MEMCOLLAB: CROSS-MODEL MEMORY COLLABORATION VIA CONTRASTIVE TRAJECTORY DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models increasingly rely on memory mechanisms to reuse knowledge from past problem-solving experiences. However, existing methods typically construct memory from the experiences of a single LLM and reuse it with the same underlying model. As a result, stored memories may entangle transferable task knowledge with model-specific solution strategies that are less effective when reused by other LLMs. This coupling becomes problematic when memory is shared across heterogeneous LLMs with different sizes, architectures, or reasoning behaviors, raising a key question: *can we construct memory that transfers reliably across different LLMs?* We find that naive cross-model memory transfer can degrade performance because model-specific solution strategies encoded in memory may not generalize to other LLMs. To address this challenge, we propose MEMCOLLAB, a collaborative memory framework that constructs shared cross-model memory by contrasting reasoning trajectories produced by different LLMs on the same problem. By contrasting how different LLMs solve the same task, MEMCOLLAB retains solution principles that are consistently useful across models and removes model-specific steps that do not transfer, while suppressing less transferable, model-specific solution patterns. We further introduce a task-aware retrieval mechanism that conditions memory access on task category, ensuring that only relevant constraints are retrieved at inference time. Experiments on mathematical reasoning and code generation benchmarks show that MEMCOLLAB consistently improves accuracy and inference-time efficiency across LLMs of different sizes and model families. These results demonstrate that collaboratively constructed memory can serve as a shared reasoning resource across different LLMs.

1 INTRODUCTION

Large language models (LLMs) are increasingly used to solve complex tasks involving multi-step reasoning, code execution, and interaction with external tools. Despite their strong capabilities, they do not naturally retain persistent knowledge from prior problem-solving experiences, and each inference is constrained by a finite context window (Laban et al., 2025; Kang et al., 2025). As a result, LLMs may repeatedly explore similar solution strategies, reproduce recurring errors, and fail to reuse useful feedback from previous attempts (Pink et al., 2025; Anthropic, 2025). Memory mechanisms address these limitations by storing and retrieving reusable information from past experiences to support future inference (Zhong et al., 2024; Xu et al., 2025).

Recent studies show that memory augmentation can improve LLM reasoning by reusing knowledge distilled from past problem-solving experiences. For example, BoT (Yang et al., 2024) extracts reasoning templates from previous solutions and retrieves them during inference. However, most existing approaches construct memory for a particular LLM, deriving it from that model's own reasoning traces and reusing it with the same model. Consequently, the stored memory may encode solution strategies that are closely tied to the model that generated them, limiting their transferability to other LLMs. This limitation becomes important when memory is reused across models with different sizes, architectures, or specializations. This raises a natural question: *can a single memory system be effectively shared across heterogeneous LLMs?*

A naive approach is to directly transfer memory constructed from one LLM to another. In Figure 1, we study how external memory affects model performance by comparing a no-memory baseline

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.

with memory constructed from the target LLM's own trajectories and memory constructed from trajectories generated by a different LLM. The results show that directly reusing memory across models can underperform memory distilled from the target model's own trajectories. This suggests that memory may encode model-specific solution strategies and heuristics that do not transfer well across LLMs, rather than purely capturing generalizable task knowledge. This observation motivates a key question: can we construct memory that preserves reusable task knowledge while reducing model-specific solution patterns?

In this paper, we propose MEMCOLLAB, a framework for constructing *shared cross-model memory* that can be reused across heterogeneous LLMs. Unlike memory distilled from a single model's trajectories, MEMCOLLAB contrasts trajectories generated by different LLMs on the same task to identify reasoning guidance that is consistently useful across models and to remove model-specific solution choices that do not transfer. The resulting memory captures reusable reasoning constraints, tool-usage guidance, and error-avoidance patterns, and can be shared across LLMs with different sizes, architectures, or model families. To further improve retrieval precision, we introduce a task-aware retrieval mechanism that conditions memory access on task category, ensuring that only task-relevant memory is retrieved at inference time. We summarize our contributions below:

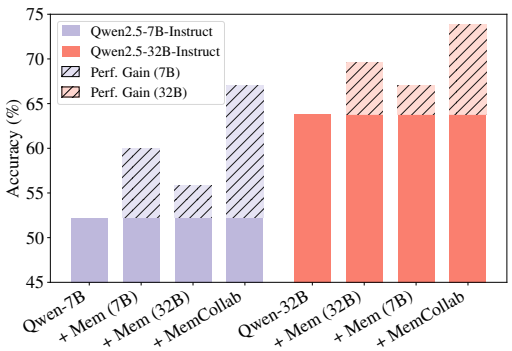

Figure 1: Accuracy on MATH500 with memory constructed from different sources.

- We identify and study the problem of constructing shared memory across heterogeneous LLMs with different model sizes, architectures, and families, a setting that remains underexplored in existing memory-augmented reasoning methods.
- We introduce a cross-model memory collaboration framework that contrasts reasoning traces from different LLMs on the same task to distill transferable solution principles and shared error-avoidance patterns into reusable memory.
- Extensive experiments on mathematical reasoning and code generation benchmarks show that MEMCOLLAB consistently improves performance across heterogeneous LLMs while also reducing inference-time overhead through compact and task-relevant memory.

2 METHODOLOGY

We propose MEMCOLLAB, a cross-model memory framework that improves LLM reasoning by distilling reusable guidance from heterogeneous model trajectories. It consists of two stages: (1) **contrast-derived memory construction**, which extracts transferable reasoning and error-avoidance constraints across models; and (2) **task-aware retrieval**, which retrieves relevant memory to guide inference. Figure 2 provides an overview.

2.1 PROBLEM SETUP AND NOTATION

Let $\mathcal{F} = \{f_i\}_{i=1}^{N}$ denote a set of heterogeneous LLMs with different reasoning capacities, such as different model sizes, architectures, or training procedures. Given a task instance x, each model f_i independently produces a multi-step reasoning trajectory

$$\tau_i^{(x)} = (r_{i,1}, r_{i,2}, \ldots, r_{i,T_i}), \tag{1}$$

where each element $r_{i,t}$ represents an intermediate reasoning step from model f_i, which may incorporate natural language reasoning, code, and code execution feedback. We assume access to a training set of tasks $\mathcal{D}$, each with a ground-truth solution or executable verifier. Our goal is to construct a shared memory bank $\mathcal{M}$ that can be retrieved and reused by multiple heterogeneous LLMs at inference time.

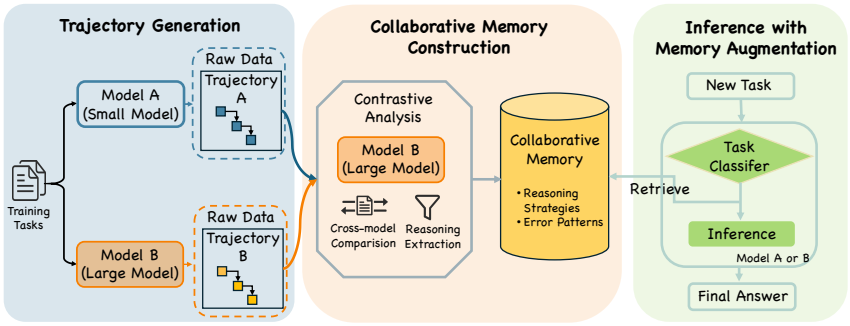

Figure 2: Framework of MEMCOLLAB. *Left*: Using different models to generate trajectory pairs for the same task. *Middle*: Contrasting the trajectory pairs to construct the memory. *Right*: Augmenting inference-time generation with retrieved memory.

2.2 CONTRAST-DERIVED MEMORY CONSTRUCTION

Algorithm 1 summarizes the detailed procedure for memory bank construction, with major steps explained as below.

Trajectory Pairing and Preference Selection. For each task $x \in \mathcal{D}$, we collect N trajectories $\{\tau_i\}_{i=1}^{N}$ by executing multiple heterogeneous LLMs on the same input. Let $i^\star$ denote the strongest model, and let $\mathcal{I}(\cdot)$ be an indicator function that evaluates whether a trajectory is correct with respect to the task ground truth, such as by checking the final answer or execution result. We select the preferred trajectory as either a correctly executed trajectory or the trajectory from the strongest model, denoted as:

$$\tau^+ = \begin{cases} \tau \sim \mathrm{Unif}(\{\tau_i \mid \mathcal{I}(\tau_i) = 1\}), & \text{if } \mathcal{I}(\tau_{i^\star}) = 0 \text{ and } \exists i \neq i^\star,\ \mathcal{I}(\tau_i) = 1, \\ \tau_{i^\star}, & \text{otherwise.} \end{cases} \tag{2}$$

The remaining trajectories are treated as unpreferred trajectories:

$$\mathcal{T}^- = \{\tau_i\}_{i=1}^{N} \setminus \{\tau^+\}. \tag{3}$$

This formulation generalizes pairwise trajectory selection to the multi-trajectory setting. It allows the preferred trajectory to originate from any model that successfully solves the task, rather than always from the strongest model, making the preference signal robust to failures of the strongest model. The unpreferred set provides contrastive references for identifying reasoning errors and model-specific solution patterns.

Contrastive Trajectory Analysis. Rather than performing step-by-step comparison, MEMCOLLAB summarizes trajectory differences at the structural reasoning level, including high-level solution strategies, planning decisions, and the interpretation of tool feedback. Given the preferred trajectory τ^+ and the set of unpreferred trajectories $\mathcal{T}^-$, we contrast τ^+ with each $\tau^- \in \mathcal{T}^-$ independently and summarize their differences into structured guidance. The summarization is performed by the strongest model $f_{i^\star}$ or a designated summarizer model; see Appendix E for the detailed prompt.

For each contrastive pair (τ^-, τ^+), we distill two types of information: (1) **violation patterns** v_k, which characterize reasoning failures observed in τ^-, such as premature computation, incorrect symbolic mapping, or misinterpretation of tool feedback; and (2) **reasoning invariants** i_k, which capture reasoning principles present in τ^+ but absent or violated in τ^-, such as structural principles, planning constraints, or calibration rules needed to avoid the corresponding failure. Analyzing each contrastive pair yields a discrepancy set:

$$\Delta(\tau^-, \tau^+) = \{(v_k, i_k)\}_{k=1}^{K}. \tag{4}$$

Memory Bank Construction. Given an extracted discrepancy (v_k, i_k), each contrastive pair is converted into a normative reasoning constraint:

$$m_k = (\text{enforce } i_k;\ \text{avoid } v_k). \tag{5}$$

Each extracted entry is further tagged with model-identity labels indicating the LLMs that produced the corresponding trajectories. These labels do not create separate memory banks; instead, they

support model-aware retrieval from a shared collaboratively constructed memory bank, reducing interference while preserving cross-model transferability.

Specifically, ℓ^+ denotes the identity of the model that generates the preferred trajectory τ^+ for task x, and ℓ^- denotes the identity of the model that generates the unpreferred trajectory τ^-. For the multi-trajectory setting, the extracted memories for task x are defined as:

$$\mathcal{M}^{(x)} = \bigcup_{\tau_{\ell^-} \in \mathcal{T}^-} \left\{ \left(\{m_k\}_{k=1}^K, \ell^+, \ell^- \right) \right\}. \tag{6}$$

By abstracting instance-specific solutions into reusable reasoning constraints, the extracted guidance can generalize across problems. The model-identity labels preserve the contrast direction, indicating which model provides the successful reference and which exhibits the corresponding failure pattern. This pairwise labeling supports model-aware retrieval and reduces interference across heterogeneous LLMs. The final memory bank $\mathcal{M}$ stores only **normative, abstract reasoning constraints** with preference-pair model labels, rather than raw trajectories, demonstrations, or final solutions.

Algorithm 1: Multi-model Memory Construction

Input: Model $\mathcal{F} = \{f_i\}_{i=1}^N$; tasks $\mathcal{D}$; summarizer $f_{i^\star}$; strongest index $i^\star$; evaluator $\mathcal{I}(\cdot)$; max entries K.
Output: Labeled memory bank $\mathcal{M}$.

$\mathcal{M} \leftarrow \emptyset$
for each task $x \in \mathcal{D}$ **do**
  ▷ collect multi-model trajectories
  for $i = 1, \ldots, N$ **do**
    $\tau_i \leftarrow f_i(x)$
  end for
  ▷ select preferred trajectory
  if $\mathcal{I}(\tau_{i^\star}) = 0$ and $\exists i \neq i^\star,\ \mathcal{I}(\tau_i) = 1$ **then**
    $l^+ \sim \mathrm{Unif}(\{i \mid \mathcal{I}(\tau_i) = 1\})$
  else
    $l^+ \leftarrow i^\star$
  end if
  $\tau^+ \leftarrow \tau_{l^+}$; $\mathcal{T}^- \leftarrow \{\tau_i\}_{i=1}^N \setminus \{\tau^+\}$
  ▷ construct preference-pair labeled memory
  for each $\tau_j^- \in \mathcal{T}^-$ **do**
    $\{(v_k, i_k)\}_{k \le K} \leftarrow f_{i^\star}(\tau_j^-, \tau^+)$
    $\{m_k\}_{k \le K} = \{(\text{enforce } i_k;\ \text{avoid } v_k)\}_{k \le K}$
  end for
  $\mathcal{M}^{(x)} \leftarrow \bigcup_{\tau_{l^-} \in \mathcal{T}^-} \left\{ \left(\{m_k\}_{k=1}^K, \ell^+, \ell^- \right) \right\}$
  $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{M}^{(x)}$
end for
return $\mathcal{M}$

2.3 Task-Aware Memory Retrieval and Inference

Given a new task q and a target LLM f_j, MemCollab retrieves memory entries from the shared memory bank $\mathcal{M}$ and conditions the model's inference on the retrieved constraints; see Algorithm 2 in the appendix. Retrieval follows a task-first and model-aware strategy. Specifically, we first select memories that match the task type of q, and then retain memories constructed from contrastive pairs in which the target model f_j participated, regardless of whether it produced the preferred or unpreferred trajectory.

Task Categorization. Each memory entry in $\mathcal{M}$ is associated with the task category of its source task and the identities of the two LLMs involved in the corresponding contrastive pair. We denote a memory entry as

$$m = (\eta_m, c_m, u_m, \ell_m^+, \ell_m^-), \tag{7}$$

where η_m denotes the stored memory content, i.e., the normative reasoning constraint of the form (enforce i_k; avoid v_k), (c_m, u_m) is the task category and subcategory inherited from the source task, and (ℓ_m^+, ℓ_m^-) are the identities of the models that produced the preferred and unpreferred trajectories, respectively. Given a test task q, an LLM classifier assigns it a category and subcategory:

$$(c_q, u_q) = f(q). \tag{8}$$

Task-Aware Filtering. We first restrict retrieval to memory entries whose task labels match the test task:

$$\mathcal{M}_q^{\text{task}} = \{m \in \mathcal{M} \mid c_m = c_q,\ u_m = u_q\}. \tag{9}$$

This step ensures that the retrieved constraints are drawn from tasks with similar reasoning structures rather than unrelated problem categories.

Model-Aware Filtering. After task-aware filtering, we further select memory entries whose contrastive construction involved the target model. Let ℓ_j denote the identity label of A_j. A memory entry is retained if ℓ_j appears as either the preferred-model label or the unpreferred-model label:

$$\mathcal{M}_{q,j}^{\text{cand}} = \{m \in \mathcal{M}_q^{\text{task}} \mid \ell_j \in \{\ell_m^+, \ell_m^-\}\}. \tag{10}$$

This design allows the target model to benefit from both directions of contrast. If $\ell_j = \ell_m^-$, the retrieved memory captures failure patterns that the target model should avoid. If $\ell_j = \ell_m^+$, the retrieved memory captures reasoning invariants that the target model has successfully followed and can reinforce. Thus, any memory constructed from a comparison involving A_j is considered model-relevant.

Relevance Ranking and Inference. Finally, MEMCOLLAB ranks the candidate memories by their semantic relevance to the test task q and selects the top-p entries:

$$\mathcal{M}_{q,j} = \text{Top-}p\left(\mathcal{M}_{q,j}^{\text{cand}}, \text{sim}(q, \eta_m)\right). \tag{11}$$

The retrieved memory set $\mathcal{M}_{q,j}$ is then incorporated into the inference prompt of the target LLM f_j. During generation, the model is encouraged to follow the retrieved reasoning invariants and avoid the corresponding violation patterns. The detailed retrieval procedure is provided in Appendix E. This two-stage retrieval process first enforces task relevance and then model relevance, ensuring that the target model receives constraints that are appropriate for the current task and grounded in contrastive experience involving that model.

2.4 WHY CONTRAST WORKS

Constructing transferable memory across LLMs is challenging because a reasoning trajectory may contain both task-relevant principles and model-specific solution choices. Some of these choices are incorrect or unsupported, while others are valid but unnecessary and may not transfer well to another model. For example, a trajectory may apply an incorrect prime-factor aggregation rule, introduce an unsupported post-processing operation, or use a longer model-preferred derivation when a simpler constraint-based strategy is sufficient. These observations motivate contrastive memory construction. Suppose a trajectory is represented as $\tau = f(s, b)$, where s denotes task-relevant reasoning structure and b denotes model-specific solution choices. By comparing preferred and unpreferred trajectories for the same task, MEMCOLLAB identifies which reasoning components are supported by successful problem solving and which components are absent, contradicted, or unnecessary. The resulting memory therefore emphasizes transferable principles, such as limiting-factor rules, physical invariants, or useful constraint manipulations, while suppressing incorrect, unsupported, or less transferable trajectory-specific procedures. We interpret the distilled memory as $m = \phi(s)$, where ϕ retains the reasoning structure most relevant to successful and transferable problem solving. This contrastive view motivates the MemNCE-style interpretation in Appendix A.1.

3 EXPERIMENTS

This section evaluates the effectiveness of our framework, MEMCOLLAB, for constructing a single collaborative memory system that benefits multiple models through contrastive memory learning.

Models and Datasets. We evaluate Qwen2.5-7B-Instruct (Team et al., 2024), Qwen2.5-32B-Instruct (Team et al., 2024), and LLaMA-3-8B-Instruct (Grattafiori et al., 2024), all instantiated with the same SmolAgents execution framework. We consider mathematical reasoning on MATH500 (Lightman et al., 2023; Hendrycks et al., 2021) and GSM8K (Cobbe et al., 2021), code generation on MBPP (Austin et al., 2021) and HumanEval (Chen, 2021), and additional agentic evaluation on AppWorld (Trivedi et al., 2024). Further details are provided in Appendix B.

Baselines. We compare MEMCOLLAB against two groups of baselines. **(i) Memory content baselines.** Buffer of Thoughts (BoT) (Yang et al., 2024) maintains a meta-buffer of reusable high-level thought templates distilled from prior problem-solving, while Dynamic Cheatsheet (DC) (Suzgun et al., 2025) maintains an evolving cheatsheet of reusable guidance during inference. **(ii) Memory source baselines.** To isolate collaborative construction from naive transfer, we vary only the source of memory: w/ Memory(7B), w/ Memory(8B), and w/ Memory(32B) use memory summarized from trajectories generated by the corresponding LLM, while w/ Self-Contrast Memory constructs memory by contrasting multiple trajectories sampled from the same model. Memory extraction follows BoT (Yang et al., 2024). Detailed prompts are provided in Appendix E.

3.1 MAIN RESULTS

Table 1 reports results within the same model family Qwen-2.5 across different tasks, with variance reported in Appendix D.6. Additional results across model families, model scales, and task settings are provided in Appendix C. Results on the agentic planning task are provided in Appendix 3.4.

Method	MATH500	GSM8K	MBPP	HumanEval	Average
Qwen-2.5-7B					
Base	52.2	85.4	47.9	42.7	57.1
+ Buffer of Thoughts	45.8 (−6.4)	86.4 (+1.0)	**57.6** (+9.7)	62.2 (+19.5)	63.0 (+5.9)
+ Dynamic Cheatsheet	66.0 (+13.8)	83.8 (−1.6)	36.7 (−11.2)	40.7 (−2.0)	56.8 (−0.3)
+ Memory (7B)	60.0 (+7.8)	86.2 (+0.8)	50.2 (+2.3)	42.7 (+0.0)	59.8 (+2.7)
+ Memory (32B)	50.6 (−1.6)	86.6 (+1.2)	48.6 (+0.7)	34.1 (−8.6)	55.0 (−2.1)
+ Self-Contrast Memory	58.4 (+6.2)	85.6 (+0.2)	52.5 (+4.6)	73.3 (+30.6)	67.5 (+10.4)
+ MEMCOLLAB	**67.0** (+14.8)	**87.4** (+2.0)	**57.6** (+9.7)	**74.4** (+31.7)	**71.6** (+14.5)
Qwen-2.5-32B					
Base	63.8	93.0	58.0	68.3	70.8
+ Buffer of Thoughts	68.2 (+4.4)	93.0 (+0.0)	63.8 (+5.8)	84.1 (+15.8)	77.3 (+6.5)
+ Dynamic Cheatsheet	73.4 (+9.6)	90.8 (−2.2)	57.6 (−0.4)	64.6 (−3.7)	71.6 (+0.8)
+ Memory (32B)	69.6 (+5.8)	93.2 (+0.2)	60.3 (+2.3)	79.3 (+11.0)	75.6 (+4.8)
+ Self-Contrast Memory	69.6 (+5.8)	93.4 (+0.4)	58.7 (+0.7)	**87.8** (+19.5)	77.4 (+6.6)
+ MEMCOLLAB	**73.8** (+10.0)	**93.6** (+0.6)	**64.3** (+6.3)	86.6 (+18.3)	**79.6** (+8.8)

Table 1: Performance of different memory methods with Qwen-2.5-7B and Qwen-2.5-32B on math and code benchmarks. Numbers in parentheses denote absolute accuracy changes relative to the corresponding vanilla model.

Cross-model memory enables smaller LLMs to benefit from larger-model trajectories. With MEMCOLLAB, Qwen2.5-7B improves from 52.2% to 67.0% on MATH500 and from 47.9% to 57.6% on MBPP. These gains indicate that contrast-derived memory can transfer useful reasoning guidance across model scales rather than simply reproducing the solution patterns of the source model.

Contrastive memory benefits both smaller and larger LLMs. For Qwen2.5-32B, MEMCOLLAB consistently improves over the no-memory setting, achieving 73.8% on MATH500, 93.6% on GSM8K, and 64.3% on MBPP. This shows that collaborative memory is useful not only for smaller models, but also for larger models by providing reusable reasoning and error-avoidance guidance.

Naive cross-model memory transfer can degrade performance. Memory distilled solely from Qwen2.5-32B trajectories (*w/ Memory (32B)*) reduces Qwen2.5-7B performance on MATH500 (50.6% vs. 52.2%) and HumanEval (34.1% vs. 42.7%), showing that memory constructed for one LLM does not necessarily transfer effectively to another. Self-contrast memory provides partial gains but remains below MEMCOLLAB, suggesting that cross-model contrast provides additional information for distilling reasoning patterns that transfer across heterogeneous models.

3.2 RESULTS ACROSS MODEL FAMILIES

Table 2 presents the results when collaborative memory is constructed from different model families.

Contrastive memory generalizes across model architectures. With cross-family model pairs, MEMCOLLAB consistently improves performance across benchmarks, indicating that the extracted memory is not limited to models from the same architecture or family. For example, LLaMA3-8B equipped with cross-family contrastive memory achieves 74.4% on MATH500 and 48.8% on HumanEval, outperforming both the no-memory baseline and single-source memory transfer. Qwen2.5-32B also benefits from memory constructed with trajectories from a different model family, despite substantial differences in architecture and pretraining. These gains suggest that the contrastive construction process captures reasoning strategies and error-avoidance constraints that remain useful across heterogeneous models, with reduced dependence on solution patterns specific to a particular model family. Overall, the results provide evidence that MEMCOLLAB can support memory sharing beyond differences in model scale and extend to cross-architecture settings.

3.3 ROBUSTNESS TO HETEROGENEOUS MEMORY SOURCES.

Table 3 examines whether adding memory from an additional heterogeneous model introduces interference into the shared bank. For the two-model settings, we construct memory banks using trajectories from Qwen2.5-32B paired with either Qwen2.5-7B or LLaMA3-8B on MATH500, as shown in the second row of Table 3. The third row reports the performance using a memory bank constructed from trajectories from all three models. The shared bank remains stable after adding this heterogeneous source, and LLaMA3-8B also benefits from the shared memory, improving from 42.4

Method	MATH500	GSM8K	MBPP	HumanEval	Average
Llama-3-8B-Instruct					
Base	27.4	73.0	37.0	29.3	41.7
+ Buffer of Thoughts	20.6 (−6.8)	73.6 (+0.6)	43.2 (+6.2)	34.2 (+4.9)	42.9 (+1.2)
+ Dynamic Cheatsheet	29.4 (+2.0)	71.2 (−1.8)	27.2 (−9.8)	17.1 (−12.2)	36.2 (−5.5)
+ Memory (8B)	29.2 (+1.8)	63.2 (−9.8)	47.1 (+10.1)	42.7 (+13.4)	45.6 (+3.9)
+ Memory (32B)	18.8 (−8.6)	56.6 (−16.4)	35.8 (−1.2)	34.1 (+4.8)	36.3 (−5.4)
+ Self-Contrast Memory	23.6 (−3.8)	61.0 (−12.0)	**51.4** (+14.4)	43.9 (+14.6)	45.0 (+3.3)
+ MEMCOLLAB	**42.4** (+15.0)	**74.4** (+1.4)	49.8 (+12.8)	**48.8** (+19.5)	**53.9** (+12.2)
Qwen-2.5-32B-Instruct					
Base	63.8	93.0	58.0	68.3	70.8
+ Memory (32B)	69.6 (+5.8)	93.2 (+0.2)	**60.3** (+2.3)	79.3 (+11.0)	75.6 (+4.8)
+ MEMCOLLAB	**70.6** (+6.8)	**95.2** (+2.2)	**60.3** (+2.3)	**86.6** (+18.3)	**78.2** (+7.4)

Table 2: Cross-family performance of Llama-3-8B-Instruct and Qwen-2.5-32B-Instruct. The results for Qwen-2.5-32B model for other baselines are provided in the Table 1.

to 43.2. This suggests that additional heterogeneous trajectories do not necessarily introduce harmful cross-model interference.

The gains are non-uniform across models. LLaMA3-8B is closer in capability to Qwen2.5-7B than to Qwen2.5-32B and may therefore contribute failure patterns that overlap with those already captured for Qwen2.5-7B. This may explain why Qwen2.5-7B sees no additional gain, while Qwen2.5-32B benefits from the new contrast. Overall, Table 3 supports the robustness of MEMCOLLAB's shared-bank and model-aware retrieval design under heterogeneous memory sources, rather than suggesting that performance should improve simply by adding more models.

# Models	Qwen2.5-7B	Qwen2.5-32B	Llama3-8B
1	52.2	58.0	27.4
2	**67.0**	73.8 70.6	42.4
3	66.8	**75.0**	**43.2**

Table 3: Three-model collaboration on MATH500.

3.4 GENERALIZATION TO AGENTIC PLANNING TASKS

Beyond math and coding benchmarks, we additionally evaluate MEMCOLLAB on AppWorld (Trivedi et al., 2024), a challenging long-horizon multi-app agentic benchmark. Task and Scenario denote task- and scenario-level goal completion; D1–D3 indicate increasing difficulty. This setting introduces additional challenges such as environment interaction, tool-use reliability, and multi-step planning. Since the original 32B setup achieves near-zero performance on this benchmark, we instead evaluate GPT-5-mini and GPT-5-nano. As shown in Table 4, MEMCOLLAB improves both task goal completion and scenario goal completion for both GPT-based models.

Method	Metric	Avg.	D1	D2	D3
GPT-5-nano	Task	4.8	11.1	0.0	0.0
	Scenario	0.0	0.0	0.0	0.0
+ MEMCOLLAB	Task	**26.7**	**44.4**	**26.7**	0.0
	Scenario	**10.0**	0.0	**20.0**	0.0
GPT-5-mini	Task	47.7	66.7	55.3	0.0
	Scenario	30.0	66.7	20.0	0.0
+ MEMCOLLAB	Task	**63.3**	66.7	**66.7**	**50.0**
	Scenario	**50.0**	66.7	**40.0**	**50.0**

Table 4: Agentic planning on AppWorld.

3.5 EFFECT OF MEMORY RETRIEVAL

Retrieval strategy. We evaluate two retrieval variants that do not use task classification, instead retrieving memory entries directly based on (1) embedding similarity and (2) prompting the model to select the top-p relevant tasks. As shown in Figure 3, these results highlight the benefit of first classifying the task and then retrieving memory conditioned on the predicted class. Compared with retrieval over the full memory bank, task-aware retrieval narrows the candidate space and reduces interference from memories associated with unrelated failure patterns.

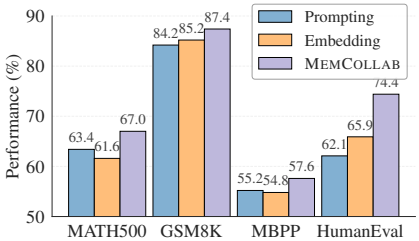

Figure 3: Retrieval method comparison.

We study the effect of memory retrieval from two perspectives: the number of retrieved memory entries and the strategy used to select relevant memories. Additional ablation studies, including the effects of summarizer choice, prompt stability, and preference selection, are provided in Appendix D.

Number of retrieved memory entries. Figure 4 shows that performance first improves as more memories are retrieved, but drops beyond a task-dependent threshold. Since retrieving the top three memories consistently yields strong results, we set $p = 3$ in all experiments. This trend can be viewed as a pruning trade-off. Retrieved contrastive memories help eliminate irrelevant reasoning directions and improve the signal-to-noise ratio. If b is the original branching factor, d is the search depth, and k erroneous patterns are avoided, the effective search space contracts by $\rho = \left(1 - \frac{k}{b}\right)^d$. When $k \geq 1$, $\rho < 1$, indicating reduced search complexity. However, retrieving too many memories introduces weakly relevant or noisy directions, causing attention dispersion and weakening the pruning effect. This explains the non-monotonic trend and motivates a task-dependent retrieval budget.

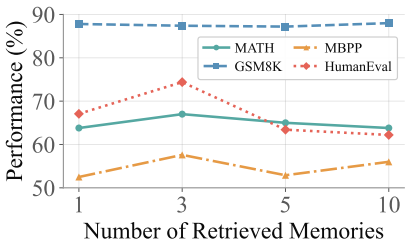

Figure 4: Ablation on the number of retrieved memories.

As we further verify in Appendix D.5, task categories exhibit distinct and internally consistent error-type distributions, supporting our design of retrieving memory at the task-category level.

3.6 Comparable-scale collaboration

To rule out the possibility that gains come only from transferring knowledge from a much larger model to a much smaller one, we test collaboration between models of similar capacity. Table 5 shows that MemCollab improves both Qwen2.5-7B-Instruct and Llama-3-8B Instruct over their vanilla and self-memory baselines, indicating that useful contrast still exists between similar-scale but architecturally different models. The gains persist at a larger comparable scale as well: on MBPP, MemCollab improves Llama-3-70B-Instruct from 54.8 to 61.9 and Qwen2.5-72B-Instruct from 59.9 to 63.0. Full results across all benchmarks and model scales are provided in Appendix C.3.

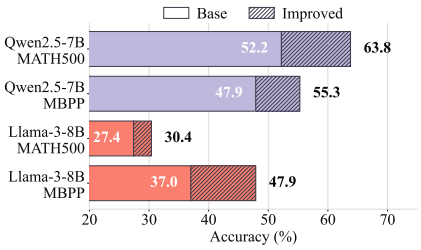

Figure 5: Similar-scale collaboration.

3.7 Case Study

In Figure 6, we illustrate how MemCollab constructs and applies contrastive memory. For the source problem, the unpreferred response incorrectly assumes independence in a probability setting with dependent events, whereas the preferred response conditions on earlier events and systematically enumerates the relevant cases. The resulting memory distills this distinction into reusable guidance. On the target problem, the vanilla model exhibits the same failure mode, while retrieving the contrastive memory leads to explicit case analysis, a correct joint-probability formulation, and the correct solution.

Incorrect reasoning rule. For the problem "Compute the largest integer k such that 2004^k divides $2004!$," one memory incorrectly suggests summing prime-factor exponents, while MemCollab retains the correct limiting-factor rule based on the minimum normalized prime valuation.

Valid but model-specific procedure. For "Let $a, b > 0$ satisfy $a + 2b = 1$," both source solutions are valid: one uses Lagrange multipliers, while the other directly eliminates a variable using the equality constraint. MemCollab retains the more direct constraint-based strategy while omitting the longer model-specific derivation.

These examples show that contrastive construction filters both incorrect reasoning rules and valid but unnecessarily model-specific procedures. Appendix D.7 further shows that MemCollab produces higher-quality memory than Self-Memory under an LLM-as-a-judge evaluation.

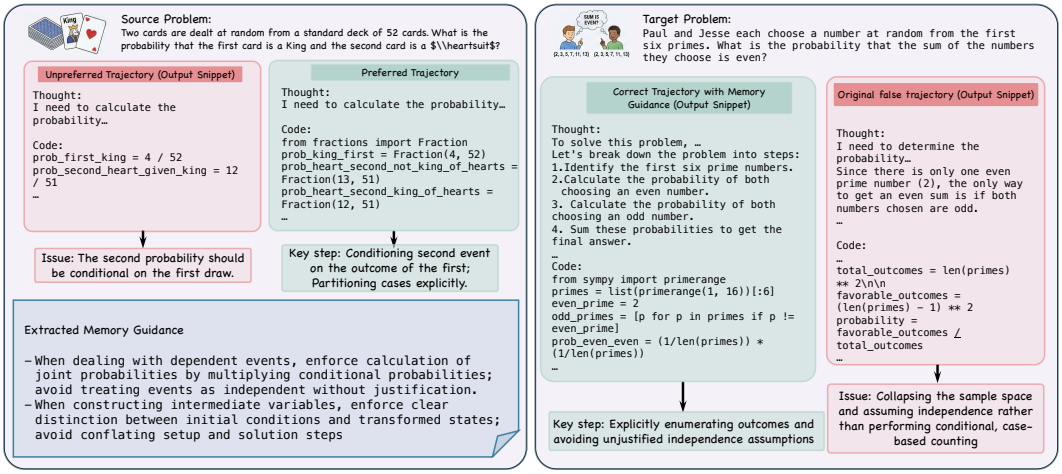

Figure 6: Case study to compare the trajectories with and without the extracted memory guidance. *Left*: The contrasted trajectories pairs. *Right*: Changes in the model's output after incorporating the memory guidance. Note: all `Key step` and `Issue` are summarized, see full output is in Appendix D.8.

4 Related Works

Memory for LLM Memory has become an important mechanism for LLMs, helping mitigate limited context windows and reuse information from prior problem-solving experiences (Zhang et al., 2025; Nuxoll & Laird, 2012). Memory may take the form of free-form text or structured artifacts and can be retrieved to guide later reasoning and tool use (Ouyang et al., 2025; Zhong et al., 2024; Packer et al., 2023). Existing approaches include episodic and long-term memory (Packer et al., 2023; Zhong et al., 2024), reasoning-oriented memory that distills reusable guidance (Yang et al., 2024; Suzgun et al., 2025; Ouyang et al., 2025), and learned mechanisms for memory organization, update, and retrieval (Xu et al., 2025; Zhou et al., 2025; Yan et al., 2025; Yang et al., 2025). However, most methods construct memory from a single LLM and reuse it with the same model, limiting transferability. In contrast, we construct shared memory by contrasting trajectories from different LLMs on the same task to distill transferable reasoning constraints and error-avoidance patterns.

Retrieval-Augmented Model Generation Retrieval-augmented generation (RAG) extends LLMs with external knowledge by conditioning generation on retrieved content (Lewis et al., 2020). Prior work shows that both retrieval source and granularity affect generation quality (Gao et al., 2023). Retrieval sources range from unstructured and structured data (Li et al., 2023; Zha et al., 2023; Gaur et al., 2022; Edge et al., 2024) to LLM-generated or distilled content (Yu et al., 2022; Cheng et al., 2023), while retrieval units range from long passages to finer-grained sentences, entities, or code snippets (Yu et al., 2024; Shi et al., 2023; Chen et al., 2024; Wang et al., 2024). Unlike document-centric RAG, our method retrieves reasoning-oriented memory. We first filter memory by task category and then rank relevant entries, reducing incompatible guidance while preserving efficient retrieval of reusable reasoning constraints.

5 Conclusion

In this work, we study shared cross-model memory for heterogeneous LLMs. We show that naive memory transfer across models can degrade performance because stored memory may mix transferable task knowledge with model-specific solution patterns. To address this, we propose MemCollab, which constructs collaborative memory by contrasting trajectories generated by different LLMs on the same task. Experiments on mathematical reasoning and code generation benchmarks show that MemCollab improves both accuracy and inference-time efficiency across different model scales and families. These results demonstrate that contrast-derived memory can serve as a reusable reasoning resource across heterogeneous LLMs.

AI DISCLOSURE

In this work, we used generative AI tools to polish the language, improve the clarity of the manuscript, and assist with literature search. We manually reviewed all AI-assisted revisions and verified the identified literature to ensure the accuracy and originality of the presented work. We take full responsibility for the final content of this work, including all text, claims, citations, and artifacts produced with the aid of generative AI.

REFERENCES

Anthropic. Effective context engineering for ai agents, September 2025. URL https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. Dense x retrieval: What retrieval granularity should we use? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 15159–15177, 2024.

Xin Cheng, Di Luo, Xiuying Chen, Lemao Liu, Dongyan Zhao, and Rui Yan. Lift yourself up: Retrieval-augmented text generation with self-memory. *Advances in Neural Information Processing Systems*, 36:43780–43799, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023.

Manas Gaur, Kalpa Gunaratna, Vijay Srinivasan, and Hongxia Jin. Iseeq: Information seeking question generation using dynamic meta-information retrieval and knowledge graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pp. 10672–10680, 2022.

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

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

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. *arXiv preprint arXiv:2510.00615*, 2025.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. Llms get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*, 2025.

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

Xiaoqian Li, Ercong Nie, and Sheng Liang. From classification to generation: Insights into crosslingual retrieval augmented icl. *arXiv preprint arXiv:2311.06595*, 2023.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Lajanugen Logeswaran and Honglak Lee. An efficient framework for learning sentence representations. *arXiv preprint arXiv:1803.02893*, 2018.

Andrew M Nuxoll and John E Laird. Enhancing intelligent agents with episodic memory. *Cognitive Systems Research*, 17:34–48, 2012.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: Towards llms as operating systems. 2023.

Mathis Pink, Qinyuan Wu, Vy Ai Vo, Javier Turek, Jianing Mu, Alexander Huth, and Mariya Toneva. Position: Episodic memory is the missing piece for long-term llm agents. *arXiv preprint arXiv:2502.06975*, 2025.

Nikunj Saunshi, Orestis Plevrakis, Sanjeev Arora, Mikhail Khodak, and Hrishikesh Khandeparkar. A theoretical analysis of contrastive unsupervised representation learning. In *International conference on machine learning*, pp. 5628–5637. PMLR, 2019.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pp. 31210–31227. PMLR, 2023.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

Qwen Team et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2(3), 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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Liang Wang, Nan Yang, and Furu Wei. Learning to retrieve in-context examples for large language models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1752–1767, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Kristian Kersting, Jeff Z Pan, Hinrich Schütze, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 37:113519–113544, 2024.

Ling Yang, Zhaochen Yu, Bin Cui, and Mengdi Wang. Reasonflux: Hierarchical llm reasoning via scaling thought templates. *arXiv preprint arXiv:2502.06772*, 2025.

Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. Generate rather than retrieve: Large language models are strong context generators. *arXiv preprint arXiv:2209.10063*, 2022.

Wenhao Yu, Hongming Zhang, Xiaoman Pan, Peixin Cao, Kaixin Ma, Jian Li, Hongwei Wang, and Dong Yu. Chain-of-note: Enhancing robustness in retrieval-augmented language models. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 14672–14685, 2024.

Liangyu Zha, Junlin Zhou, Liyao Li, Rui Wang, Qingyi Huang, Saisai Yang, Jing Yuan, Changbao Su, Xiang Li, Aofeng Su, et al. Tablegpt: Towards unifying tables, nature language and commands into one gpt. *arXiv preprint arXiv:2307.08674*, 2023.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025.

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*, volume 38, pp. 19724–19731, 2024.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*, 2025.

A Additional Theoretical Details

This appendix provides supplementary theoretical analysis to support our interpretation of our contrastive memory construction.

A.1 Contrastive Learning

Contrastive learning provides a statistical framework for learning representations by distinguishing positive pairs from negative pairs. A widely used objective is the InfoNCE loss:

$$\mathcal{L}_{\text{InfoNCE}} = -\mathbb{E}\left[\log \frac{\exp(g(x, x^+))}{\exp(g(x, x^+)) + \sum_j \exp(g(x, x_j^-))}\right], \tag{12}$$

where $g(\cdot, \cdot)$ is a scoring function, (x, x^+) denotes a positive pair, and $\{x_j^-\}$ are negative samples (Logeswaran & Lee, 2018).

For the pairwise setting with a single negative sample, the objective reduces to (Saunshi et al., 2019):

$$-\mathbb{E}\left[\log \frac{\exp(g(x, x^+))}{\exp(g(x, x^+)) + \exp(g(x, x^-))}\right]. \tag{13}$$

A.2 Contrastive Memory Learning

We use contrastive learning as an interpretive perspective for understanding why contrast-derived memory can improve cross-model transfer. A reasoning trajectory τ may contain both task-relevant reasoning structure and model-specific solution patterns. We write

$$\tau = f(s, b), \tag{14}$$

where s denotes reasoning principles relevant to solving the task, while b captures model-dependent choices such as preferred solution procedures, heuristic shortcuts, or stylistic variations. The goal of contrast-derived memory is to retain information associated with s while reducing dependence on b. We therefore view the resulting memory as

$$m = \psi(s). \tag{15}$$

Given a preferred trajectory τ^+ and unpreferred trajectories $\{\tau_j^-\}$ for the same task, we define the following proxy objective:

$$\mathcal{L}_{\text{MemNCE}} = -\mathbb{E}\left[\log \frac{\exp(g(\tau^+, m))}{\exp(g(\tau^+, m)) + \sum_j \exp(g(\tau_j^-, m))}\right]. \tag{16}$$

Here, $g(\tau, m)$ measures the compatibility between a trajectory and the distilled memory. In our implementation, each contrast is formed between one preferred and one unpreferred trajectory, corresponding to the pairwise case with a single negative sample. Prior work shows that pairwise contrast retains the density-ratio interpretation underlying contrastive objectives (Oord et al., 2018).

Importantly, MemCollab does not directly optimize $\mathcal{L}_{\text{MemNCE}}$. Instead, this objective serves as a conceptual proxy for the memory construction process. Contrasting trajectories from the same task exposes which reasoning components are supported by successful problem solving and which are incorrect, unsupported, or specific to a particular model's solution procedure. The resulting memory therefore emphasizes transferable reasoning constraints and error-avoidance patterns while suppressing less transferable trajectory-specific information. Compared with storing raw trajectories or demonstrations, this contrastive abstraction provides a mechanism for distilling compact, reusable guidance that can generalize across tasks and heterogeneous LLMs.

B Additional Experimental Details

All experiments are conducted on 8 NVIDIA RTX 6000 GPUs with CUDA 12.8. We adopt smolmodels' Codemodel as the underlying execution framework. During inference, we set the temperature

to 0 and top-p to 0.8. For memory retrieval, we retrieve the top-3 most similar samples within the task-filtered subset using TF-IDF similarity scores; each retrieved sample contributes up to 3 extracted memory entries, which are then provided to the model. For the self-contrast baseline, each model generates two sampled outputs to construct contrastive memory. We use the random seed 42 for all experiments.

B.1 DATASET

- **MATH500**: A curated reasoning benchmark consisting of 500 problems sampled from the MATH dataset (Hendrycks et al., 2021) following the protocol of Lightman et al. (2023). The benchmark spans seven mathematical domains, including Prealgebra, Algebra, Number Theory, Counting and Probability, Geometry, Intermediate Algebra, and Precalculus, ensuring to provide diverse coverage of problem types and reasoning requirements.
- **GSM8K** A grade-school mathematics benchmark consisting of 8,500 natural-language word problems that require multi-step arithmetic reasoning (Cobbe et al., 2021). Each problem is paired with a step-by-step solution and a final numeric answer, and the dataset is commonly used to evaluate an model's ability to perform structured mathematical reasoning.
- **MBPP** (Mostly Basic Programming Problems): A benchmark for evaluating code generation and program synthesis, consisting of 974 Python programming tasks (Austin et al., 2021). Each task provides a natural-language problem description, a reference solution, and a set of test cases, and is designed to assess an model's ability to generate correct and executable code from textual specifications.
- **HumanEval**: A widely used benchmark for evaluating code generation and program synthesis capabilities of large language models, consisting of 164 hand-written Python programming tasks (Chen, 2021). Each task specifies a function signature and a natural-language docstring, and models are evaluated based on functional correctness using unit tests, typically reported with pass@k metrics.

B.2 BASELINES

- Buffer of Thoughts (Yang et al., 2024): proposes a reasoning-oriented memory mechanism that distills reusable thought templates from prior problem-solving traces. During inference, the model retrieves and instantiates these templates to guide its reasoning process, aiming to improve consistency and efficiency on complex reasoning tasks.
- Dynamic Cheatsheet (Suzgun et al., 2025): introduces an inference-time memory mechanism that maintains a persistent, editable cheatsheet of task-relevant guidance. The cheatsheet is updated online based on model successes and failures, enabling black-box language models to adapt their behavior across tasks without parameter updates.
- Vanilla: is to use the basic model wthout memory augmentation.
- w/ Memory (7B): is to utilize the memory constructed exclusively from reasoning trajectories generated by Qwen2.5-7B. During inference, the model retrieves the stored memory and conditions its generation on the retrieved content to guide problem solving.
- w/ Memory (32B): has the setting mirrors w/ Memory (7B), except that the memory is constructed from reasoning trajectories generated by Qwen2.5-32B instead of Qwen2.5-7B.
- w/ Self-Contrast Memory: constructs memory by contrasting the model's own self-generated reasoning trajectories, using the resulting memory to guide inference.

C ADDITIONAL EXPERIMENTS

We further examine whether MEMCOLLAB generalizes across model families, scale regimes, and task settings. Rather than exhaustively covering all possible model combinations, this evaluation tests representative axes of heterogeneity beyond the main experiments, including larger-scale cross-family collaboration, additional cross-family collaboration, comparable-scale collaboration, larger comparable-scale collaboration, and long-horizon agentic planning.

These settings clarify that the gains of MEMCOLLAB do not rely on a single backbone family, a single scale regime, or a large weak–strong capability gap. Since MEMCOLLAB extracts reusable reasoning

invariants and failure-avoidance constraints by contrasting reasoning trajectories, its effectiveness depends on useful cross-model contrast rather than model size differences alone. Taken together, the results show that MEMCOLLAB remains effective across larger-scale, comparable-scale, cross-family, and GPT-based agentic settings.

C.1 LARGER-SCALE CROSS-FAMILY COLLABORATION.

We first evaluate Llama-3-70B-Instruct in collaboration with Qwen2.5-7B, extending the study to a larger-scale Llama model. As shown in Table 5, MEMCOLLAB improves Llama-3-70B over both the vanilla and self-memory baselines across all benchmarks.

Model	Method	MATH500	GSM8K	MBPP	HumanEval
Llama-3-70B	Vanilla	58.0	93.2	54.8	64.6
	w/ Memory (self)	59.6	93.5	56.3	68.6
	w/ MEMCOLLAB	**65.2**	**93.6**	**63.0**	**74.4**

Table 5: Larger-scale cross-family collaboration with Llama-3-70B-Instruct.

C.2 ADDITIONAL CROSS-FAMILY COLLABORATION.

We also evaluate Gemma3-4B-Instruct with Qwen2.5-7B on MBPP, providing another cross-family setting beyond Qwen and Llama. As shown in Table 6, MEMCOLLAB improves both Gemma3-4B-Instruct and Qwen2.5-7B-Instruct over their vanilla and self-memory baselines.

Model	Vanilla	w/ Memory (self)	w/ MEMCOLLAB
Gemma3-4B	45.9	45.1	**48.4**
Qwen2.5-7B	47.9	50.2	**50.8**

Table 6: Additional cross-family collaboration between Gemma3-4B-Instruct and Qwen2.5-7B-Instruct on MBPP.

C.3 COMPARABLE-SCALE COLLABORATION.

Model	Setting	MATH500	GSM8K	MBPP	HumanEval
Qwen2.5-7B	Vanilla	52.2	85.4	47.9	42.7
	w/ Memory (self)	60.0	**86.2**	50.2	42.7
	w/ MEMCOLLAB	**63.8**	85.4	**55.3**	**71.2**
Llama-3-8B	Vanilla	27.4	73.0	37.0	29.3
	w/ Memory (self)	29.2	63.2	47.1	42.7
	w/ MEMCOLLAB	**30.4**	**73.4**	**47.9**	**45.5**

Table 7: Comparable-scale cross-family collaboration between Qwen2.5-7B-Instruct and Llama-3-8B-Instruct.

We further test whether MEMCOLLAB remains useful when collaborating models have similar overall capacity in Table 7. This setting is important because it rules out the possibility that the gains come only from transferring knowledge from a much larger model to a much smaller one. In the comparable-scale cross-family setting, MEMCOLLAB improves both Qwen2.5-7B-Instruct and Llama-3-8B-Instruct over their vanilla and self-memory baselines, showing that useful contrast still exists between similar-scale but architecturally different models.

We additionally evaluate a larger comparable-scale setting with Llama-3-70B-Instruct and Qwen2.5-72B-Instruct on MBPP. As shown in Table 8, MEMCOLLAB improves Llama-3-70B from 54.8 to 61.9 and Qwen2.5-72B from 59.9 to 63.0, outperforming both vanilla and self-memory baselines.

Model	Setting	Accuracy
Llama-3-70B	Vanilla	54.8
	w/ Memory (self)	56.3
	w/ MEMCOLLAB	**61.9**
Qwen2.5-72B	Vanilla	59.9
	w/ Memory (self)	61.1
	w/ MEMCOLLAB	**63.0**

Table 8: Large comparable-scale collaboration between Llama-3-70B and Qwen2.5-72B on MBPP.

C.4 EXTENSION TO THE OPEN-ENDED TASK

Our method requires a correctness signal only during offline memory construction, where the indicator identifies preferred/unpreferred trajectories. At inference on new tasks, MEMCOLLAB does not rely on ground truth; it only retrieves abstract reasoning constraints distilled from prior tasks as soft guidance. In this paper we focus on math/code because ground-truth answers or executable verifiers provide a clean testbed. The same framework can extend to more open-ended tasks by replacing the current indicator with other preference sources, e.g., human preferences, model judges, or task-specific proxy rewards. To test this, we additionally evaluate on ASQA, using the backbone model itself to select the preferred trajectory and the results are shown in the Table 9. This suggests the framework can extend beyond closed-form tasks when a task-level preference signal is unavailable.

Model	Vanilla	w/ MEMCOLLAB
Qwen2.5-7B	15.4	**21.3**
Qwen2.5-32B	15.9	**33.3**

Table 9: Performance comparison with and without MEMCOLLAB on open-ended task ASQA.

D ADDITIONAL STUDIES

In this section, we provide additional analysis of MEMCOLLAB.

D.1 EFFECT OF PREFERENCE SELECTION

We further test whether the preference-selection rule affects the quality of the constructed memory. On Qwen2.5-7B, we compare the default rule with two alternative strategies: *random*, which randomly chooses the preferred trajectory, and *reverse*, which swaps the preferred and unpreferred trajectories under the default rule.

As shown in Table 10, both alternatives still improve over the vanilla baseline, but they are clearly worse than the default selection rule. MEMCOLLAB with the default rule achieves 57.6 on MBPP, compared with 50.2 for random selection and 52.9 for reverse selection. This indicates that correctly identifying the preferred trajectory is important for producing higher-quality memory.

Method	MBPP
Vanilla	47.9
MemCollab (default)	**57.6**
MemCollab (random)	50.2
MemCollab (reverse)	52.9

Table 10: Effect of preference selection on MBPP with Qwen2.5-7B.

D.2 Effect of Summarizer Choice

Algorithm 1 uses the stronger model for trajectory summarization. To test whether the gains depend on this specific summarizer, we vary only the summarizer model while keeping the rest of the MemCollab pipeline fixed. As shown in Table 11, summarizer capability affects memory quality: using Qwen2.5-32B-Instruct gives the best HumanEval result (74.4). However, the improvement is not specific to the larger model. Using Qwen2.5-7B-Instruct or Llama-3-8B-Instruct as the summarizer still substantially improves over the vanilla baseline, reaching 65.9 and 63.4, respectively. These results suggest that stronger summarizers can produce higher-quality memory, but MemCollab does not rely exclusively on the strongest model.

Target Model	Summarizer	HumanEval
Vanilla	–	42.7
w/ MemCollab	Qwen2.5-32B	**74.4**
	Qwen2.5-7B	65.9
	Llama-3-8B	63.4

Table 11: Effect of summarizer choice on HumanEval with Qwen2.5-7B as the target model.

D.3 Prompt Sensitivity

We evaluate the sensitivity of MemCollab to the summarization prompt. We keep the same trajectory pairs and downstream pipeline, and vary only the prompt used to summarize contrastive trajectories into memory entries. As shown in Table 12, the original prompt achieves 74.4 on HumanEval, and adding an instruction to output a numbered list achieves the same performance. Enforcing a JSON output format slightly reduces performance to 70.7, but still substantially improves over the no-memory baseline of 42.7. These results suggest that the gain mainly comes from the distilled memory content rather than a brittle prompt template.

Prompt Change	HumanEval
No memory	42.7
Original prompt	**74.4**
Add instruction to numbered list	**74.4**
Add instruction to JSON format	70.7

Table 12: Prompt sensitivity on HumanEval.

D.4 Efficiency Analysis

Memory Construction Cost We analyze the memory construction cost on MBPP using Qwen3-32B as the summarizer. As shown in Table 13, MemCollab requires more cost than self-memory, but is cheaper than self-contrast memory in both wall-clock time and total token usage. Specifically, MemCollab uses 38.3 seconds and 929.90 total tokens on average, compared with 50.4 seconds and 950.50 tokens for self-contrast. Although self-memory is cheaper, MemCollab provides a stronger trade-off between construction cost and downstream performance in our main experiments.

Method	Wall Time (s)	Input Tokens	Output Tokens	Total Tokens
MemCollab	38.3	812.42	117.50	929.90
Self-Contrast	50.4	828.90	121.56	950.50
Self-Memory	13.35	499.02	63.78	562.80

Table 13: Memory construction cost on MBPP using Qwen3-32B as the summarizer.

Memory improves inference-time efficiency. We evaluate inference-time efficiency by comparing the average number of reasoning turns required by a vanilla model and by MEMCOLLAB. As shown in Table 14, models equipped with collaborative memory consistently require fewer reasoning turns while achieving better overall performance. It indicates that contrast-derived memory provides effective guidance that reduces redundant exploration and unnecessary trial-and-error during inference. By discouraging previously observed failure patterns and instructing valid reasoning procedures, memory enables models to converge to correct solutions more efficiently.

Dataset	Vanilla	MEMCOLLAB
MATH500	2.7	2.2
GSM8K	1.8	1.6
MBPP	3.1	1.4
HumanEval	3.3	1.5

Table 14: Average number of reasoning turns for Qwen2.5-7B-Instruct

D.5 ERROR-PATTERN ANALYSIS BY TASK CATEGORY

To demonstrate the relationship between task categories and error patterns, we analyze the pairwise Jensen–Shannon divergence (JSD) between task categories on the MATH500 dataset using the Qwen2.5-7B model, as shown in Figure 7. Each task category is represented by an empirical distribution over error types aggregated from model-generated trajectories. For each data sample and its corresponding trajectory, we use GPT-4o-mini to annotate both the task category and the error type, and compute the JSD between all pairs of categories.

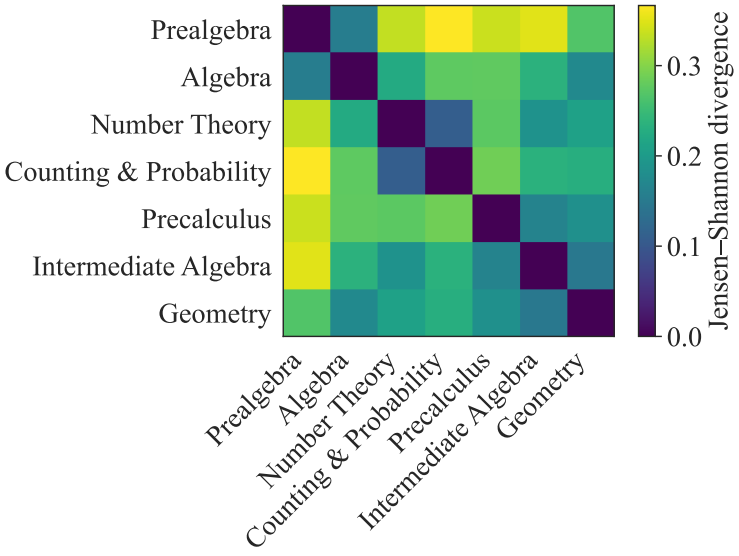

Figure 7: Task-category similarity of error-type distributions on MATH500.

Task categories exhibit characteristic and consistent error patterns. The results in Figure 7 reveal that algebraically related tasks, including Prealgebra, Algebra, Intermediate Algebra, and Precalculus, form a low-divergence cluster, indicating highly similar error distributions dominated by reasoning and execution-related failures. In contrast, categories such as Number Theory and Counting & Probability deviate from this cluster, indicating distinct failure mechanisms. These findings support the design of our task-aware retrieval mechanism, as memory entries derived from the same task category are more likely to preserve relevant reasoning constraints and error-forbidden patterns, enabling more targeted guidance during inference.

D.6 VARIANCE ACROSS RANDOM SEEDS

We report the standard deviation over three random seeds for both the vanilla baseline and MEMCOLLAB. As shown in Table 15, MEMCOLLAB improves average performance across all benchmarks

Method	MATH500	GSM8K	MBPP	HumanEval
Vanilla	52.2 ± 0.40	85.4 ± 0.00	47.9 ± 3.37	42.7 ± 5.32
w/ MEMCOLLAB	$\mathbf{67.0} \pm 0.42$	$\mathbf{87.4} \pm 0.31$	$\mathbf{57.6} \pm 0.90$	$\mathbf{74.4} \pm 3.52$

Table 15: Mean performance and standard deviation over three random seeds.

while maintaining comparable or lower variance on most tasks. These results suggest that collaborative memory not only improves accuracy, but also helps stabilize trajectory-sensitive reasoning and implementation.

D.7 LLM-AS-A-JUDGE EVALUATION OF MEMORY QUALITY

We further compare MEMCOLLAB against Self-Memory using an LLM-as-a-judge evaluation to assess the quality of the constructed memory. For each problem, we provide GPT-5-mini with the task question and two candidate memory entries, one produced by MEMCOLLAB and the other produced by Self-Memory. To reduce position bias, we randomize the order of the two candidate memories before presenting them to the judge. The judge is asked to select the memory that is more helpful for solving the task under a unified rubric covering relevance, correctness, actionability, generalization, and safety. Aggregated over the evaluation set, MEMCOLLAB achieves a winner rate of 0.72, compared with 0.24 for Self-Memory, with 0.04 ties. This suggests that the memory distilled by MEMCOLLAB is judged to be more useful overall, providing more relevant and actionable guidance while maintaining correctness and generality.

D.8 ADDITIONAL CASE STUDIES AND EXAMPLES

This section provides additional examples and details for analyzing the memory constructed by MEMCOLLAB. Table 16 presents representative retrieved trajectory pairs in which models with different reasoning capacities preserve the same underlying reasoning invariants despite variations in symbolic form, execution order, and implementation detail. These examples show that retrieval can match problems at the level of problem decomposition rather than surface-level reasoning steps.

We also provide the complete trajectories for the case study, including both the memory summarization process and the memory-augmented retrieval in Figure 8 and Figure 9. The memory used for augmentation is shown in Figure 6.

E ADDITIONAL DETAILS OF MEMCOLLAB

Shared Memory Clarification. We distinguish between a shared memory bank and model-agnostic memory. A shared memory bank refers to a unified repository constructed from trajectories of multiple heterogeneous models and accessible to all participating models. It does not require every memory entry to be equally useful for every model. In contrast, fully model-agnostic memory assumes that each memory entry is universally transferable across models, which may be unrealistic because different backbone models exhibit different failure modes. MEMCOLLAB therefore adopts a shared-bank, model-aware-retrieval design: memory entries are stored in a common bank but annotated with model-identity labels that indicate which model's failure mode the entry was distilled to address. At inference time, the target model retrieves entries from the shared bank that match both the task category and its model identity, reducing cross-model interference while preserving the benefit of collaborative memory construction.

Model- and Task-Aware Memory Retrieval Algorithm 2 describes the inference-time retrieval procedure. Given a query and a target model, MEMCOLLAB first filters memory entries by both the predicted task category and the target model identity, and then ranks the filtered entries by relevance.

Prompts We use the following prompt in Table 17, adapted from Buffer of Thoughts (Yang et al., 2024), to extract high-level reasoning behavior from a model's single trajectory. We also provide the prompt for contrasting trajectories on the math problems in Table 18.

Algorithm 2 Model- and Task-Aware Memory Retrieval and Inference

Require: Query q; labeled memory bank $\mathcal{M}$; retrieval count p; target model $A_j \in \mathcal{A}$; task classifier f

Ensure: Solution y

procedure SOLVE($q, \mathcal{M}, p, A_j$)
  $(c_q, u_q) \leftarrow f(q)$ ▷ LLM-based task classification
  $\ell_j \leftarrow \mathrm{ID}(A_j)$ ▷ target model identity
  $\mathcal{M}_{q,j} \leftarrow \text{Top-}p(\{m \in \mathcal{M} \mid c_m = c_q,\ u_m = u_q,\ \ell_m = \ell_j\})$ ▷ task- and model-aware retrieval
  $y \leftarrow A_j(q, \mathcal{M}_{q,j})$
  return y
end procedure

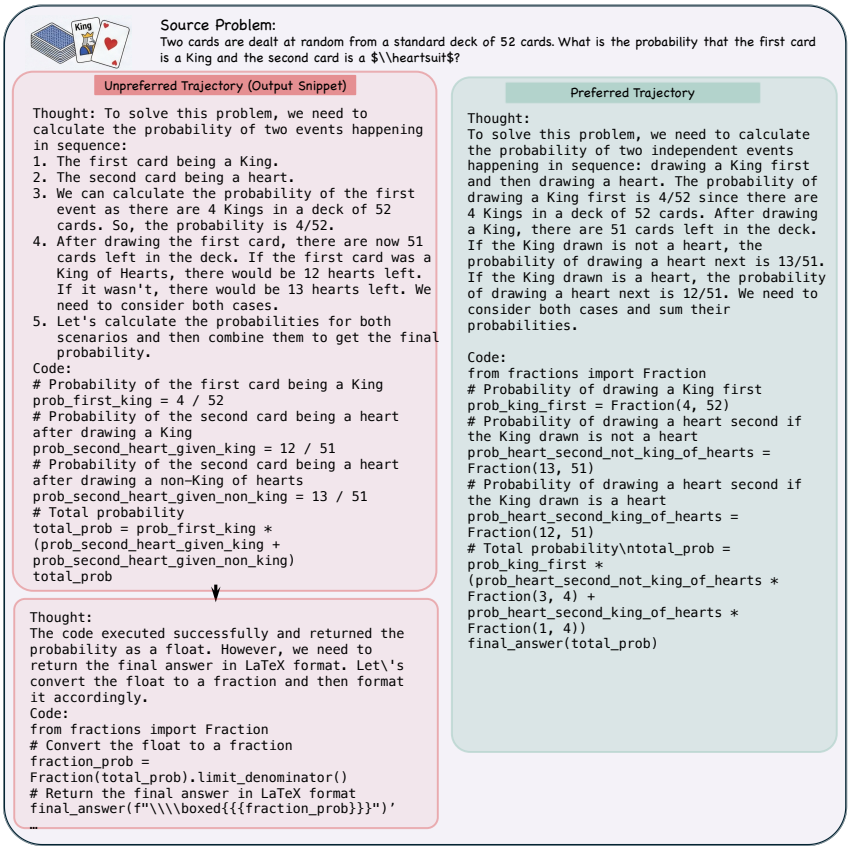

Figure 8: Full trajectory used for memory summarization

F LIMITATIONS.

MEMCOLLAB studies how to construct transferable reasoning memories for heterogeneous models, but practical deployments may require policy-aware governance over memory access and reuse. For example, different models, applications, or user groups may be subject to different safety policies or permission constraints. Incorporating access control and safety filtering into collaborative memory retrieval is an important direction for future work.

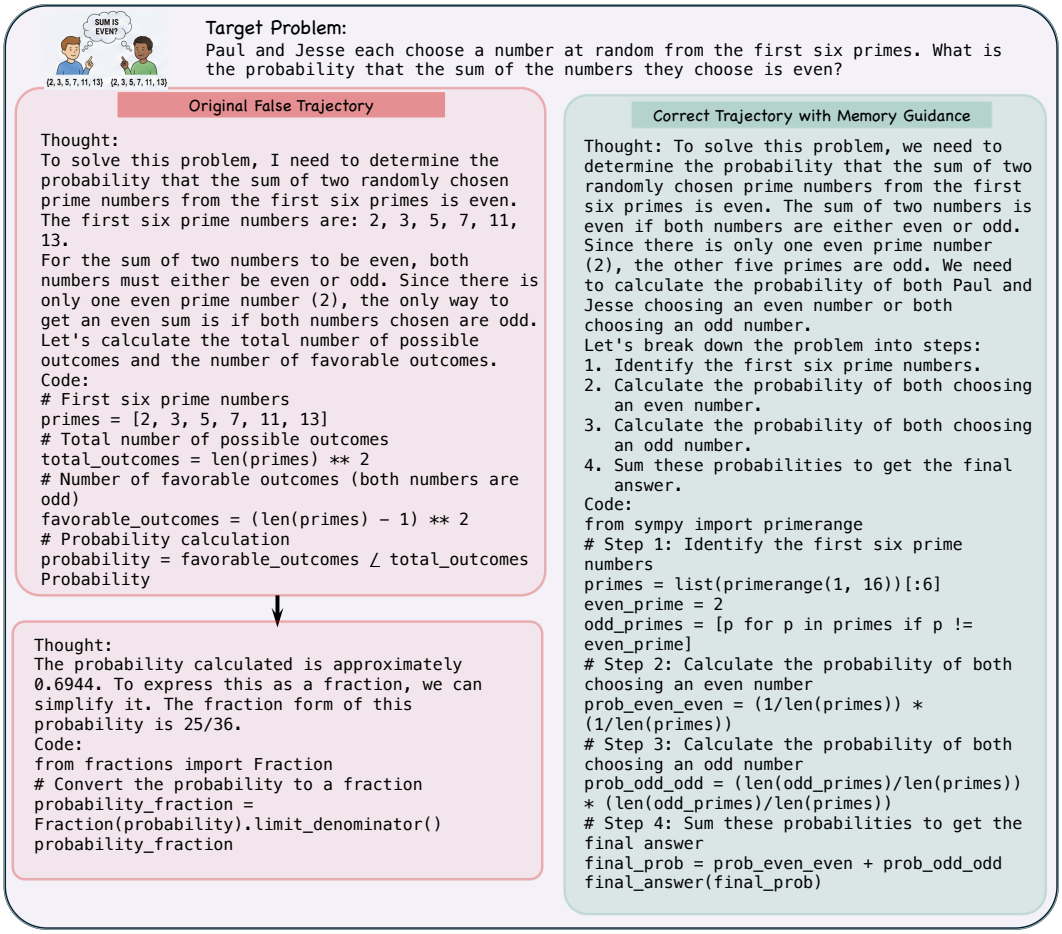

Figure 9: Full trajectory of the memory-augmented inference

Source Question:
Determine the range of parameters for which a triangle exists.
Summarized Un-preferred Trajectory:
Explicitly writes triangle inequalities and solves the feasible range.
Summarized Preferred Trajectory:
Uses the same inequality-based feasibility reasoning before computation.
Memory Guidance:
When determining geometric feasibility, enforce triangle inequalities by converting them into explicit inequality constraints before solving.
Invariants:
Translate geometric existence into inequality constraints.
Variants:
One sets equations first, then simplifies; the other simplifies before imposing constraints.

Source Question:
How many intersection points are formed by a given number of circles?
Summarized Un-preferred Trajectory:
Identifies all diagonal pairs and counts only those that intersect, rather than enumerating points directly.
Summarized Preferred Trajectory:
Counts all circle pairs using $\binom{n}{2}$ and multiplies by the maximum possible intersections per pair.
Memory Guidance: When dealing with combinatorial geometry problems involving intersections, enforce counting principles by first identifying all possible pairs of elements and then filtering based on intersection criteria; avoid direct enumeration without structured counting.
When determining the probability of geometric events, enforce the calculation of favorable outcomes relative to total possible outcomes; avoid assuming uniform distribution without justification.
Invariants: Decompose intersection problems into pairwise interactions with bounded intersection counts.
Variants: One reasons at the geometric-object level (explicit intersection checks), while the other operates directly at the combinatorial level.
One performs conditional filtering; the other applies a closed-form counting bound.

Source Question: Determine a vector onto which multiple vectors have the same projection.
Summarized Un-preferred Trajectory: Sets up dot-product equations equating scalar projections.
Summarized Preferred Trajectory:
Uses the same projection formula and explicitly enforces equality across vectors.
Memory Guidance:
When projecting multiple vectors onto the same vector and obtaining identical projections, enforce equality of scalar projection coefficients rather than treating each projection independently.
Invariants:
Equal projections imply a shared scalar projection constraint.
Variants: One sets equations first, then simplifies; the other simplifies before imposing constraints.

Table 16: Illustration of invariant reasoning patterns captured in the summarized memory entries.

Assistant Task

```
1. Core task summarization: Identify and describe the basic type
and core challenges of the problem, such as classifying it as a
mathematical problem (e.g., solving a quadratic equation), a data
structure problem (e.g., array sorting), or an algorithmic problem
(e.g., search algorithms).
2. Solution steps description: Outline the general solution steps,
including how to define the problem, determine variables, list key
equations or constraints, and choose appropriate solving strategies
and methods.
3. General answer template: Based on the above analysis, propose
a reusable template or approach that can be widely applied to this
type of problem, including possible variables, functions, or class
definitions. For programming problems, provide base classes or
interfaces that can be instantiated to solve specific instances.
Ensure the response is concise and structured so that instance-specific
solutions can be generalized.
```

Reusable Solution Skeleton

```
1. Define symbolic variables or core data structures.
2. Express the main constraints, equations, or problem conditions.
3. Convert the problem into a canonical or solvable form.
4. Apply the appropriate mathematical, algorithmic, or structural
method.
5. Solve for the required variable(s) or compute the target quantity.
6. Verify or validate the result using the original constraints or
expected outputs.
```

Table 17: Prompt template for single-model reasoning extraction.

You are an expert analyst for extracting reusable **REASONING MEMORY** from contrastive multi-step mathematical reasoning trajectories. You are given multiple incorrect or partially correct reasoning traces or predicted outputs.

Your goal is NOT to solve the problems. Your goal is NOT to correct the predictions. Your goal is to extract:
1) reusable failure-aware reasoning constraints, and
2) high-level reasoning strategies that characterize correct multi-step mathematical reasoning, expressed using abstract reasoning rules and high-level mathematical forms (rather than problem-specific calculations).

Each extracted strategy must combine:
- a trigger (when this strategy should be considered), and
- an enforcement rule (what must be enforced or avoided).

ANALYSIS SCOPE
Analyze failures and near-misses at the reasoning level, focusing on:
- explicit formalization of constraints and domains
- correct construction and use of intermediate variables
- alignment between mathematical structure and solution method
- ordering of steps: formalize $\rightarrow$ transform $\rightarrow$ solve $\rightarrow$ verify
- separation between intermediate expressions and final solutions
- handling of extraneous roots, boundary cases, and invalid domains

Use high-level mathematical forms where appropriate, such as:
- constraint systems: $\{f(x) = 0,\ g(x) \geq 0\}$
- canonical equations: $ax^2 + bx + c = 0,\ A \cdot x = b$
- optimization conditions: $\nabla f(x) = 0$
- modular forms: $ax \equiv b \pmod{n}$
- geometric invariants: distance, angle, or area relations

Do NOT focus on:
- surface arithmetic mistakes without structural impact
- numerical simplification details
- stylistic presentation or verbosity
- producing corrected solution chains or final answers

STRICT CONSTRAINTS
- Each strategy MUST be grounded in an observable pattern across trajectories.
- Do NOT introduce speculative or unverifiable strategies.
- Strategies MUST be abstract and reusable across different mathematical domains.

REQUIRED OUTPUT
Each strategy MUST:
- be written as one sentence
- optionally reference high-level mathematical forms or invariants
- follow this format exactly:

When ... , enforce ... ; avoid ...

At least:
- 1 strategy MUST describe failure-prevention or constraint enforcement
- 1 strategy MUST describe high-level reasoning structure or invariants
Do NOT:
- include explanations
- reference specific problems, constants, or numeric values
- include worked examples or derivations
- add any text before or after the list

Table 18: Prompt template for contrastive extraction of reusable mathematical reasoning memory.

You are a mathematical problem classifier. Your task is to assign each math problem to exactly one subcategory within the given main category. You will be provided with:
1. The problem statement.
2. The main category (e.g., Algebra, Precalculus, Geometry, Number Theory).
3. A list of allowed subcategories for this category.

Your output must be exactly one subcategory from the allowed list.

CLASSIFICATION RULES
1. You MUST choose exactly one subcategory from the provided list. You may NOT output "None". You may NOT invent new categories. Only choose from the allowed subcategories provided.
2. If the problem clearly matches a subcategory definition, choose it, even if other superficial features appear.
3. If a problem fits multiple subcategories, choose the most specific one.
4. If a problem matches a special enforcement rule, you MUST apply that rule. These rules override all ambiguity.

SPECIAL ENFORCEMENT RULES
These rules override ambiguity and enforce consistent labeling:

- Polar coordinates, polar-rectangular conversion, parametric coordinates → **Coordinate Systems.**
- Vector dot products, cross products, projections, vector angles, 3D line geometry → **Analytic Geometry (Vectors & Lines).**
- Distance between points, midpoint, slope, basic coordinate geometry → **Analytic Geometry (Distance & Midpoints).**
- Digit-restricted numbers or divisibility based on digit properties → **Modular Arithmetic & Digit Properties.**
- Proper divisors, prime factorization, number of divisors, GCD/LCM → **Divisibility & Prime Factorization.**
- Complex number rotation, geometric interpretation, roots of unity, transformations → **Complex Numbers (Geometry & Transformations).**
- Equations involving radicals or exponentials (including logarithms) → **Radical & Exponential Equations.** *Exception:* Compound interest or exponential growth/decay word problems → **Exponential & Logarithmic Equations.**
- Counting or arrangement problems with restrictions (e.g., adjacency constraints) → **Combinatorial Argument & Constraints.**
- Constructing polynomials from values or missing coefficients → **Polynomial Interpolation.**
- Equations involving functional constraints (e.g., $f(x + y)$, invariance) → **Functional Equations.**

OUTPUT FORMAT
Your response must be ONLY the name of the chosen subcategory. No explanations. No justification. No additional text.

INPUT FORMAT
Problem: {PROBLEM}
Main Category: {CATEGORY}
Allowed Subcategories: {SUBCATEGORY_LIST}

YOUR TASK
Select exactly one subcategory from the list.

Table 19: Prompt template for task classification used in task-aware retrieval.