by simple geometric differences in latent space. Let $R_i = V_i - V_1$ denote the residual of candidate V_i relative to the anchor memory. Residual cosine similarity and norm provide only weak separation between candidates that induce the same versus different actions, suggesting that these coarse geometric statistics do not fully characterize branch behavior.

We therefore intervene directly on the candidate residual:

$$V_i^{(\alpha)} = V_1 + \alpha(V_i - V_1), \qquad \alpha \in \{0, 0.25, 0.5, 1\}.$$

Figure 5 reports this intervention on Llama-3.2-3B SFT GSM8K. With $\alpha = 0.25$ and 0.5, the intervention recovers 44.7% and 46.3% of the original branch events, respectively. Increasing the residual scale does not monotonically preserve the original branch events. Moreover, shuffling

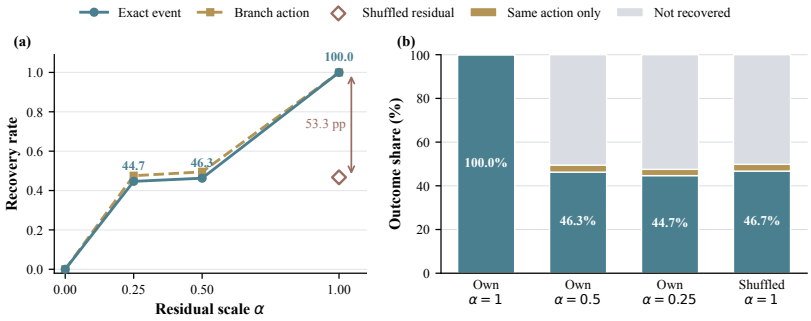


Figure 5: Residual interventions on Llama-3.2-3B SFT GSM8K. Left: branch-event recovery under residual scaling. Right: outcomes under scaled and shuffled residuals.

residuals across candidates changes which events are recovered despite producing a similar aggregate recovery rate.

In this diagnostic, branch-inducing behavior is therefore not explained by residual magnitude alone. The identity of the residual and the reasoning context in which it is applied both matter for whether the original branch behavior is preserved.

B.2 Downstream Memory Utility and Retrieval Ranking

We further examine whether the pre-generation retrieval score predicts which lower-ranked candidate will become useful downstream. We focus on **rescue cases**, where the Top-1 memory fails but at least one candidate among ranks 2–8 produces a successful trajectory.

For each rescue query, we compute a conditional pairwise AUC over ranks 2–8, measuring how often a successful candidate receives a higher retrieval score than an unsuccessful candidate, with half credit for ties. We additionally report the prevalence of rescue cases and the average rank of the first successful non-anchor candidate.

As shown in Table 6, rescue cases occur in 2.81–9.08% of the evaluated TriviaQA and PopQA settings, while the first successful non-anchor candidate appears at an average rank of 3.12–4.19. The conditional AUC ranges from 0.422 **to** 0.552, with a mean of 0.498 and a median of 0.507. Thus, once a useful memory has already entered the retrieved Top-8 set, its initial retrieval score provides little consistent separation between successful and unsuccessful lower-ranked candidates.

These results highlight the distinction between **candidate recall** and **within-set utility**. Retrieval can place useful memories in the candidate set, but static query-level similarity does not fully determine which candidate will become useful downstream. This is consistent with the state-dependent behavior analyzed in the main text, where a memory's effect depends on the reasoning context in which it is applied.

Table 6: Retrieval-utility diagnostic on rescue cases.

Backbone	Training	Task	Rescue (%)	First rescue rank	Conditional AUC
Llama-3.1-8B	SFT	TriviaQA	2.81	3.38	0.514
	SFT	PopQA	2.94	3.12	0.552
	GRPO	TriviaQA	7.01	4.19	0.463
	GRPO	PopQA	5.89	4.08	0.422
Llama-3.2-3B	SFT	TriviaQA	9.08	3.57	0.510
	SFT	PopQA	4.34	3.63	0.503
	GRPO	TriviaQA	8.92	3.74	0.483
	GRPO	PopQA	5.40	3.34	0.536

B.3 Independent Top-8 vs. Action Branching at $B = 1$

We compare Action Branching under a one-branch budget ($B = 1$) with eight independent memory-conditioned completions generated from the same retrieved Top-8 set. Independent Top-8 provides a pass@8-style coverage reference: a query is covered if any of the eight independently generated trajectories is correct.

Table 7 reports solution coverage together with autoregressive generation counts. Across the 20 settings, Action Branching at $B = 1$ is within 0.62–8.59 **percentage points** of Independent Top-8 coverage, with a mean gap of 2.65 **points**. The largest gap occurs on StrategyQA with Llama-3.1-8B GRPO, while several settings remain within roughly one percentage point of the full independent-generation reference.

For token accounting, **Top-8 tok/q** is the sum of the generated lengths of the eight independent completions, whereas **AB tok/q** counts the anchor and newly generated branch suffix. Token reduction is

$$1 - \frac{\text{AB tok/q}}{\text{Top-8 tok/q}}.$$

Under this restricted $B = 1$ setting, Action Branching reduces autoregressive generation by 81.44–85.76% across the evaluated configurations. This comparison measures how much Independent Top-8 coverage can be recovered using the anchor and only the first reachable branch.

Table 7: Independent Top-8 versus Action Branching with one reachable branch ($B = 1$).

Group	Dataset	Top-8 Coverage	AB Coverage ($B = 1$)	Top-8 tok/q	AB tok/q	Token reduction
Llama-3.1-8B SFT	GSM8K	78.09%	76.95%	1116.3	164.2	85.29%
	KodCode	71.95%	71.15%	1092.7	156.2	85.71%
	PopQA	43.17%	42.40%	777.1	118.6	84.74%
	StrategyQA	74.23%	69.72%	1456.0	230.8	84.15%
	TriviaQA	71.23%	70.61%	815.3	116.1	85.76%
Llama-3.1-8B GRPO	GSM8K	88.02%	86.50%	1258.2	180.7	85.64%
	KodCode	68.00%	66.35%	3350.4	545.8	83.71%
	PopQA	47.72%	44.43%	917.9	170.4	81.44%
	StrategyQA	86.46%	77.87%	2285.1	382.1	83.28%
	TriviaQA	75.77%	72.96%	916.0	156.7	82.89%
Llama-3.2-3B SFT	GSM8K	70.65%	69.45%	1111.6	164.7	85.18%
	KodCode	69.65%	68.90%	1083.2	155.8	85.62%
	PopQA	42.40%	40.50%	816.4	136.6	83.27%
	StrategyQA	72.19%	67.98%	872.5	137.3	84.26%
	TriviaQA	65.45%	62.48%	1053.0	172.1	83.66%
Llama-3.2-3B GRPO	GSM8K	83.54%	80.74%	1390.1	207.0	85.11%
	KodCode	62.85%	61.30%	2424.7	389.8	83.92%
	PopQA	39.59%	37.00%	607.1	102.7	83.08%
	StrategyQA	71.61%	65.07%	1906.9	323.3	83.05%
	TriviaQA	69.97%	67.16%	766.7	125.1	83.68%

B.4 KodCode Case Study

The following example juxtaposes an abridged task input, the two latent-conditioned paths, selected generated code, and the resulting behavior on a test input from the prompt. The latent values are shown symbolically rather than decoded into text.

KodCode query: Median of Two Sorted Arrays

MODEL INPUT (ABRIDGED)
Write `find_median_sorted_arrays(nums1, nums2)` using binary search in $O(\log \min(n, m))$ time. For `[1, 2]` and `[3, 4]`, the expected median is 2.5.

LATENT-CONDITIONED INPUT
Top-1 anchor: $x \mid P_{\text{out}}(V_A) \mid u_{<p} \mid u_p \cdots$
First branch: $x \mid P_{\text{out}}(V^\star) \mid u_{<p} \mid a \cdots$

Here $V^\star \neq V_A$ and $a \neq u_p$. One latent memory is inserted after the query on each path; the response prefix $u_{<p}$ is reused at the branch.

MODEL OUTPUT: SHARED PREFIX (selected lines)

```
left, right = 0, len(nums1) - 1
while True:
  i = (left + right) // 2
  j = half_length - i - 2
```

MODEL OUTPUT: TOP-1 ANCHOR

```
if i < left:  i = left
if j < 0:  j = 0
```

Evaluation: incorrect. On the example above, the anchor reaches $i = 1$ and then attempts to access `nums1[2]`, which is out of bounds.

MODEL OUTPUT: FIRST EXPANDED BRANCH

```
nums1_left = nums1[i] if i >= 0 else float('-infinity')
if nums1_left <= nums2_right and nums2_left <= nums1_right:
```

Evaluation: correct. This continuation uses infinite sentinels at missing boundaries and returns 2.5 on the same example.