MILES: Modular Instruction Memory with Learnable Selection for Self-Improving LLM Reasoning

Ruilin Tong **Dong Gong***

School of Computer Science and Engineering
University of New South Wales (UNSW Sydney)
{ruilin.tong, dong.gong}@unsw.edu.au

Abstract

Large language models (LLMs) increasingly improve their reasoning at test time via additional computation, yet most existing works treat each problem in isolation. When problems arrive sequentially, accumulating reusable experience across them can further improve performance. Existing memory-based methods either store whole-solution templates that generalize poorly to novel problems or use heuristic step-level selection that is not optimized for final-answer correctness. Learning selection policies requires large-scale training data and fixed action spaces, making such approaches unsuitable for test-time settings where memory expands incrementally and only limited supervision is available. We propose **MILES** (**M**odular **I**nstruction Memory with **LE**arnable **S**election), a framework that dynamically expands step-wise memory and applies correctness-optimized memory composition under realistic test-time constraints. MILES maintains modular memory units consisting of asymmetric pairs of sub-goal embeddings and sub-instructions, each associated with a learnable selection head. This memory structure enables a coarse-to-fine retrieval mechanism: The coarse level enables memory expansion and collects supervision for training selection heads from confident samples, while the fine stage applies learned selection heads to rerank coarse-level candidates and guide reasoning for uncertain samples. MILES achieves self-improvement by using its own confidence as the learning signal at test time, or given feedback such as labeled data in an offline setting. MILES consistently matches or outperforms prior methods while achieving superior accuracy–efficiency tradeoffs. Extensive experiments demonstrate its effectiveness, robustness, and transferability. Code and resources are available at `https://artificer-ai-lab.github.io/MILES`.

1 Introduction

Large language models (LLMs) demonstrate strong capabilities in complex reasoning and question answering. Recent advances in test-time computation, including chain-of-thought prompting [44], self-consistency [42], and tree search [50, 30], have further improved reasoning performance. In many practical settings, however, problems arrive sequentially rather than in isolation, allowing models to benefit from *accumulating reusable experience* across related queries [49, 28, 24]. Updating model parameters for every new problem is often costly or infeasible, especially for closed or frozen models [52, 53]. A natural alternative is therefore to maintain an *external memory* that the LLM consults at test time while its parameters remain frozen [37, 55, 35]. The central question becomes: *how can such a memory be organized, maintained, and reused so that it actually improves reasoning—and continues to improve as more experience accumulates?*

*D. Gong is the corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Existing memory methods for LLM reasoning store *whole-solution* templates or strategies and retrieve them per problem [49, 37, 55], which are useful when the new problem closely resembles a seen one, but limited when the new problem has novel structure. *Step-level* memory [3, 36, 13] stores smaller units that can be composed flexibly across reasoning steps and across problems with different overall structures, shifting the problem from finding a similar past problem to finding applicable local blocks for the next reasoning step. Among step-level designs, prior work commits to all-text concept/situation pairs [13], pre-formed templates [3], or free-form strategies [35, 28]. A common limitation persists across all of these: *selecting* which unit to apply at the current step is heuristic, via similarity retrieval or LLM prompting, without optimizing the decision against final-answer correctness. Methods that learn selection from feedback [58, 6, 59] require fixed action spaces, external labeled corpora, or fine-tuning, which are not compatible with a memory that grows incrementally from the model's own experience under limited test-time data. Moreover, policies learned offline may not generalize well as test data distribution shifts.

We propose **MILES** (**M**odular **I**nstruction Memory with **LE**arnable **S**election), a framework that enables test-time self-improvement by learning to compose reusable step-level reasoning modules from accumulated reasoning experience while keeping the underlying LLM frozen. To support reusable step-level reasoning, MILES stores memory as modular asymmetric pairs of a *sub-goal* embedding and a corresponding *sub-instruction*. The sub-goal embedding serves as a retrieval key representing the objective of the next reasoning step, while the sub-instruction is natural-language guidance used to condition generation. Unlike raw CoT traces, MILES stores reusable local reasoning modules. Its asymmetric (embedding, text) representation supports scalable retrieval and incremental memory expansion while preserving flexible instruction generation. To enable *learnable memory selection*, MILES associates each memory unit with a lightweight selection head and employs a coarse-to-fine retrieval mechanism at inference. A cheap sub-goal similarity-based retrieval stage (Layer 1) followed by selection-head applicability scoring (Layer 2); Layer 1 expands memory and collects dense step-level supervision from confident trajectories, enabling the selection heads to learn reusable memory-selection patterns from reliable reasoning experience. Layer 2 then applies the learned selection heads to guide reasoning on uncertain samples leveraging memory-selection patterns learned from confident trajectories. As a result, MILES turns step-wise memory composition into a learned, correctness-optimized decision under realistic test-time constraints: incremental memory growth, sparse per-unit data, no ground-truth labels, and no parameter updates. This enables MILES to self-improve at test time, using its own confidence (agreement among sampled answers) as the learning signal. In an offline setting, it can instead learn from given feedback such as labeled data (Table 3). MILES consistently matches or outperforms prior methods across six reasoning benchmarks while achieving superior accuracy–efficiency tradeoffs. Extensive experiments further validate its effectiveness, robustness, and transferability. MILES, with its step-wise modular memory composition, complements prompting and test-time search by reusing experience across problems.

The main contributions of this work are summarized as follows:

- **The MILES framework** for self-improving step-wise memory composition at test time. MILES incrementally builds modular instruction memory from confident reasoning traces and applies the learned memory to uncertain cases, without parameter updates, external supervision, curated training data, or fixed action spaces.
- **A modular asymmetric memory structure** consisting of (sub-goal, sub-instruction) pairs associated with per-unit selection heads, naturally enabling a coarse-to-fine selection mechanism that enables both incremental memory expansion and learnable memory composition.
- **Learnable memory composition.** MILES trains lightweight per-unit selection heads from rollout outcomes on confident samples, enabling self-supervised and correctness-optimized memory composition under limited test-time data while supporting incremental memory expansion.
- **Empirical validation.** MILES consistently matches or outperforms baseline methods across six reasoning benchmarks and four backbones while achieving superior accuracy–efficiency tradeoffs. Further analyses demonstrate its effectiveness, robustness, and transferability.

2 Related work

Test-time LLM reasoning. Chain-of-Thought (CoT) prompting [44] and Self-Consistency (SC) [42] improve LLM reasoning through intermediate reasoning and multi-trajectory aggregation. Structured-search methods such as Tree-of-Thought (ToT) [50], Graph-of-Thought (GoT) [5], rStar

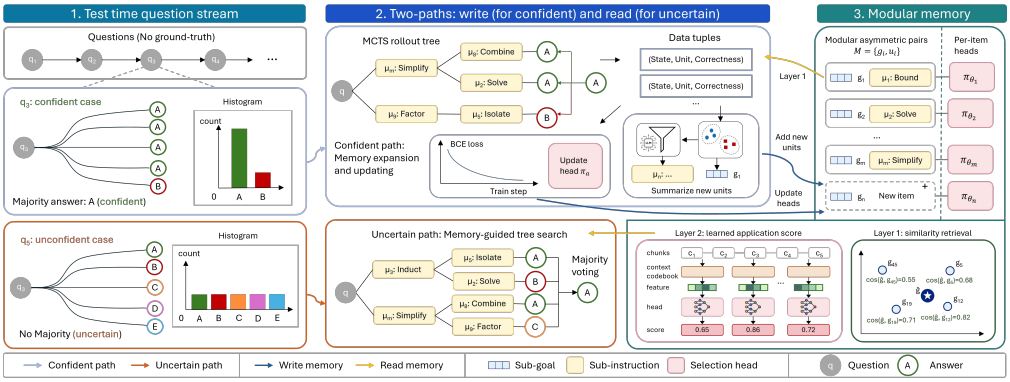


Figure 1: Overview of MILES. MILES improves test-time reasoning by constructing memory and learning memory selection from confident samples, then applying the learned memory to guide reasoning on uncertain samples.

[30], Tree-BoN [31], and DORA [41] further improve reasoning through structured exploration and adaptive search. However, these methods search over memoryless rollouts without reusable cross-problem knowledge. In contrast, MILES integrates learned step-level memory into the search process through per-step sub-instruction selection.

Improve LLM reasoning with memory. Several recent works improve LLM reasoning with reusable memory derived from past experience. Buffer-of-Thought [49], Dynamic CheatSheet [37], and ACE [55] maintain and retrieve problem-solving strategies during test time. A position paper [9] argues that modular memory is key to developing continually adaptable agents. Modular parametric memory designs are used to reduce forgetting in continual learning [25, 57]. ArcMemo [13] and Retrieval-of-Thought [3] introduce step-level or template-based [48] reasoning memory. However, these methods rely on heuristic memory selection via similarity or prompting rather than optimizing selection for final-answer correctness. Beyond reusable memory construction, recent works improve reasoning by updating memory or retrieval mechanisms using feedback. ICPI [6], Memento [59], and DynaAct [58] learn action utilities or value estimates from past trajectories, while ReasoningBank [28], EGUR [36], and BCI [8] iteratively refine reusable strategies through self-feedback. ASCENT [24] trains agents online at test time by distilling verified experience into fast weights. These methods typically rely on fixed action spaces, external labeled data, or parameter updates, making them not compatible with incrementally growing, *frozen-LLM* test-time memory. In contrast, MILES incrementally expands its memory while learning reusable memory-selection patterns from limited test-time data.

3 The Proposed Method: MILES

3.1 Problem formulation and setting

Test-time LLM reasoning. In many practical settings, problems arrive sequentially. We consider a pretrained large language model (LLM) that solves a question q by generating a reasoning trajectory $e_1, e_2, \ldots, e_T$ followed by a final answer y. We define the reasoning state after t steps as $x_t = (q, e_1, \ldots, e_t)$ with $x_0 = q$, and denote the step-level generation distribution by $p(e \mid x)$. Let y^* denote the ground-truth answer and $r \in \{0, 1\}$ the binary correctness indicator, where $r = 1$ if $y = y^*$.

Test-time improvement via external memory. In a stream of test questions, accumulating reusable problem-solving experience into an *external memory* $\mathcal{M}$ allows past experience to benefit subsequent reasoning. The LLM parameters remain frozen throughout, so all learning in MILES is performed by lightweight auxiliary modules built on top of the fixed generation model $p(\cdot \mid \cdot)$.

3.2 Overview of MILES

Our goal is to improve test-time reasoning by augmenting a frozen LLM with an external step-wise memory built from the model's own reasoning experience. MILES maintains a memory $\mathcal{M}$ of modular (sub-goal embedding, sub-instruction text) units with lightweight per-unit selection heads. As the model processes a stream of test problems, MILES operates in two modes: (1) For confident samples, it incrementally expands memory and learns selection heads from rollout outcomes, enabling

the system to accumulate reusable reasoning experience over time. (2) For uncertain samples, it retrieves and reranks relevant memory units through a coarse-to-fine selection mechanism to guide reasoning and improve accuracy. Figure 1 illustrates the overall framework of MILES. The remainder of this section details the four core components that realize this design.

We first define and construct modular memory units in Section 3.3. Building on this memory structure, Section 3.4 introduces a coarse-to-fine selection mechanism that supports both memory expansion and training-data collection for the selection heads. Based on the collected supervision, Section 3.5 presents the learning procedure for the lightweight selection heads under limited data and incrementally expanding memory. Finally, Section 3.6 integrates these components into a unified test-time pipeline that expands memory from confident samples and applies memory-guided tree search to uncertain samples.

3.3 Step-wise memory unit: (sub-goal, sub-instruction) pairs

A memory unit is an asymmetric pair (g_i, u_i), where the *sub-goal* $g_i \in \mathbb{R}^d$ is an embedding that summarizes the objective the sub-instruction is intended to achieve, and the *sub-instruction* u_i is a natural-language directive used to condition generation when the unit is applied. The memory is defined as $\mathcal{M} = \{(g_i, u_i)\}_{i=1}^{N}$, where N grows over time as new reasoning experience is accumulated. We use a_i to denote the action of applying unit (g_i, u_i). Each unit is associated with a learnable selection head that predicts the applicability based on previous experience.

We formulate that the application of a sub-instruction depends on two factors: the objective of the next reasoning step and the conditions in the current context. The sub-goal captures the first factor and serves as a retrieval key for determining whether a sub-instruction is applicable to the current reasoning state. The learned selection heads further identify relevant conditions in the reasoning state and predict applicability scores to rerank memory units (Section 3.4). We organize memory as a collection of *modular* units, enabling *incremental* test-time expansion: new units are identified based on sub-goal similarity (Section 3.4), and their corresponding selection heads (Section 3.5) can be added without retraining existing components, allowing memory to grow continuously with the test stream.

Memory unit summarization. Given a set of (reasoning step, step-goal) tuples collected from confident reasoning trajectories, where the step-goal describes the objective of the reasoning step and is generated by prompting the LLM, we embed the reasoning steps and step-goals using an embedding model and cluster them using k-means with the following joint distance:

$$d(\mathbf{s}_1, \mathbf{o}_1; \mathbf{s}_2, \mathbf{o}_2) \;= 1 - \cos(\mathbf{s}_1, \mathbf{s}_2) \;+\; \alpha_{\text{sum}} \left[1 - \cos(\mathbf{o}_1, \mathbf{o}_2)\right], \tag{1}$$

where $\mathbf{s}_{1,2}$ and $\mathbf{o}_{1,2}$ denote step and step-goal embeddings, and α_{sum} balances the two terms. Within each cluster, the LLM summarizes a generalized sub-instruction u_i from reasoning steps, while the average step-goal embedding defines the sub-goal g_i of the new memory unit. Summarizing sub-instructions from similar reasoning steps promotes their generalizability, while pairing each reasoning step with its corresponding step-goal aims to ensure that the resulting sub-instruction can achieve the intended step-goal.

We also store the mean intra-cluster distance $\overline{d}_i$ to define the per-unit *effective threshold* $R_{\text{eff}}^{(i)}$ (Appendix D.1). All embeddings of reasoning steps, step-goals, sub-goals, and context chunks are produced by the same frozen embedding model, as specified in Section 4.1.

3.4 Coarse-to-fine memory retrieval and selection

Built upon sub-goal embeddings and selection heads, we propose a two-layer *coarse-to-fine memory retrieval and selection* mechanism: a lightweight similarity-based stage, followed by a learned selection head that scores each candidate using features of the full reasoning state.

Layer 1: coarse retrieval based on sub-goal similarity. We first prompt the LLM to generate one or more candidate step-goals for the next reasoning step and compute their embeddings using the embedding model. For each candidate step-goal embedding $\hat{g}$, we compute the cosine similarity $\cos(\hat{g}, g_i)$ between $\hat{g}$ and each stored sub-goal g_i in $\mathcal{M}$, and retrieve the most similar memory units. To avoid retrieving irrelevant units, we apply a per-unit *effective threshold* $R_{\text{eff}}^{(i)} = \alpha_{\text{eff}} \cdot \overline{d}_i$; units with distance exceeds the threshold are excluded. If no unit matches a candidate step-goal, the LLM directly generates the next reasoning step, and the resulting (step-goal, reasoning step) pair is stored as a seed for incremental memory expansion, gradually expanding coverage over a broader range of

conditions in the step-goal embedding space. Layer 1 selects memory units whose sub-instructions address step-goals most similar to that of the next step, Layer 1 thus selects by semantic relevance.

Layer 2: learned applicability scoring. Layer 1 does not incorporate selection knowledge learned from previous outcome feedback. Layer 2 addresses this limitation with a learned per-unit *memory selection head* $\pi_{\boldsymbol{\theta}_i}(a_i \mid x)$ that scores the applicability of the i-th unit in state x. Specifically, we first identify contextual patterns from the input context and then use the selection heads to predict applicability scores based on these patterns. To identify contextual patterns most relevant to the applicability of a_i and to obtain a fixed-length representation of a variable-length reasoning state x, we pre-compute a small set of *context prototypes* $\{\mathbf{q}_{i1}, \ldots, \mathbf{q}_{in}\}$ by clustering embeddings of stored step-goals and contextual conditions from previous applications of the i-th memory unit; the full set $\{\mathbf{q}_{ij}\}$ forms a *context codebook* (Appendix E). Given x, we split it into overlapping chunks with embeddings $\mathbf{c}_1, \ldots, \mathbf{c}_T$ and apply the codebook via max-similarity pooling:

$$\mathbf{m}_i(x) \;=\; [s_{i1}, s_{i2}, \ldots, s_{in}], \qquad s_{ij} \;=\; \max_{k \in [1,T]} \cos(\mathbf{q}_{ij}, \mathbf{c}_k), \tag{2}$$

where $\mathbf{m}_i(x) \in \mathbb{R}^n$ is the context feature vector. The max-similarity construction gives a location-invariant fingerprint of the reasoning state with respect to each prototype, regardless of how long x becomes. Based on the context feature vector, each per-unit selection head scores the applicability of the i-th unit at state x: $\pi_{\boldsymbol{\theta}_i}(a_i \mid x) = f_{\boldsymbol{\theta}_i}\big(\mathbf{m}_i(x)\big) \in [0, 1]$. Each selection head is implemented as a lightweight one-vs-rest binary classifier, as detailed in Section 3.5. Layer 2 reranks memory units using selection heads trained on previous experience, enabling selection based on learned applicability.

Combined score. To balance semantic relevance (Layer 1) and learned applicability (Layer 2), we combine the Layer 1 similarity score with the Layer 2 applicability score:

$$\text{score}(a_i, x) \;=\; (1 - \beta_{\text{c2f}}) \cdot \cos(\hat{g}, g_i) \;+\; \beta_{\text{c2f}} \cdot \pi_{\boldsymbol{\theta}_i}(a_i \mid x), \tag{3}$$

where $\beta_{\text{c2f}} \in [0, 1]$ controls the relative weight of the two layers. The memory unit that passes Layer 1 filtering and achieves the highest combined score is applied in the next reasoning step.

Train and apply selection heads via two-layer selection. Layer 1 selection is used on confident samples to collect sub-instruction application trajectories for training the selection heads. This enables the selection heads to learn when each sub-instruction should be applied from the LLM's confident reasoning trajectories. For uncertain samples, Layer 1 selection first retrieves candidate memory units, after which Layer 2 selection uses the trained heads to rerank candidate sub-instructions, allowing the model to leverage knowledge learned from confident cases to guide reasoning under uncertainty.

3.5 Learning the memory selection heads

To update the selection heads for improved performance. We formulate memory selection as a sequential decision-making problem, where taking action a_i (selecting the i-th memory unit) at reasoning state x should maximize downstream final-answer correctness. We first derive the optimal selection policy under a KL regularization constraint, then approximate it with a lightweight per-unit classifier suitable for incremental test-time learning. Training data is collected from confident samples using the Layer 1 selection.

Selection objective. We formalize step-wise memory selection as a sequential decision-making problem and model the choice of which action a_i to apply at state x as a stochastic *selection policy* $\pi(a_i \mid x)$ over the units in $\mathcal{M}$. Let $p(\cdot \mid \cdot)$ denote the original LLM generation distribution and $p_m(\cdot \mid \cdot)$ the memory-guided generation distribution. Our objective is to maximize the probability of producing a correct final answer under memory guidance:

$$\max_{\pi(a_i|x)} p_m(y^*|x) = \sum\nolimits_i \pi(a_i \mid x) \cdot V(x, a_i), \quad V(x, a_i) \;=\; \mathbb{E}_{p(e|x,a_i)}\big[\mathbb{E}_{p_m(y|e,x)}[r]\big], \tag{4}$$

where r is a binary correctness indicator, e denotes the next reasoning step generated under the guidance of the sub-instruction associated with the i-th memory unit, and y is the resulting final answer. The full derivation is provided in Appendix A.1.

Closed-form optimum under a KL constraint. Given trajectories collected by the Layer 1 data-collection distribution $\pi_{\text{sim}}(a_i \mid x)$, augmenting Eq. (4) with a KL-divergence penalty $\beta_{\text{KL}}\,\text{KL}\big(\pi(\cdot \mid x) \,\|\, \pi_{\text{sim}}(\cdot \mid x)\big)$ yields the following closed-form optimum [29]:

$$\pi^*(a_i \mid x) \;=\; \frac{1}{Z(x)}\, \pi_{\text{sim}}(a_i \mid x) \cdot \exp\big(V(x, a_i)/\beta_{\text{KL}}\big), \tag{5}$$

Table 1: Final-answer accuracy (%) on six reasoning benchmarks for four frozen backbones. MILES (rightmost column) matches or exceeds baseline works on all rows; per-row best in bold.

Dataset	Model	ZS-CoT	SC	DC	BoT	AM	RB	**Ours**
MATH-500	GPT-4.1-mini	87.33 (1.52)	89.20 (1.31)	84.53 (4.10)	87.47 (1.52)	83.47 (0.76)	84.67 (3.04)	**92.27 (1.03)**
	GPT-4.1	84.60 (5.60)	87.00 (2.28)	84.30 (2.73)	85.80 (2.48)	85.53 (0.29)	82.80 (0.86)	**91.67 (0.29)**
AIME 2024	GPT-4.1-mini	48.67 (3.02)	59.67 (1.76)	62.33 (2.26)	52.67 (5.60)	42.00 (2.56)	47.33 (2.93)	**66.36 (1.38)**
	GPT-4.1	40.67 (4.02)	46.67 (2.75)	52.00 (3.41)	48.34 (1.26)	48.00 (3.41)	51.33 (4.08)	**54.66 (2.01)**
	GPT-OSS-20B	86.67 (2.75)	93.33 (1.59)	39.33 (2.46)	53.34 (2.51)	72.66 (3.33)	70.67 (5.13)	**94.00 (1.88)**
	Qwen3-30B-Instruct	67.33 (1.88)	80.00 (3.89)	28.00 (3.02)	66.67 (2.51)	71.33 (2.01)	70.00 (2.25)	**85.33 (1.23)**
AIME 2025	GPT-4.1-mini	44.67 (2.56)	53.67 (2.38)	59.33 (3.14)	41.37 (3.62)	31.33 (2.56)	40.00 (5.73)	**60.33 (1.35)**
	GPT-4.1	34.00 (2.93)	38.00 (1.23)	**47.33 (2.46)**	30.67 (2.46)	36.00 (3.69)	36.67 (2.75)	42.67 (1.88)
	GPT-OSS-20B	84.67 (3.41)	88.67 (1.23)	31.33 (4.93)	45.00 (8.80)	68.00 (3.02)	71.33 (2.01)	**92.66 (1.00)**
	Qwen3-30B-Instruct	56.67 (1.59)	66.67 (2.25)	30.00 (2.25)	46.67 (5.03)	63.33 (3.90)	65.33 (1.23)	**72.66 (1.01)**
GPQA -Diamond	GPT-4.1-mini	70.71 (2.18)	72.02 (2.70)	64.82 (3.15)	66.50 (4.40)	67.34 (1.91)	66.67 (3.32)	**73.06 (1.91)**
	GPT-4.1	68.44 (3.92)	69.19 (3.32)	62.97 (7.24)	62.80 (3.63)	68.18 (3.76)	69.19 (3.76)	**70.88 (0.72)**
	GPT-OSS-20B	69.36 (5.08)	71.05 (2.90)	61.95 (0.72)	37.20 (0.72)	64.98 (5.94)	62.80 (4.41)	**74.58 (2.61)**
	Qwen3-30B-Instruct	70.03 (0.72)	72.89 (1.45)	42.76 (0.73)	68.01 (0.72)	70.37 (6.90)	70.03 (1.92)	**73.74 (1.25)**
MMLU-Pro Physics	GPT-4.1-mini	82.33 (2.59)	83.00 (1.24)	83.50 (1.24)	83.67 (1.50)	81.17 (3.79)	80.67 (1.90)	**84.83 (1.90)**
	GPT-4.1	80.50 (2.15)	82.83 (1.90)	75.83 (9.32)	83.17 (3.59)	82.83 (2.86)	83.00 (2.48)	**85.00 (1.24)**
	GPT-OSS-20B	84.67 (2.59)	85.83 (1.90)	80.33 (5.74)	43.00 (2.48)	81.33 (2.59)	79.83 (5.02)	**86.83 (0.72)**
	Qwen3-30B-Instruct	87.33 (0.72)	**88.50 (3.73)**	64.33 (1.90)	83.83 (3.52)	83.17 (1.90)	85.50 (3.73)	**88.50 (1.24)**
MMLU-Pro Engineering	GPT-4.1-mini	67.67 (4.70)	72.33 (3.13)	68.67 (5.02)	71.50 (4.30)	61.83 (5.17)	61.17 (0.72)	**75.17 (1.89)**
	GPT-4.1	73.67 (3.99)	78.50 (1.24)	67.67 (7.05)	72.17 (4.46)	68.17 (1.43)	69.67 (1.90)	**80.00 (1.24)**
	GPT-OSS-20B	67.17 (9.16)	68.67 (4.90)	62.00 (7.32)	38.17 (5.42)	66.17 (3.79)	62.17 (4.36)	**71.00 (1.24)**
	Qwen3-30B-Instruct	74.67 (1.90)	77.17 (1.90)	52.50 (1.24)	75.33 (0.72)	79.00 (5.69)	75.50 (2.15)	**80.83 (3.73)**

where $\beta_{\text{KL}} > 0$ is the KL coefficient and $Z(x)$ is a normalization constant ensuring that $\pi^*(\cdot \mid x)$ is a valid distribution. A detailed derivation is provided in Appendix A.2. The closed-form optimum characterizes the desired sub-instruction selection behavior but does not provide a practical mechanism for generalization beyond the sampled trajectories. We therefore train lightweight selection heads to approximate $\pi^*(\cdot \mid x)$, enabling effective memory-selection patterns learned from rollout outcomes to be transferred to unseen reasoning states.

From the closed-form optimum to a tractable surrogate. Directly approximating the optimal policy in Eq. (5) is impractical because π_{sim} lacks a tractable density and rollout-based value estimates are noisy. We therefore approximate the policy by training one-vs-rest binary classifiers on thresholded correctness labels, encouraging the selection heads to imitate successful decisions sampled from $\pi_{\text{sim}}(a_i \mid x)$. This per-unit design supports incremental memory expansion without retraining existing heads. Further discussion is provided in Appendix A.3.

Per-unit binary cross-entropy training. For each memory unit, we train its selection head $\pi_{\boldsymbol{\theta}_i}(a_i \mid x)$ via binary cross-entropy on (state, label) pairs collected from MCTS rollouts (Section 3.6):

$$\ell_i(x, \boldsymbol{\theta}_i; r_i) \;=\; -\left[\, r_i \cdot \log \pi_{\boldsymbol{\theta}_i}(a_i \mid x) \;+\; (1 - r_i) \cdot \log\!\big(1 - \pi_{\boldsymbol{\theta}_i}(a_i \mid x)\big) \,\right]. \tag{6}$$

In the one-vs-rest setting, $r_i \in \{0, 1\}$ denotes the binary label for action a_i. Positive examples for the i-th unit are states x at which applying a_i led to a correct final answer. Negative examples for the i-th unit include: (i) states where applying a_i leads to an incorrect answer, and (ii) states where a different unit $i' \neq i$ led to a correct answer. This dense step-level supervision enables effective learning under limited-data in test time. A detailed comparison with prior optimization methods is presented in Appendix A.3.

3.6 Incremental test-time memory construction and inference

In this section, we integrate the previous components into a unified pipeline, including: (1) incrementally constructing memory and learning sub-instruction selection from confident samples, and (2) applying memory-guided reasoning to uncertain samples.

Confidence-based routing of test samples. In the test-time setting, samples arrive sequentially without ground-truth labels. For each test sample q, we generate K independent LLM responses and check whether they agree on a final answer. Samples whose responses converge to a consistent answer are treated as *confident samples*: the majority-vote answer serves as a pseudo-label for evaluating MCTS rollouts, and the resulting trajectories are used to expand memory and train the selection heads. Samples with inconsistent responses are instead routed to memory-guided tree search using the learned heads. Intuitively, confident samples provide a reliable supervisory signal, whereas the LLM should leverage prior experience to guide reasoning when it is uncertain about a sample.

Data collection and memory expansion via MCTS on confident samples. For each confident sample we run MCTS over the reasoning tree to collect training data for the selection heads. At each tree node (a reasoning state x), we use the Layer 1 sub-goal similarity to sample candidate memory units to apply, run the LLM to expand the chosen step, and propagate correctness signals from rollouts evaluated against the pseudo-label. The detailed MCTS procedure is presented in Appendix C.1.

The collected (step-goal, reasoning step) pairs (Section 3.4) are summarized into new memory units as described in Section 3.3. Data collection uses only Layer 1 similarity-based sampling via π_{sim} to avoid bias toward the current selection heads and preserve trajectory diversity. The resulting tuples are stored in per-unit datasets $\mathcal{D}_i = \{(x_j, r_{i,j})\}$ for training the corresponding selection heads $\pi_{\boldsymbol{\theta}_i}$ with the BCE loss in Eq. (6). The detailed training procedure is described in Appendix E.

Memory-guided tree search for uncertain samples. For samples whose initial K responses are inconsistent, we apply a multi-stage memory-guided tree search similar to [31]. At each stage, a sub-context is sampled from existing reasoning chains, candidate step-goals are generated, and memory units are retrieved and ranked using the combined score in Eq. (3). The top-ranked unit is then applied to expand the reasoning trajectory, and the resulting branch is added to the response pool. As search progresses, longer sub-contexts are used to guide later reasoning steps. The final answer is determined by majority voting over all generated responses. Details of the memory-guided tree search procedure are provided in Appendix C.2.

4 Experiments

4.1 Main results on LLM reasoning: accuracy, efficiency, and transferability

Datasets and models evaluation. We evaluate on six reasoning benchmarks spanning competition mathematics and graduate-level academic exams: MATH-500 (500 samples) [12], AIME 2024 (30 samples), AIME 2025 (30 samples), GPQA-Diamond (198 samples) [33], and the Physics and Engineering subsets of MMLU-Pro [43]. Following [37], we randomly sample 200 instances from each MMLU-Pro subset and evaluate the remaining datasets in full. We evaluate MILES on four backbone LLMs spanning proprietary and open-weight families: GPT-4.1, GPT-4.1-mini, Qwen3-30B-Instruct [47], and GPT-OSS-20B [2]. For fair comparison, the same backbone is used for both reasoning and memory construction across all methods within each row. All embeddings in MILES are computed using the embedding model `text-embedding-3-small`. Final-answer accuracy is used as the evaluation metric.

Evaluation: In the test-time setting, samples arrive sequentially, and memory is constructed only from solutions to previously processed samples. Predictions for previous samples cannot be revised and all methods are evaluated on the same sequence of questions. We evaluate final-answer accuracy across multiple runs, with numbers in parentheses indicating 95% confidence intervals.

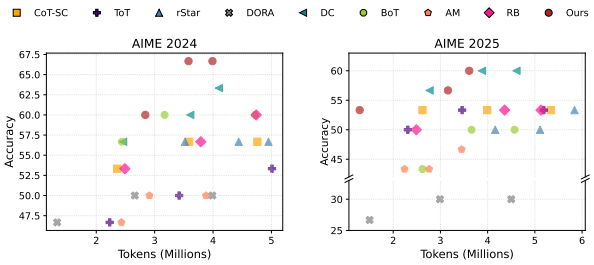


Figure 2: Accuracy vs. response-token cost on AIME 2024 and AIME 2025 with GPT-4.1-mini, comparing MILES with test-time scaling and memory-based baselines. MILES reaches the highest accuracy on AIME 2024 and ties the strongest baseline, DC, on AIME 2025.

Baselines. We compare MILES against test-time memory methods, including Buffer-of-Thoughts (BoT) [49] and Dynamic CheatSheet (DC) [37], ArcMemo (AM) [13] and ReasoningBank (RB) [28], as well as two standard baselines: zero-shot Chain-of-Thought (ZS-CoT) [19] and Self-Consistency (SC) [42]. Additional comparisons with memoryless test-time scaling methods are provided in Appendix F.

MILES configuration. Across all experiments, MILES uses the same component design described in Section 3, with per-unit selection heads implemented as an ensemble of 10 independently trained 3-layer MLPs for robust inference. Additional implementation details are provided in Appendix I.

Table 1 shows that MILES matches or outperforms baseline works across all six benchmarks and four backbones, with consistent gains on both math (MATH-500, AIME 2024–2025) and academic (GPQA-Diamond, MMLU-Pro Physics/Engineering) datasets. The improvements are especially

pronounced on smaller open-weight backbones (GPT-OSS-20B and Qwen3-30B-Instruct), where the sample-level memory baselines BoT [49] and DC [37] fail to outperform zero-shot CoT due to overly complex prompt. In near-saturated settings (e.g., GPT-OSS-20B on AIME 2024), MILES matches self-consistency without degradation, demonstrating its effectiveness and robustness. We note that the strong performance of DC on the AIME datasets mainly comes from its use of coding; additional details are provided in Appendix F.

Performance under different computation budget. We probe how accuracy scales with test-time compute by plotting accuracy against response-token cost on AIME 2024 and AIME 2025 with GPT-4.1-mini, comparing MILES to representative memoryless test-time scaling methods: Self-Consistency [42], Tree-of-Thoughts [50], rStar [30], and DORA [41]. As shown in Figure 2, MILES benefits more from increased computation budgets by constructing additional memory units (Section 4.2), dominating the accuracy/token frontier and continuing to improve as the budget grows, whereas the baselines quickly plateau. This highlights the effectiveness of reusing cross-problem memory, which captures knowledge that per-problem scaling alone cannot.

Table 2: Cross-model transfer on AIME 2024 and AIME 2025 with GPT-4.1-mini as the reasoning backbone. Memory built from rollouts of an auxiliary model (Qwen3-30B-Instruct, Qwen3-4B-Instruct, or GPT-4.1-mini itself) improves over the no-auxiliary baseline (top row in each block).

Dataset	Auxiliary model	Accuracy
AIME 2024	no-auxiliary model	59.33 (1.00)
	Qwen3-30B-Instruct	**68.00** (**2.01**)
	Qwen3-4B-Instruct	64.67 (1.23)
	GPT-4.1-mini	66.36 (1.38)
AIME 2025	no-auxiliary model	53.33 (1.59)
	Qwen3-30B-Instruct	60.00 (1.12)
	Qwen3-4B-Instruct	57.33 (1.88)
	GPT-4.1-mini	**60.33** (**1.35**)

Cross-model transfer. Because MILES operates on textual memory representations, memory construction need not use the same model as the reasoning backbone. We evaluate cross-model transfer on AIME 2024 and AIME 2025 using GPT-4.1-mini as the reasoning backbone: the backbone generates the first 10 self-consistency rollouts for routing, while an *auxiliary* model generates the next 40 rollouts for memory construction. Table 2 shows that auxiliary trajectories from Qwen3-30B-Instruct, Qwen3-4B-Instruct, and GPT-4.1-mini itself all improve over the no-auxiliary baseline, demonstrating that memory constructed by open-weight auxiliary models transfers effectively to a stronger frozen backbone.

Offline memory construction with training set. When a separate labeled set is available, memory can be built ahead of time and memory-guided tree search applied only to the most uncertain test samples (Section 3.6). We evaluate our method on AIME 2025 using memory constructed from AIME 2020–2024, and on GPQA-Diamond [33] using memory constructed from the non-Diamond GPQA split, reporting accuracy on the 13 and 31 most uncertain test samples, respectively. Table 3 shows that memory pre-built by either Qwen3-30B-Instruct or GPT-4.1-mini itself substantially improves over the no-memory baseline. These results demonstrate that MILES learns reusable problem-solving knowledge that transfers across different models and samples within the same domain.

Table 3: Memory pre-built by an auxiliary model on a separate training set with ground-truth labels (AIME 2020–2024 for AIME 2025; non-Diamond GPQA for GPQA-Diamond), evaluated on the most uncertain test samples with GPT-4.1-mini as the reasoning backbone. Pre-built memory consistently improves over the no-memory baseline (top row in each block).

Dataset	Auxiliary model	Accuracy
AIME 2025	no memory	18.46 (2.84)
	Qwen3-30B-Instruct	28.46 (2.66)
	GPT-4.1-mini	**36.92** (**2.32**)
GPQA-Diamond	no memory	45.97 (2.39)
	Qwen3-30B-Instruct	**56.77** (**1.19**)
	GPT-4.1-mini	54.19 (0.97)

4.2 Ablation study

Effectiveness of each component in two-layer selection. To evaluate the two-layer selection mechanism, we conduct experiments on AIME 2024 and AIME 2025 with GPT-4.1-mini. When both layers are disabled, no memory is constructed and no selection heads are trained. We further compare Layer 2 against two alternative rerankers: (i) *state-similarity reranking*, which scores candidates by the similarity between the current reasoning-state embedding and the average embeddings of previously successful states; and (ii) *LLM-prompt reranking*, which prompts LLM with the current reasoning state and the Layer-1 candidate sub-instructions and asks it to identify the most applicable candidate.

Table 4 shows that combining both layers (Exp 4) yields the best accuracy on both datasets, while disabling either layer (Exp 2–3) reduces performance. Disabling both layers causes accuracy to fall close to the self-consistency baseline. Both alternative rerankers (Exp 5 and Exp 6) outperform the no-reranker baseline (Exp 2). LLM-prompt reranking (Exp 6) slightly outperforms state-similarity reranking (Exp 5), but both perform worse than the learned classifier (Exp 4). This suggests that the learned classifier captures additional information beyond what LLM prompting or state similarity can provide. These results further demonstrate the effectiveness of our two-layer selection mechanism.

Table 4: Ablation study of memory selection components. Incorporating selection heads leads to consistent performance improvements.

Dataset	Exp. ID	Layer 1	Layer 2	Accuracy
AIME 2024	1	✗	✗	59.33 (1.00)
	2	✓	✗	62.66 (1.01)
	3	✗	✓	64.00 (1.88)
	4	✓	✓	**66.36 (1.38)**
	5	✓	State sim.	64.00 (1.01)
	6	✓	LLM rerank	64.67 (2.01)
AIME 2025	1	✗	✗	53.33 (1.59)
	2	✓	✗	55.33 (2.01)
	3	✗	✓	58.00 (1.23)
	4	✓	✓	**60.33 (1.35)**
	5	✓	State sim	56.00 (1.88)
	6	✓	LLM rerank	56.67 (1.60)

We further inspect reasoning trajectories using two metrics: *correct-answer proportion* (fraction of trajectories that produce the correct final answer) and *sub-instruction accuracy* (fraction of sub-instruction applications that lead to a correct final answer), on samples near the correct/incorrect decision boundary where small effects matter most. Figure 3 shows that our full method (Exp 4) performs best on both metrics. The gaps between Exp 4 and Exp 1–3 validate the asymmetric memory and two-layer selection design, as well as between Exp 4 and Exp 5–6 validate the learned heads over alternative rerankers.

Why both selection layers matter. The results in Table 4 demonstrate the importance of combining Layer 1 and Layer 2 for effective memory reuse. With only semantic similarity-based selection (Exp 2), a sub-instruction may be relevant to the current reasoning state but may not lead to a correct final answer when applied. Conversely, relying solely on learned selection heads (Exp 3) cannot effectively filter out semantically unrelated candidates. Combining both layers (Exp 4) achieves the best performance by integrating semantic relevance with feedback from previous experience. The improvements in both sub-instruction accuracy and correct-answer proportion (Figure 3) further support the importance of learning when to apply retrieved knowledge, rather than relying solely on semantic relevance.

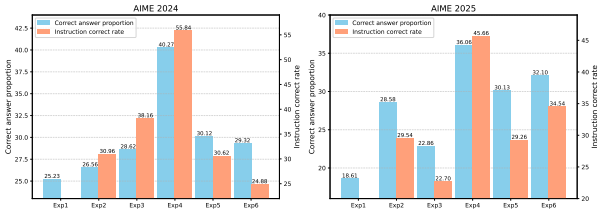


Figure 3: Correct-answer proportion and sub-instruction accuracy on uncertain samples for each experiment in Table 4. Our method (Exp 4) achieves the best performance on both.

Effect of MCTS rollouts. MILES summarizes sub-instructions in memory and collects classifier-training data via MCTS on confident samples; we therefore study how the rollout budget affects both the resulting memory size and the final accuracy. We sweep rollouts $\in \{20, 30, 40\}$ on AIME 2024 and AIME 2025 with GPT-4.1-mini and report results in Table 5.

Table 5: Effect of the per-confident-sample MCTS rollout budget on memory size and accuracy, evaluated on AIME 2024 and AIME 2025 with GPT-4.1-mini. More rollouts produce a richer memory and higher accuracy.

Dataset	Rollouts	Mem. size	Accuracy
AIME 2024	20	9	60.67 (1.00)
	30	15	66.00 (1.01)
	40	20	66.36 (1.38)
AIME 2025	20	10	51.33 (2.01)
	30	19	58.00 (1.23)
	40	25	60.33 (1.35)

More rollouts yield both a larger memory with more sub-instruction units and higher accuracy. A larger number of sub-instructions corresponds to better coverage of the step-goal embedding space (Section 3.4). These results suggest that improved coverage leads to higher performance, and further demonstrate the effectiveness of our memory selection strategy.

Impact of sample order. Because MILES updates memory online from confident samples (Section 3.6), sample order affects how knowledge is accumulated and reused. To evaluate the robustness of MILES under different sample orders, we conduct experiments on AIME 2024 and AIME 2025 with GPT-4.1-mini under three sample orders: the default question-index order, easiest-to-hardest, and hardest-to-easiest, using two MCTS rollout budgets (20 and 40 per confident sample). Sample difficulty is defined by the proportion of correct answers across self-consistency rollouts. Results are in Table 6.

Under the limited rollout budget (20 per sample), the easy-to-hard order achieves the highest accuracy on both datasets, suggesting that solving easier samples first allows MILES to accumulate useful memory before encountering harder ones. With 40 rollouts, the performance gap largely disappears, indicating that the method becomes effectively order-insensitive once sufficient confident-sample data has been collected.

Table 6: Performance of our method on the AIME 2024 and AIME 2025 datasets under different sample orders and MCTS rollout budgets using GPT-4.1-mini.

Dataset	Sample order	Accuracy	
		20 rollouts	40 rollouts
AIME 2024	default	60.67 (1.00)	66.36 (1.38)
	easy-to-hard	66.00 (1.01)	66.34 (0.76)
	hard-to-easy	64.00 (2.46)	67.00 (0.75)
AIME 2025	default	51.33 (2.01)	60.33 (1.35)
	easy-to-hard	56.67 (1.59)	59.67 (0.75)
	hard-to-easy	55.33 (2.01)	63.00 (1.35)

Experience accumulation and robustness. Tables 5 and 6 reveal two related effects of increasing the rollout budget. First, additional rollouts allow MILES to construct more memory units, improving the coverage of reusable reasoning experience. Second, increasing the rollout budget reduces sensitivity to the order in which samples are processed. Under a limited budget, processing easier samples first can provide useful experience for subsequent problems. With sufficient rollouts, MILES becomes less dependent on this ordering. These observations suggest that richer experience accumulation can improve both reasoning performance and robustness in the online setting.

Additional design ablations and analyses are provided in Appendix F.

5 Conclusion

In this work, we propose MILES, a framework for test-time self-improvement that incrementally accumulates and composes reusable step-wise reasoning experience. By learning memory-selection patterns from confident trajectories and transferring them to uncertain reasoning states, MILES enables correctness-optimized memory composition without parameter updates, external supervision, or fixed action spaces. Extensive experiments across multiple reasoning benchmarks demonstrate its effectiveness, robustness, and transferability. These results suggest that learning reusable memory-selection knowledge is a promising direction for building continually improving reasoning systems.

Limitations and future work. MILES assumes a stream of related problems with reusable reasoning structure, as memory is accumulated from previously processed samples. Consequently, its benefits depend on the available experience and may vary with sample order, as examined in Section 4.2. Moreover, MILES relies on the backbone model's ability to generate reliable pseudo-labels through majority agreement. When the backbone performs poorly, agreement may not indicate correctness, potentially introducing incorrect supervision. Although our experiments demonstrate the reliability of pseudo-labels on the evaluated benchmarks (Appendix F.10), their reliability on more challenging tasks may depend on the capability of the backbone and remains to be investigated.

MILES also relies on the instruction-following capability of the underlying LLM and requires rollout-based data collection, which introduces additional computational overhead. This cost can be reduced by using auxiliary models for trajectory generation (Section 4.1). Future work includes combining MILES with self-refinement methods [28, 35] to improve instruction quality and with hierarchical memory [48, 7] to improve selection efficiency.

Acknowledgments and Disclosure of Funding

This research was partially supported by the Australian Research Council through a Discovery Project grant (DP260103379) and a DECRA Fellowship (DE230101591) awarded to D. Gong.

References

[1] R. Agarwal, A. Singh, L. Zhang, B. Bohnet, L. Rosias, S. Chan, B. Zhang, A. Anand, Z. Abbas, A. Nova, et al. Many-shot in-context learning. *Advances in Neural Information Processing Systems*, 37:76930–76966, 2024.

[2] S. Agarwal, L. Ahmad, J. Ai, S. Altman, A. Applebaum, E. Arbus, R. K. Arora, Y. Bai, B. Baker, H. Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

[3] A. Ahmed, A. A. Khan, A. Ahmad, S. Di, Z. Liu, and A. Anwar. Retrieval-of-thought: Efficient reasoning via reusing thoughts. *arXiv preprint arXiv:2509.21743*, 2025.

[4] T. Anthony, Z. Tian, and D. Barber. Thinking fast and slow with deep learning and tree search. *Advances in neural information processing systems*, 30, 2017.

[5] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 17682–17690, 2024.

[6] E. Brooks, L. A. Walls, R. Lewis, and S. Singh. In-context policy iteration. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

[7] Z. Cai, X. Guo, Y. Pei, J. Feng, J. Su, J. Chen, Y.-Q. Zhang, W.-Y. Ma, M. Wang, and H. Zhou. Flex: Continuous agent evolution via forward learning from experience. *arXiv preprint arXiv:2511.06449*, 2025.

[8] A. Didolkar, N. Ballas, S. Arora, and A. Goyal. Metacognitive reuse: Turning recurring llm reasoning into concise behaviors. *arXiv preprint arXiv:2509.13237*, 2025.

[9] V. Dorovatas, M. Schwerin, A. D. Bagdanov, L. Caccia, A. Carta, L. Charlin, B. Hammer, T. L. Hayes, T. Hess, C. Kanan, D. Kudithipudi, X. Liu, V. Lomonaco, J. Mendez-Mendez, D. Patil, A. Prabhu, E. Ricci, T. Tuytelaars, G. M. van de Ven, L. Wang, J. van de Weijer, J. Choi, M. Mundt, and R. Aljundi. Position: Modular memory is the key to continual learning agents, 2026.

[10] A. Z. Dou, Z. Wan, D. Cui, X. Wang, J. Xiong, H. Lin, C. Tao, S. Yan, and M. Zhang. Enhancing test-time scaling of large language models with hierarchical retrieval-augmented mcts. *arXiv preprint arXiv:2507.05557*, 2025.

[11] Q. Hao, S. Li, J. Yuan, and Y. Li. Rl of thoughts: Navigating llm reasoning with inference-time reinforcement learning. *arXiv preprint arXiv:2505.14140*, 2025.

[12] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[13] M. Ho, C. Si, Z. Feng, F. Yu, Y. Yang, Z. Liu, Z. Hu, and L. Qin. Arcmemo: Abstract reasoning composition with lifelong llm memory. *arXiv preprint arXiv:2509.04439*, 2025.

[14] Y. Hu, Y. Zhao, C. Zhao, and A. Cohan. MCTS-RAG: Enhancing retrieval-augmented generation with Monte Carlo tree search. In C. Christodoulopoulos, T. Chakraborty, C. Rose, and V. Peng, editors, *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 12581–12597, Suzhou, China, Nov. 2025. Association for Computational Linguistics.

[15] T. Jiang, A. An, H. Feng, N. Zhai, H. Li, X. Yu, J. Liu, H. Du, S. Zhang, Z. Yang, J. Huang, Y. Li, Y. Ni, H. Wang, and R. Chen. Chain of mindset: Reasoning with adaptive cognitive modes, 2026.

[16] M. Jin, Q. Yu, D. Shu, H. Zhao, W. Hua, Y. Meng, Y. Zhang, and M. Du. The impact of reasoning step length on large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 1830–1842, 2024.

[17] O. Khattab and M. Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48, 2020.

[18] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[19] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[20] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[21] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[22] Y. Li. Policy guided tree search for enhanced llm reasoning. *arXiv preprint arXiv:2502.06813*, 2025.

[23] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

[24] H. Lu and D. Gong. ASCENT: Online test-time training of long-horizon agents via self-distillation of verified experience. *arXiv preprint arXiv:2610.05303*, 2026.

[25] H. Lu, C. Zhao, M. Xue, L. Yao, K. Moore, and D. Gong. Little by little: Continual learning via incremental mixture of rank-1 associative memory experts. In *International Conference on Machine Learning*. PMLR, 2026.

[26] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

[27] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[28] S. Ouyang, J. Yan, I. Hsu, Y. Chen, K. Jiang, Z. Wang, R. Han, L. T. Le, S. Daruki, X. Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

[29] X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[30] Z. Qi, M. Ma, J. Xu, L. L. Zhang, F. Yang, and M. Yang. Mutual reasoning makes smaller llms stronger problem-solvers. *arXiv preprint arXiv:2408.06195*, 2024.

[31] J. Qiu, Y. Lu, Y. Zeng, J. Guo, J. Geng, H. Wang, K. Huang, Y. Wu, and M. Wang. Treebon: Enhancing inference-time alignment with speculative tree-search and best-of-n sampling. *arXiv preprint arXiv:2410.16033*, 2, 2024.

[32] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

[33] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First conference on language modeling*, 2024.

[34] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[35] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[36] A. Stein, M. Trager, B. Bowman, M. Kleinman, A. Chattopadhyay, W. Xia, and S. Soatto. Experience-guided adaptation of inference-time reasoning strategies. *arXiv preprint arXiv:2511.11519*, 2025.

[37] M. Suzgun, M. Yuksekgonul, F. Bianchi, D. Jurafsky, and J. Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

[38] F. Teng, Q. Shi, Z. Yu, J. Zhang, Y. Luo, C. Wu, and Z. Guo. Atom of thoughts for markov llm test-time scaling. *arXiv preprint arXiv:2502.12018*, 2025.

[39] X. Wan, H. Zhou, R. Sun, H. Nakhost, K. Jiang, and S. Ö. Arık. From few to many: Self-improving many-shot reasoners through iterative optimization and generation. *arXiv preprint arXiv:2502.00330*, 2025.

[40] P. Wang, L. Li, Z. Shao, R. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, and Z. Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2023.

[41] X. Wang, Y. Li, S. Feng, P. Yuan, Y. Zhang, J. Shi, C. Tan, B. Pan, Y. Hu, and K. Li. Every rollout counts: Optimal resource allocation for efficient test-time scaling. *arXiv preprint arXiv:2506.15707*, 2025.

[42] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[43] Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

[44] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[45] J. Wu, M. Feng, S. Zhang, F. Che, Z. Wen, C. Liao, and J. Tao. Beyond examples: High-level automated reasoning paradigm in in-context learning via mcts. *arXiv preprint arXiv:2411.18478*, 2024.

[46] S. M. Xie, A. Raghunathan, P. Liang, and T. Ma. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.

[47] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[48] L. Yang, Z. Yu, B. Cui, and M. Wang. Reasonflux: Hierarchical llm reasoning via scaling thought templates. *arXiv preprint arXiv:2502.06772*, 2025.

[49] L. Yang, Z. Yu, T. Zhang, S. Cao, M. Xu, W. Zhang, J. E. Gonzalez, and B. Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 37:113519–113544, 2024.

[50] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[51] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[52] E. Zelikman, Y. Wu, J. Mu, and N. D. Goodman. Star: Self-taught reasoner bootstrapping reasoning with reasoning. In *Proc. the 36th International Conference on Neural Information Processing Systems*, volume 1126, 2024.

[53] W. Zeng, Y. Huang, L. Zhao, Y. Wang, Z. Shan, and J. He. B-star: Monitoring and balancing exploration and exploitation in self-taught reasoners. *arXiv preprint arXiv:2412.17256*, 2024.

[54] D. Zhang, S. Zhoubian, Z. Hu, Y. Yue, Y. Dong, and J. Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772, 2024.

[55] Q. Zhang, C. Hu, S. Upasani, B. Ma, F. Hong, V. Kamanuru, J. Rainton, C. Wu, M. Ji, H. Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.

[56] Y. Zhang, Z. Qin, Z. Wu, W. Zhang, and S. Deng. Reinforcement fine-tuning for history-aware dense retriever in rag, 2026.

[57] C. Zhao, M. Li, H. Lu, and D. Gong. On token's dilemma: Dynamic MoE with drift-aware token assignment for continual learning of large vision language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

[58] X. Zhao, W. Wu, J. Guan, Q. Li, and L. Kong. Dynaact: Large language model reasoning with dynamic action spaces. *arXiv preprint arXiv:2511.08043*, 2025.

[59] H. Zhou, Y. Chen, S. Guo, X. Yan, K. H. Lee, Z. Wang, K. Y. Lee, G. Zhang, K. Shao, L. Yang, et al. Memento: Fine-tuning llm agents without fine-tuning llms. *arXiv preprint arXiv:2508.16153*, 2025.

A Objective formulation and optimization

A.1 Memory retrieval objective

Inspired by the formulations of prior works [21, 46], we formalize sub-instruction selection as a sequential decision-making problem, where the objective is to maximize the probability of generating the correct solution by optimizing the instruction selection function. At each step, with a_i denoting the action of applying the i-th memory unit, we model its effect on generating the next reasoning step e as

$$p_m(e|x) = \sum_i p(e|x, a_i)\pi(a_i \mid x) = \mathbb{E}_{\pi(a_i|x)}\left[p(e \mid x, a_i)\right], \tag{7}$$

where $\pi(a_i|x)$ denotes the memory selection function conditioned on the reasoning state x, $p(\cdot \mid \cdot)$ is the generation probability of LLM, and $p_m(\cdot \mid \cdot)$ denotes the generation probability conditioned on sub-instructions. The reasoning state x consists of the question q and the reasoning trajectory generated so far.

Under the causal decoding mechanism, subsequent reasoning steps are conditioned on the current reasoning state and the newly generated step. Thus, the probability of generating the final solution y can be written as $p_m(y|x, e)$. By the chain rule of probability,

$$p_m(y, e|x) = p_m(y|e, x) \cdot p_m(e|x), \tag{8}$$

which allows us to express the probability of generating the final solution as

$$\begin{aligned} p_m(y \mid x) &= \int p_m(y, e \mid x)de \\ &= \int p_m(y \mid e, x) \cdot p_m(e \mid x)de \\ &= \int p_m(y \mid e, x) \left(\sum_i p(e \mid x, a_i)\pi(a_i \mid x)\right) de \\ &= \sum_i \left(\pi(a_i \mid x) \int p_m(y \mid e, x) \cdot p(e \mid x, a_i)de\right) \\ &= \sum_i \pi(a_i \mid x) \cdot \mathbb{E}_{p(e|x,a_i)}[p_m(y \mid e, x)]. \end{aligned} \tag{9}$$

Our objective is to maximize the probability of generating the correct final solution y^* by optimizing $\pi(a_i \mid x)$, while keeping the LLM parameters fixed. Therefore, our optimization objective is

$$\max_{\pi(a_i|x)} \; p_m(y^* \mid x) = \sum_i \pi(a_i \mid x) \cdot \mathbb{E}_{p(e|x,a_i)}[p_m(y^* \mid e, x)]. \tag{10}$$

The effect of selecting an instruction at the current step propagates through subsequent reasoning steps and influences the final answer. However, $p_m(y^* \mid e, x)$ is intractable in practice. Instead of explicitly modeling future memory selections, we approximate their impact using the expected correctness of the final answer, following the standard reinforcement learning practice of using rollout returns to evaluate intermediate decisions. Therefore, we propose to approximate $p_m(y^* \mid e, x)$ using a binary correctness indicator $r \in \{0, 1\}$, defined as

$$r = \begin{cases} 1, & y = y^*, \\ 0, & y \neq y^*. \end{cases} \tag{11}$$

Because r follows a Bernoulli distribution with success probability $p_m(y^* \mid e, x)$, we approximate $p_m(y^* \mid e, x)$ by the expectation over r,

$$p_m(y^* \mid e, x) = \mathbb{E}_{p_m(y|e,x)}[r]. \tag{12}$$

This approximation reduces the sequential decision problem to evaluating the expected return of a one-step decision, while implicitly accounting for future reasoning through rollout-based reward signals. Substituting this into the objective in Eq. (10) yields

$$\max_{\pi(a_i|x)} \; p_m(y^* \mid x) = \sum_i \pi(a_i \mid x) \cdot V(x, a_i), \quad V(x, a_i) = \mathbb{E}_{p(e|x,a_i)}\left[\mathbb{E}_{p_m(y|e,x)}[r]\right], \tag{13}$$

where $V(x, a_i)$ represents the expected correctness of the final answer when selecting action a_i under reasoning state x.

A.2 Optimal teacher policy

During data collection, we select sub-instructions based on the similarity between step-goal embeddings and sub-goal. Let this sampling policy be denoted as π_{sim}. Following [29], we optimize a target policy $\pi(a_i \mid x)$ under a KL-regularized objective:

$$\begin{aligned} \arg\max_{\pi(a_i|x)} & \int d_{\pi_{\text{sim}}}(x) \left(\sum_i \pi(a_i \mid x) \cdot V(x, a_i) \right) dx \\ & \text{s.t. } \mathrm{D}_{\text{KL}}(\pi(\cdot \mid x) || \pi_{\text{sim}}(\cdot \mid x)) \leq \epsilon, \quad \forall\, x \\ & \sum_i \pi(a_i \mid x) = 1, \quad \forall\, x. \end{aligned} \tag{14}$$

where $d_{\pi_{\text{sim}}}(x)$ denotes the state distribution induced by π_{sim}, and $\pi(a_i \mid x)$ is the policy to be optimized.

Following the derivation in [29], we form the corresponding Lagrangian:

$$\begin{aligned} \mathcal{L}(\pi, \beta_{\text{KL}}, \alpha_{\text{prob}}) = & \int d_{\pi_{\text{sim}}}(x) \left(\sum_i \pi(a_i \mid x) \cdot V(x, a_i) \right) dx \\ & + \beta_{\text{KL}} \left(\epsilon - \int d_{\pi_{\text{sim}}}(x) \cdot \mathrm{D}_{\text{KL}}(\pi(\cdot \mid x) || \pi_{\text{sim}}(\cdot \mid x)) dx \right) \\ & + \int \alpha_x \left(1 - \sum_i \pi(a_i \mid x) \right) dx, \end{aligned} \tag{15}$$

where β_{KL} and $\alpha_{\text{prob}} = \{\alpha_x | \forall x\}$ are Lagrange multipliers for the KL and normalization constraints, respectively. Setting $\partial \mathcal{L} / \partial \pi(a_i \mid x) = 0$ yields the optimal policy:

$$\pi^*(a_i \mid x) = \frac{1}{Z(x)} \pi_{\text{sim}}(a_i \mid x) \exp\left(V'(x, a_i)\right), \quad V'(x, a_i) = \frac{1}{\beta_{\text{KL}}} V(x, a_i), \tag{16}$$

where $Z(x)$ is the normalization constant:

$$Z(x) = \exp\left(\frac{\alpha_x}{d_{\pi_{\text{sim}}}(x) \beta_{\text{KL}}} + 1 \right) = \sum_i \left(\pi_{\text{sim}}(a_i \mid x) \exp\left(\frac{1}{\beta_{\text{KL}}} V(x, a_i) \right) \right), \tag{17}$$

ensuring that $\pi^*(a_i \mid x)$ is a valid probability distribution. Notably, DPO [32] derives a similar form of the optimal policy via a different formulation. The optimal form $\pi^*(a_i \mid x)$ is invariant to subtracting a state-dependent baseline from $V(x, a_i)$ in the exponent, since any such baseline is absorbed into $Z(x)$; we therefore use the bare value $V(x, a_i)$ rather than an advantage.

A.3 Practical optimization objective

Training objective. Our goal is to train the selection heads to approximate the optimal policy in Eq. (16). However, direct optimization using the weighted maximum-likelihood approach of [29] is impractical because the sampling policy $\pi_{\text{sim}}(a_i \mid x)$ is intractable and rollout-based value estimates become noisy when amplified by small β_{KL}. We therefore approximate the optimal policy using binary correctness labels, where $r_i = 1$ if selecting action a_i leads to the correct final answer, and $r_i = 0$ otherwise.

In Eq. (16), the optimal policy $\pi^*(a_i \mid x)$ reweights the sampling policy $\pi_{\text{sim}}(a_i \mid x)$ according to the value $V(x, a_i)$. Approximating $\pi^*(a_i \mid x)$ therefore encourages the selection heads to assign higher probabilities to action a_i selected by $\pi_{\text{sim}}(a_i \mid x)$ that lead to high-value outcomes and lower probabilities to those that lead to poor outcomes. Our binary surrogate follows the same intuition by training the selection heads to distinguish successful trajectories sampled from $\pi_{\text{sim}}(a_i \mid x)$ from unsuccessful ones.

Per-unit selection head training. To support incrementally expanding memory, we train a separate selection head for each sub-instruction rather than a single joint classifier. This design allows newly added instructions to be trained independently without retraining existing classifiers. Specifically,

we decompose selection into one-vs-rest binary classification problems. For the i-th memory unit, positive samples are states where applying a_i leads to a correct final answer, while negative samples include states where applying a_i leads to an incorrect answer or where another instruction succeeds instead. We train each classifier using binary cross-entropy:

$$\ell(x, \boldsymbol{\theta}_i; r_i) = -\left[r_i \cdot \log\left(\pi_{\boldsymbol{\theta}_i}(a_i \mid x)\right) + (1 - r_i) \cdot \log\left(1 - \pi_{\boldsymbol{\theta}_i}(a_i \mid x)\right)\right], \tag{18}$$

where $r_i \in \{0, 1\}$ denotes the binary label for action a_i. This objective encourages high scores for instructions that lead to correct solutions while suppressing unsuitable ones.

Collecting diverse training data. In our data collection procedure, we select sub-instructions based on the similarity between step-goal embeddings and sub-goal, without relying on predictions from the selection heads. This design aims to mitigate selection bias and promote data diversity. In contrast, using the selection head during data collection would bias the sampled applications toward its current preferences, resulting in less informative and less diverse training data. By decoupling data collection from the selection head, we obtain a more diverse set of application trajectories, which enables the selection head to more effectively learn when to apply each sub-instruction.

Connection to and departure from prior derivations. The closed-form Eq. (16) we use is the same form as the optimum derived in [29] for advantage-weighted regression and, in a different domain, in [32] for direct preference optimization. We share with these works only the *derivation of the optimum*.

In AWR [29], the policy is iteratively improved by applying the current policy and updating it toward a new optimal policy. However, in our setting, iteratively refining the policy for a single question may introduce bias and degrade performance on other questions, while updating the policy across all existing questions is computationally expensive. In practice, we replace AWR's soft advantage-weighted maximum-likelihood surrogate with a binary cross-entropy surrogate over per-unit classifiers, and we do not perform iterative policy improvement: training data is collected once via MCTS rollouts on confident samples (Section 3.6), and the per-unit heads are trained on the collected data. We therefore avoid iterative policy updates.

ExIt [4] also employs iterative updates: in each iteration, it collects trajectories via MCTS and trains the policy to imitate these trajectories, biasing future exploration toward more promising decisions. In contrast, we estimate the optimal policy using the AWR formulation and train selection heads to directly imitate this policy, eliminating the need for repeated MCTS-based iterations.

B Broader context and related work

B.1 Step-wise guidance for LLM reasoning

Different reasoning strategies should be applied as the reasoning state evolves. ArcMemo [13] selects modular concepts by prompting the reasoning model, while CoM [15] prompts a meta-agent LLM to select a predefined cognitive mode for the next reasoning step as the reasoning process progresses.

Process reward models (PRMs) may provide reliable evaluations of reasoning trajectories [23]. PGTS [22] trains a graph neural network to select tree-search actions with the goal of maximizing cumulative rewards computed by a PRM or LLM-based evaluation. RLoT [11] formulates the reasoning process as a Markov Decision Process (MDP) and trains an RL agent to predict the next logic block based on the current reasoning state, using a PRM as a teacher to guide the training of the RL agent.

In contrast to these approaches, our method enables adaptive step-wise memory selection without requiring an additional PRM to guide training or inference. Moreover, our approach progressively accumulates knowledge as experience grows, rather than restricting the LLM to a fixed set of predefined logic blocks.

B.2 Self-improvement for LLM reasoning

Self-improvement methods aim to enhance LLM reasoning performance using the model's own generations. STaR [52] prompts the LLM to generate reasoning chains and then fine-tunes the model on trajectories that lead to correct answers. B-STaR [53] further improves performance by adaptively balancing exploration and exploitation. While effective, these approaches require updating model parameters, which is infeasible for closed-source models.

Self-Refine [26] and Reflexion [35] instead iteratively refine LLM responses using self-generated feedback. These methods improve performance by modifying the context rather than updating model parameters. However, their effectiveness largely depends on the quality of the generated feedback. BRIDGE [39] iteratively generates and selects influential demonstrations to enhance the reasoning capability of in-context learning.

Compared to existing approaches, our method constructs and learns to select reusable step-wise memory from confident samples, and then applies this memory to guide reasoning on uncertain samples. Unlike prior self-improvement methods, our approach requires neither model parameter updates nor additional supervision signals or self-feedback. As a result, it is applicable to both open-source and closed-source models in the test-time setting. Moreover, because our method relies on reusable memory derived from previous experience rather than iterative self-feedback, its effectiveness is less dependent on the quality of generated feedback.

B.3 Retrieval-augmented generation for reasoning

Retrieval-augmented generation (RAG) methods dynamically retrieve external information relevant to an input query and condition generation on the retrieved content. ReAct [51] combines internal and external knowledge by prompting the LLM to search and retrieve information from Wikipedia, aiming to improve reasoning through external knowledge access. HARR [56] formulates multi-hop retrieval as a Markov Decision Process and optimizes the retrieval policy to maximize the F1 score of LLM responses using GRPO [34]. However, existing retrieval-based methods typically operate on a fixed corpus and rely on large-scale training data. In contrast, our work focuses on dynamically expanding memory, where each step-wise instruction is associated with only limited training data.

Rather than retrieving from a fixed corpus, our method incrementally accumulates reusable knowledge by constructing and learning to select step-wise memory. Furthermore, our training framework employs dense step-wise supervision with a one-vs-rest objective, enabling effective learning from limited data while supporting incremental memory expansion during test time.

B.4 Test-time scaling for LLM reasoning

Test-time scaling methods improve model performance by allocating additional computation during inference without modifying model parameters. Chain-of-Thought (CoT) prompting [44] improves the reasoning ability of large language models (LLMs) by prepending manually crafted reasoning examples to the target question. [16] shows that longer reasoning chains consistently improve reasoning performance. Agarwal et al. [1] further demonstrate that increasing the number of few-shot demonstrations mitigates pretraining bias and enhances reasoning ability.

Generating and aggregating multiple responses during inference can further improve reasoning performance. Self-Consistency (SC) [42] aggregates multiple reasoning trajectories to produce more reliable answers.

Structured-search methods for reasoning explore multiple step-level continuations and select among them using value or process rewards. Tree-of-Thought (ToT) [50] and Graph-of-Thought (GoT) [5] introduce structured exploration strategies that improve reasoning performance. rStar [30] explores reasoning states using human-like actions within an MCTS framework and selects trajectories based on mutual consistency. Tree-BoN [31] reduces the computational cost of tree search by expanding only promising reasoning chains. DORA [41] adaptively allocates rollout budgets based on PRM scores and trajectory similarity, assigning more computation to reasoning states with lower similarity and higher PRM scores.

However, these methods search over memoryless LLM rollouts without reusable cross-problem knowledge guiding expansion. In contrast, MILES integrates learned, reusable step-level memory into the search, where selection heads choose which sub-instruction to apply at each expansion step.

C Detailed method pipeline

Algorithm. Algorithm 1 summarizes one pass through the test stream.

Algorithm 1 MILES: one pass through the test stream

Require: Frozen LLM with generator $p(\cdot \mid \cdot)$, embedding model, number of initial responses K

Initialize memory $\mathcal{M} \leftarrow \emptyset$, context codebook $\{\mathbf{q}_i\} \leftarrow \emptyset$, selection heads $\{\pi_{\boldsymbol{\theta}_i}\} \leftarrow \emptyset$
for each test sample q arriving in the stream **do**
  Sample K independent responses from p for q
  if the K responses are consistent (*confident sample*) **then** ▷ *confident sample*
    Set pseudo-label $\hat{y} \leftarrow$ majority-vote answer
    Run MCTS from $x_0 = q$ with Layer 1 sampling π_{sim} to expand reasoning states
    for each expansion (x, a_i) produced by MCTS **do**
      Roll out from (x, a_i) and set $r_i(x) \leftarrow \mathbb{1}[\text{rollout answer} = \hat{y}]$
      Append $(x, r_i(x))$ to per-unit dataset $\mathcal{D}_i$
    end for
    Summarize new (step-goal, reasoning step) tuples into new memory units
    instantiate corresponding selection heads and context codebooks
    Update each affected $\pi_{\boldsymbol{\theta}_i}$ by minimizing the BCE loss (6) on $\mathcal{D}_i$
    Output the pseudo-label $\hat{y}$ as the answer for q
  else ▷ *uncertain sample*
    Run memory-guided tree search using current $(\mathcal{M}, \{\mathbf{q}_i\}, \{\pi_{\boldsymbol{\theta}_i}\})$ as described above
    Output the majority-vote answer over all branches in the response pool
  end if
end for

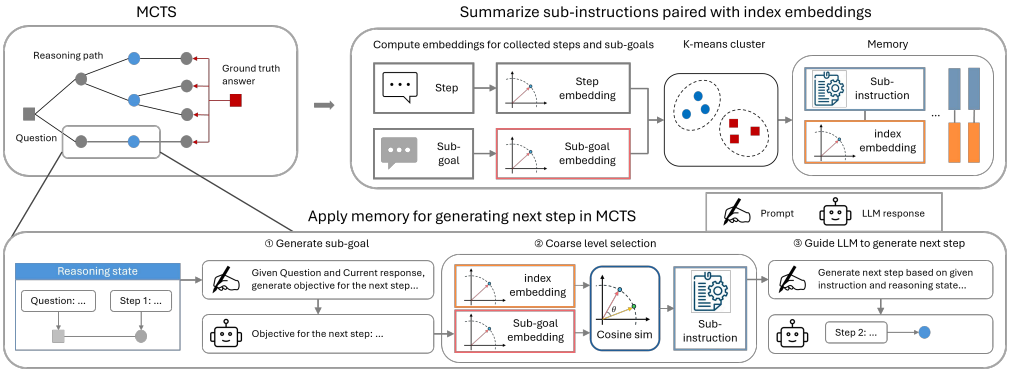


Figure 4: Overview of MCTS for memory construction and data collection. Sub-instructions are summarized from reasoning trajectories and applied during MCTS to collect training data for the selection heads.

C.1 Detailed MCTS pipeline

As discussed in [40, 54]. MCTS enables step-wise evaluation while balancing exploration and exploitation. Following [10, 45, 14], our MCTS procedure consists of four main phases: *selection*, *expansion*, *simulation*, and *backpropagation*. Each node represents a reasoning state corresponding to a partial response, with the root node representing the question q. Each edge denotes the transition between two reasoning states.

Selection. Starting from the root node, the MCTS algorithm recursively selects nodes for expansion until reaching a leaf node using the Upper Confidence Bound for Trees (UCT) criterion [18]:

$$\text{UCT}(v) = \frac{Q(v)}{N(v)} + C \cdot \sqrt{\frac{\ln N(\text{parent}(v))}{N(v)}}, \tag{19}$$

which balances exploration and exploitation when selecting nodes for expansion. Here, $Q(v)$ denotes the accumulated reward of node v, $N(v)$ and $N(\text{parent}(v))$ denote the visit counts of node v and its parent node, respectively, and C is a scalar hyperparameter controlling the exploration–exploitation trade-off. Larger values of C encourage selecting nodes with fewer visits, promoting exploration, while smaller values favor nodes with higher rewards, promoting exploitation. In our setting, the

Algorithm 2 MCTS for memory construction.

Require: Maximum number of rollouts n_{mcts}, number of children nodes c_{mcts}, memory $\mathcal{M}$, question q, estimated ground-truth final answer $\hat{y}$.

Set q as the root node.
for $i = 1$ to n_{mcts} **do**
▷ Selection
    Select a node for expansion using UCT in Eq. (19).
    Extract the reasoning state x from the selected node.
▷ Expansion
    Generate a step-goal based on x by prompting the LLM.
    Select c_{mcts} sub-instructions based on the similarity between the step-goal embedding and sub-goals.
    Generate the next reasoning step guided by the selected sub-instruction.
▷ Simulation
    Prompt the LLM to complete the reasoning process and obtain the final answer y.
▷ Backpropagation
    Compute the correctness reward r, where $r = 1$ if $y = \hat{y}$ and $r = 0$ otherwise.
    Backpropagate r through the MCTS trajectory and update the reward and visit count of each node.
▷ Training trajectory collection
    Collect (x, r) for each applied sub-instruction for training selection head.
end for

reward is defined as the binary correctness of the final answer, and rewards are backpropagated through all nodes along the reasoning path.

Expansion. Previous work typically expands a selected node by applying an action from a predefined action space. In contrast, our method generates the next reasoning step by applying a sub-instruction rather than relying on a fixed action space. These instructions are selected based on the similarity between step-goal embeddings and sub-goals. The expansion phase therefore collects application data for sub-instructions, which is subsequently used to train the selection heads.

Simulation. During the simulation phase, the LLM is prompted to complete the reasoning process and produce a final answer, which is then used to evaluate correctness. In our formulation, each node corresponds to a reasoning state. For a reasoning chain with n steps, there are up to n nodes along the reasoning path, where the i-th node contains the first i reasoning steps. To prevent excessive rollouts at large depths and maintain diversity in the contexts where sub-instructions are applied, we divide the simulated reasoning path into equal segments and merge multiple steps into a single node, keeping the total search depth fixed.

Backpropagation. After each simulation, the generated answer is evaluated and the leaf node is assigned a binary correctness signal. This signal is then propagated back through all nodes along the reasoning path leading to the final answer, refining the estimated quality of each node and guiding node selection in subsequent search iterations.

During the initial rollouts from the root node, we do not apply sub-instructions. This stage is therefore equivalent to self-consistency [42], and is used to estimate a pseudo ground-truth answer via majority voting. Subsequent backpropagation is then performed based on this estimated answer. Overall, MCTS enables efficient exploration of reasoning trajectories while collecting sub-instruction application data for training the selection heads. The sub-instruction trajectories explored during MCTS are collected as training data for the selection heads. The detailed MCTS procedure is described in Alg. 2. The details of each step in MCTS are illustrated in Figure 4.

C.2 Detailed memory-guided tree search pipeline

When the estimated answers are not consistent on given question, the estimated ground-truth answer and may not be correct and the correctness feedback may be noisy. As discussed in Section C.1, the reward backpropagation and selection in MCTS may bias the expansion towards incorrect answer. Therefore, we do not apply MCTS when model is uncertain.

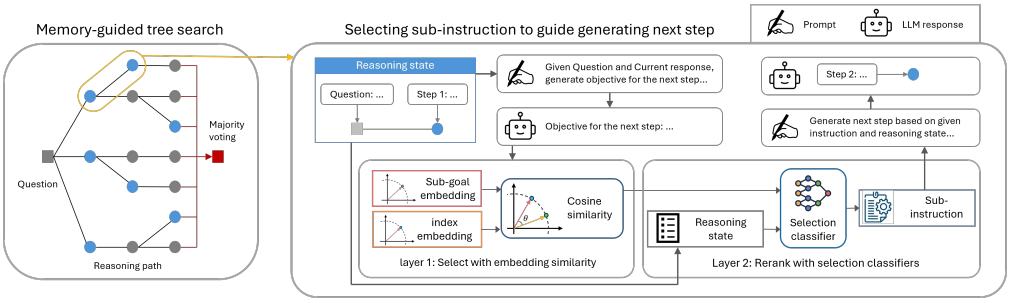


Figure 5: Overview of memory-guided tree search for inference. The search progressively expands reasoning steps on greater depths. At each step, we select multiple candidate step-wise instructions and rerank them using selection heads. The selected instruction is then used to generate the next reasoning step.

During memory-guided tree search, we apply a two-layer selection mechanism to choose sub-instructions for generating the next reasoning step given the current reasoning state. Starting from the question, we iteratively expand reasoning chains over multiple stages while maintaining a response pool that stores all generated trajectories.

At each stage, we randomly sample a sub-context from the response pool and combine it with the question to form the reasoning state. Based on this state, we select sub-instructions and guide the LLM to generate the next reasoning step. The LLM then completes the reasoning, and the resulting trajectory is added back to the response pool. As more sub-instructions are applied, we progressively increase the length of the sub-context, allowing the reasoning state to incorporate previously guided steps. Similar to MCTS, each node corresponds to a reasoning state and each edge represents a transition between states, with the question serving as the root node.

Our method predicts the applicability of sub-instructions based on the current reasoning state, but does not evaluate the overall quality of the reasoning state, unlike PRM-based methods [31, 10, 41]. As a result, we randomly select expansion positions during search. In contrast to TreeBoN [31], we do not prune branches, since the quality of a reasoning trajectory cannot be reliably determined. The detailed memory-guided tree search procedure is described in Alg. 3. Figure 5 illustrates the details of each step in memory-guided tree search.

D Memory construction details

D.1 Effective threshold

The effective threshold is designed to regulate the application of memory unit based on the similarity between step-goal embeddings and sub-goals. We assume that higher similarity between a step-goal embedding and the sub-goal indicates that the corresponding memory unit is more suitable to apply.

Each memory unit may have a different effective threshold. Intuitively, if a memory unit is summarized from a more dispersed cluster of (step-goal, reasoning step) pairs, it is more general and can be applied under a wider range of conditions. In contrast, instructions derived from tightly clustered pairs are more specialized and should be applied more selectively. Based on this intuition, we compute the effective threshold according to the distribution of the corresponding cluster.

Specifically, for each memory unit summarized from a cluster of (step-goal, reasoning step) pairs, we first compute the normalized distance between the sub-goal and other step-goal embeddings in the cluster as

$$d(\mathbf{o}_j, g_i) = 0.5 \cdot (1 - \cos(\mathbf{o}_j, g_i)), \tag{20}$$

where $\cos(\cdot, \cdot)$ denotes cosine similarity, $\mathbf{o}_j$ is the j-th step-goal embedding in the cluster, and g_i is the sub-goal. We then define the effective threshold as

$$R_{\text{eff}} = \alpha_{\text{eff}} \cdot \frac{1}{N} \sum_{j=1}^{N} d(\mathbf{o}_j, g_i), \tag{21}$$

where α_{eff} is a hyperparameter controlling the applicability of the memory unit.

Algorithm 3 Memory-Guided Tree Search

Require: Maximum search depth d_{mgts}, number of rollouts per stage c_{mgts}, memory $\mathcal{M}$, question q.

Initialize the response pool $\mathcal{S}_{\text{res}} \leftarrow \emptyset$.
for $i = 1$ to d_{mgts} **do**
▷ Selection
    Randomly sample c_{mgts} responses from $\mathcal{S}_{\text{res}}$.
    Split the selected responses into reasoning steps.
    for each selected response **do**
        Randomly sample a truncation position p_{step} within $[(i-1)/d_{\text{mgts}} \cdot n_{\text{step}},\ i/d_{\text{mgts}} \cdot n_{\text{step}}]$, where n_{step} denotes the number of reasoning steps.
        Use the first p_{step} reasoning steps as the reasoning state x for expansion.
    end for
▷ Expansion
    Generate multiple step-goals based on x by prompting the LLM.
    Select multiple candidate sub-instructions for each step-goal using Layer 1 selection.
    Rerank the retrieved sub-instructions using the corresponding selection heads in Layer 2 selection.
    Generate the next reasoning step guided by the selected sub-instruction.
▷ Simulation
    Prompt the LLM to complete the reasoning process and obtain the final answer y.
    Add the generated response to $\mathcal{S}_{\text{res}}$.
end for
▷ Answer Estimation
Compute the final answer by majority voting.

Given the effective threshold defined above, when selecting memory unit given a step-goal embedding, we first compute the distance between the step-goal embedding and the sub-goal using Eq. (20). If this distance exceeds the effective threshold, we mask the corresponding sub-instruction and exclude it from selection.

D.2 Incrementally summarize new sub-instructions

As discussed in Section C.1, our method collects data by retrieving sub-instructions during MCTS. Since each instruction is constrained by an effective threshold, there are cases in which no sub-instruction is selected for the current reasoning state and step-goal. In such cases, we prompt the LLM to generate the next reasoning step that achieves the given step-goal, and store the resulting (step-goal, reasoning step) pair for summarizing new sub-instructions. As sub-instructions are incrementally summarized, their sub-goals gradually cover a larger region of the step-goal embedding space. This expansion enables the memory to handle a broader range of conditions as experience accumulates. The memory construction process is illustrated in detail in Figure 4.

E Detailed training method

The detailed training process is described in Alg. 4.

Construction and application of context code book. The step-goal describes the objective the next reasoning step is intended to achieve, while the contextual condition captures the most relevant information for generating that step. Together, they are the key factors determining whether a sub-instruction is applicable to a given reasoning state. During data collection, since each instruction may be applied across multiple reasoning states, we collect the corresponding step-goal and contextual conditions and compute their embeddings. We then cluster these embeddings and use the centroid of each cluster as a context codebook. In this way, context codebooks serve as representative embeddings of similar step-goal and contextual conditions. Each selection head maintains an individual context codebook to identify the information most relevant to its corresponding sub-instruction.

The context codebooks are used to identify relevant information in the reasoning state. Specifically, we first split the reasoning state into overlapping chunks and compute embeddings for each chunk.

Algorithm 4 Training selection heads

Require: Training datasets $\mathcal{D}_{1:m} = \{(x, r)\}$ for all sub-instructions, where m is the number of sub-instructions to update, and optimization steps n_{opt}.

for $i = 1$ to m **do**

▷ Context Codebook Construction

  Cluster sub-goal embeddings and contextual-condition embeddings using k-means clustering.

  Concatenate the centers of all clusters to construct the context codebook.

▷ Training Data Preparation

  for each x in $\mathcal{D}_i$ **do**

    Split x into overlapping chunks.

    Compute the feature vector using the context codebook according to Eq. (22).

  end for

  Collect all feature vectors as training data.

▷ selection head Training

  Initialize the selection head for the i-th sub-instruction as an MLP.

  Train the selection head for n_{opt} optimization steps using the BCE loss in Eq. (18).

end for

We then compute the maximum cosine similarity between each context codebook and the chunk embeddings:

$$\mathbf{m}_i(x) \;=\; [s_{i1}, s_{i2}, \ldots, s_{in}], \qquad s_{ij} \;=\; \max_{k \in [1,T]} \cos(\mathbf{q}_{ij}, \mathbf{c}_k), \tag{22}$$

where $\mathbf{m}_i(x)$ denotes the feature vector, $\mathbf{q}_{ij}$ and $\mathbf{c}_k$ represent the j-th context prototype and the k-th chunk embedding, respectively, T is the number of chunks, and n is the number of context prototypes.

Intuitively, the maximum cosine similarity captures whether information relevant to a given context codebook is present in the reasoning state: a higher similarity indicates stronger evidence of its presence. Thus, Eq. (22) encodes the presence of different step-goal and condition patterns into a fixed-dimensional feature vector, which is then used by the selection head to determine whether a sub-instruction should be applied.

Discussion with Q-learning. In LLM reasoning tasks, supervision is typically limited to the correctness of the final solution, resulting in sparse and delayed rewards. This makes it challenging to reliably estimate the contribution of individual intermediate actions. Value-based approaches, such as Q-learning, require estimating the expected reward of each action under a given reasoning state, which in practice relies on sufficient rollout samples. When the number of observations per sub-instruction is limited, accurately estimating such values becomes difficult and may suffer from high variance. In our setting, each sub-instruction is applied under diverse reasoning states, and only a small number of samples are available for each instruction–state pair. As a result, value estimation for individual instructions can be unreliable. In extreme cases, if the observed trajectories associated with a given instruction all lead to correct final answers, a value-based model may assign uniformly high values, failing to distinguish the conditions under which the instruction is truly applicable. Instead of explicitly estimating action values, our approach directly approximates closed form optimal policy over instructions using binary correctness signals, avoiding per-action value estimation under limited samples.

Discussion with ColBERT. ColBERT [17] measures document relevance to a query by summing the maximum similarity between each context codebook and the document vectors, where both are encoded by a BERT encoder. We use the cluster centers as context codebooks and follow the same idea that identify key information from one sequence by maximum similarity. Different from ColBERT [17], we retain the full sequence of maximum similarities between the context codebooks and the document vectors as the input representation, this indicates a combination of different queries.

Method for preventing overfitting. Since only a limited amount of data is collected for each sub-instruction, the selection heads may overfit to the training data and fail to generalize to new cases. To mitigate this issue, we adopt the following strategies:

1. **Cross-instruction pretraining.** For a target sub-instruction, we first collect data from other instructions with similar sub-goals. A model is then trained on this combined dataset using

Table 7: Performance comparison on AIME 2024 and AIME 2025 datasets using GPT-4.1-mini model.

Method type	Method	AIME 2024	AIME 2025
Prompt-based methods	Zero-shot CoT [19]	48.67 (3.02)	44.67 (2.56)
	AoT [38]	62.00 (3.41)	49.33 (1.00)
Memory-based methods	BoT [49]	52.67 (5.60)	41.37 (3.62)
	DC (*w/* coding) [37]	62.33 (2.26)	59.33 (3.14)
	DC (*w/o* coding) [37]	44.00 (2.46)	31.00 (2.01)
Test-time scaling	CoT-SC [42]	59.67 (1.76)	53.67 (2.38)
	ToT [50]	54.00 (1.88)	51.33 (1.23)
	rStar [30]	58.00 (1.23)	50.67 (1.88)
	DORA [41]	49.33 (1.00)	29.67 (1.05)
	Our method	**66.36** (**1.38**)	**60.33** (**1.35**)

the binary cross-entropy loss, which encourages the model to learn more diverse patterns for instruction selection. Finally, we fine-tune the classification layer on the training data of the target sub-instruction while keeping the remaining parameters fixed.

2. **Input dropout.** During training, we apply dropout to the input dimensions of the selection heads to prevent certain dimensions from dominating the prediction. This regularization encourages the model to rely on more robust features and improves generalization to unseen conditions.
3. **Regularization techniques.** In practice, we train multiple selection heads and average their predicted probabilities to obtain a more robust estimate. We additionally employ early stopping and weight decay to further reduce the risk of overfitting to the training data.

Preventing catastrophic forgetting. During test time, samples arrive sequentially, and the training data for each selection head is incrementally accumulated. As a result, the selection heads must be updated to incorporate newly acquired knowledge. However, updating the model on new data can lead to a significant degradation in performance on previously seen data, a phenomenon known as catastrophic forgetting [27]. To mitigate this issue while preventing extra complexity, we adopt a simple rehearsal strategy: we retain all previously collected data and train the selection heads jointly on both new and past data.

F Additional experiments

To further evaluate the effectiveness of our method, we compare it against a broad range of baselines on the AIME 2024 and AIME 2025 datasets using the GPT-4.1-mini model. The baselines include: **Prompt-based methods** guide LLM reasoning by designing effective prompts.

- **Zero-shot CoT** [19] elicits reasoning by simply adding the prefix *"Let's think step by step."*.
- **Atom of Thoughts (AoT)** [38] models reasoning as a Markovian process, where each state corresponds to a self-contained subproblem.

Memory-based methods accumulate reusable knowledge and strategies from prior reasoning experiences and leverage memory to guide LLMs in solving new problems.

- **Buffer of Thoughts (BoT)** [49] distills reasoning templates from past experiences and retrieves the most relevant templates to guide reasoning on new problems.
- **Dynamic CheatSheet (DC)** [37] distills and refines reusable strategies from prior experiences, and applies the most relevant strategy to solve new problems.

Test-time scaling methods allocate additional computation during inference to further improve reasoning performance.

Table 8: Final-answer accuracy using single-run majority voting under comparable output-token budgets on AIME 2024 and AIME 2025. MILES achieves the highest accuracy on both datasets.

Method	AIME 2024		AIME 2025	
	Acc.	Out tokens	Acc.	Out tokens
DC w/ code	63.33	4.10M	**60.00**	4.61M
DC w/o code	60.00	4.40M	53.33	4.99M
BoT	60.00	4.72M	50.00	4.57M
AM	50.00	3.88M	46.67	3.45M
RoT	60.00	3.99M	56.67	4.59M
RB	60.00	4.74M	53.33	5.13M
Ours	**66.67**	3.99M	**60.00**	3.61M

- **Self-Consistency (SC)** [42] aggregates multiple reasoning trajectories to produce more reliable predictions.
- **Tree-of-Thoughts (ToT)** [50] introduces a structured reasoning framework that enables LLMs to flexibly explore and expand multiple reasoning states, improving their problem-solving capabilities.
- **rStar** [30] explores reasoning states using human-like actions within a Monte Carlo Tree Search (MCTS) framework, and selects trajectories based on mutual consistency.
- **DORA** [41] adaptively allocates rollout budgets based on PRM scores and trajectory similarity, assigning more computation to reasoning states that exhibit lower similarity and higher PRM scores.

We compare all methods in terms of final accuracy; results are reported in Table 7. Compared to other test-time scaling approaches, MILES achieves higher accuracy, demonstrating the effectiveness of its memory construction and selection strategy.

DORA adaptively allocates rollouts to promising reasoning states, but performs worse than other methods, suggesting that PRMs may be unreliable for API-based models. In contrast, MILES improves performance without relying on external PRMs, highlighting its robustness.

Among memory-based methods [49, 37], DC achieves strong performance when coding is enabled, but its performance degrades substantially without coding. MILES achieves superior performance by leveraging additional computation for training-data collection and memory-guided tree search.

Across all compared methods, MILES consistently outperforms the baselines on both datasets, demonstrating the effectiveness and robustness of our method.

F.1 Performance with comparable token cost

To further evaluate the effectiveness of MILES, we compare it with memory-based baselines under comparable output-token budgets using GPT-4.1-mini. The baselines include Buffer-of-Thoughts (BoT) [49], Dynamic CheatSheet (DC) [37], ArcMemo (AM) [13], and ReasoningBank (RB) [28]. DC w/ code and DC w/o code denote Dynamic CheatSheet with and without code execution, respectively. For AIME, we additionally compare with Retrieval-of-Thought (RoT) [3].

All final answers are obtained by majority voting over multiple reasoning rollouts. As the majority-vote results are stable across runs, we omit error bars for this experiment.

As shown in Table 8, MILES achieves the highest accuracy on both AIME 2024 and AIME 2025. Because all methods use comparable output-token budgets, the performance differences primarily reflect differences in memory design rather than compute. RoT uses external memory constructed from ReasonFlux [48], which may contribute to its stronger performance relative to the other baselines.

We further evaluate MILES and the baseline methods on three academic benchmarks: GPQA-Diamond, MMLU-Pro Physics, and MMLU-Pro Engineering. As shown in Table 9, MILES achieves the highest accuracy across all three benchmarks under comparable output-token budgets, further demonstrating the effectiveness and robustness of our method.

Table 9: Final-answer accuracy under comparable output-token budgets on GPQA-Diamond, MMLU-Pro Physics and Engineering subset. MILES achieves the highest accuracy on all the datasets.

Method	GPQA-Diamond		MMLU-Pro Physics		MMLU-Pro Engineering	
	Acc.	Out tokens	Acc.	Out tokens	Acc.	Out tokens
DC	65.66	5.69M	83.50	4.54M	68.00	8.72M
BoT	67.17	4.81M	84.00	3.48M	72.50	7.54M
AM	69.19	4.20M	82.00	3.99M	64.50	7.38M
RB	66.67	5.37M	82.00	4.56M	64.00	7.78M
Ours	**73.23**	4.66M	**85.00**	3.74M	**75.00**	7.46M

Table 10: Ablation study on different components using GPT-4.1-mini and Qwen3-30B-Instruct.

Dataset	Layer 1	Layer 2	GPT-4.1-mini	Qwen3-30B-Instruct
AIME 2024	✗	✗	59.33 (1.00)	82.67 (1.51)
	✓	✗	62.66 (1.01)	83.00 (1.35)
	✓	✓	**66.36** (**1.38**)	**85.33** (**1.23**)
AIME 2025	✗	✗	53.33 (1.59)	68.34 (1.26)
	✓	✗	55.33 (2.01)	68.67 (2.31)
	✓	✓	**60.33** (**1.35**)	**72.66** (**1.01**)

F.2 Component ablation with more models

In addition to the ablation study in Section 4.2, we further evaluate our method across different LLMs, including GPT-4.1-mini and Qwen3-30B-Instruct, on the AIME 2024 and AIME 2025 datasets. Results are reported in Table 10.

Our method consistently improves performance across different models and datasets, demonstrating its effectiveness and robustness. We also observe that the performance of Qwen3-30B-Instruct is already close to saturation.

F.3 Cross-task knowledge transfer

To further evaluate the knowledge transferability of our memory, we conduct experiments in which memory is constructed and applied on different tasks. For mathematical reasoning, we construct cross-task and in-task memories from MATH-500 and AIME 2020-2024 and apply them to the 13 most uncertain AIME 2025 samples. For academic reasoning, we construct cross-task and in-task memories from the non-Diamond split of GPQA and from MMLU-Pro Physics samples disjoint from the evaluation subset, respectively, and evaluate them on the 25 most uncertain samples from the held-out MMLU-Pro Physics test set.

As shown in Table 11, cross-task memory consistently improves over the no-memory setting in both cases, indicating that the stored units capture transferable reasoning strategies rather than only dataset-specific patterns. For the closely related math-to-math transfer, cross-task memory performs comparably to in-task memory. For the more distant graduate-science-to-physics transfer, the improvement is smaller but remains positive, suggesting that transfer gradually decreases as task distance increases rather than failing entirely.

Table 11: Performance on the most uncertain samples from AIME 2025 and MMLU-Pro Physics using memory constructed from different tasks.

Test set	No memory	Cross-task memory	In-task memory
AIME 2025	18.46 (2.84)	35.25 (3.94) (MATH-500)	**36.92** (**2.32**) (AIME 2020–2024)
MMLU-Pro Physics	32.00 (4.05)	36.80 (2.96) (GPQA)	**46.00** (**2.02**) (MMLU-Pro Physics)

Table 12: Effect of the minimum cluster size on AIME 2024 with GPT-4.1-mini. Accuracy is stable for moderate cluster sizes (10–15); cluster sizes that are too small (5) produce overly specific sub-instructions, while too large (20) produce overly generic ones.

Cluster size	Accuracy
5	64.67 (1.23)
10	**66.36 (1.38)**
15	64.00 (1.51)
20	60.33 (0.75)

Table 13: Effect of α_{sum} on AIME 2024 with GPT-4.1-mini. Accuracy is stable within $\alpha_{\text{sum}} \in [1.0, 2.0]$ and drops only when step-goal similarity is ignored ($\alpha_{\text{sum}} = 0$).

α_{sum}	Accuracy
0	65.00 (1.69)
1.0	66.36 (1.38)
2.0	**67.36 (1.00)**

Table 14: Effect of the effective threshold α_{eff} on AIME 2025 with GPT-4.1-mini. Accuracy stays above 60.00% across the moderate range $\alpha_{\text{eff}} \in [2.0, 3.0]$ and only degrades at the extreme values 1.0 and 4.0.

Effective threshold	Accuracy
1.0	58.34 (1.26)
2.0	60.33 (1.35)
3.0	**64.67 (1.23)**
4.0	58.34 (1.25)

Table 15: Ablation of the two contributors to the context codebook (contextual conditions and step-goals) on AIME 2024 with GPT-4.1-mini. Both contributors are needed for the best accuracy.

Contextual condition	Step-goal	Accuracy
✗	✓	62.33 (1.15)
✓	✗	64.67 (2.27)
✓	✓	**66.36 (1.38)**

F.4 Ablation of summarization parameters

To evaluate the effect of our summarization method, we study two hyperparameters: (1) α_{sum}, which balances the similarities between step-goal embeddings and step embeddings; and (2) the minimum cluster size, which controls the level of abstraction, as only clusters whose size exceeds this threshold are summarized into sub-instructions. Both are evaluated on AIME 2024 with GPT-4.1-mini in Tables 13 and 12.

Both hyperparameters show the same robustness pattern. Accuracy remains stable across a moderate range and degrades only at the extremes. For α_{sum} (Table 13), values in $[1.0, 2.0]$ yield stable performance (66.36% and 67.36%). When $\alpha_{\text{sum}} = 0$, which entirely removes step-goal similarity from clustering, performance degrades. For the minimum cluster size (Table 12), values in $[10, 15]$ also remain stable; cluster sizes that are too small (e.g., 5) produce overly specific sub-instructions and drop accuracy, while cluster sizes that are too large (e.g., 20) produce overly generic sub-instructions and also drop. In practice we use $\alpha_{\text{sum}} = 1.0$ and a minimum cluster size in $[10, 20]$ depending on the dataset (Appendix I).

F.5 Impact of effective threshold

The effective threshold α_{eff} controls how broadly a sub-instruction is allowed to be applied: a larger threshold allows more diverse application contexts (and more diverse training trajectories) but raises the risk of applying a sub-instruction to states it does not fit. We evaluate the effect of $\alpha_{\text{eff}} \in \{1.0, 2.0, 3.0, 4.0\}$ on AIME 2025 with GPT-4.1-mini.

Table 14 shows that MILES is largely insensitive to α_{eff} over a wide range: accuracy stays at or above 60.00% for $\alpha_{\text{eff}} \in \{2.0, 3.0\}$, peaking at 3.0, and only degrades to 58.34% at the extreme values 1.0 and 4.0, where overly restrictive or broad applicability results in no relevant or additional irrelevant sub-instructions, respectively. In our experiments we use $\alpha_{\text{eff}} = 2.0$ for math datasets (Appendix I) without further tuning.

F.6 Ablation on context codebook

The context codebook in Layer 2 is built by clustering embeddings of *contextual conditions* and *sub-goals* (Section 3.4); we ablate the contribution of these two components on AIME 2024 with

Table 17: Performance comparison with and without MCTS on AIME 2024 and AIME 2025 using GPT-4.1-mini.

Dataset	Tree search method	Accuracy	Correct-answer proportion	Instruction-correct rate
AIME 2024	MCTS (MILES)	**66.36** (1.38)	**40.27**	**55.84**
	Random Tree search	66.02 (1.01)	34.21	40.82
AIME 2025	MCTS (MILES)	**60.33** (1.35)	**36.06**	**45.66**
	Random Tree search	58.00 (2.30)	34.18	41.35

GPT-4.1-mini. Removing either component drops accuracy, while combining both yields the best result (Table 15), supporting the assumption that a sub-instruction's applicability is jointly determined by what the next step should achieve i.e., step-goal and what context the current state provides.

F.7 Ablation on number of candidate sub-instructions

In memory-guided tree search, Layer 1 first retrieves candidate sub-instructions based on sub-goal similarity, which are then reranked by Layer 2 selection. We study the effect of the number of Layer 1 candidates on AIME 2025 with GPT-4.1-mini; results are reported in Table 16. Performance drops when too few candidates are retrieved (8 candidates), but remains stable with larger candidate sets (16 and 24), indicating that Layer 2 selection effectively identifies useful sub-instructions from the retrieved candidates. In all experiments, we use 16 candidates without tuning this hyperparameter.

Table 16: Effect of the number of Layer 1 candidates on AIME 2025 with GPT-4.1-mini. Performance degrades when too few candidates are retrieved.

Candidate number	Accuracy
8	53.33 (1.95)
16	60.33 (1.35)
24	**61.00** (1.15)

F.8 Ablation on MCTS

To evaluate the effect of MCTS, We conduct an ablation study on AIME 2024 and AIME 2025 using GPT-4.1-mini, replacing MCTS with random tree search. Specifically, we replace UCT-based selection with random selection during data collection from confident samples, while holding memory construction and selection head training fixed. The accuracy, correct-answer proportion, and instruction-correct rate are reported in Table 17.

Because only the search procedure varies, this comparison isolates the contribution of MCTS. MILES remains robust when MCTS is replaced by random tree search, indicating that its performance does not depend on MCTS and supporting the memory and selector design as the core contribution. MCTS nevertheless produces higher-quality training data, with a higher correct-answer proportion and instruction-correct rate on both datasets. The improved data quality directly benefits selector training.

F.9 Effect of selected sub-instructions

To validate that the sub-instructions selected during memory-guided tree search actually help, we compare, under matched reasoning states, the accuracy of reasoning trajectories that include steps guided by a selected sub-instruction against trajectories generated without such guidance. Experiments are conducted on all six benchmarks with GPT-4.1-mini and GPT-4.1; Table 18 reports the resulting accuracy improvement in percentage points.

As shown in Table 18, applying selected sub-instructions consistently improves reasoning performance across all 12 dataset-model cells (range 0.63–10.80 points). This confirms that the sub-instructions retrieved and reranked by our two-layer mechanism are effective at steering the LLM toward correct final answers.

F.10 Pseudo-label quality

The correctness labels of confident trajectories are determined by comparing the rollout results with the pseudo-labels estimated through self-consistency. To analyze the reliability of these pseudo-labels,

Table 18: Accuracy improvement (percentage points) of reasoning paths with vs. without applying a selected sub-instruction in memory-guided tree search, under matched reasoning states. Selected sub-instructions consistently improve performance across all six benchmarks and both backbones.

Dataset	Accuracy improvement	
	GPT-4.1-mini	GPT-4.1
MATH-500	6.48	2.23
AIME 2024	10.80	3.77
AIME 2025	9.12	2.31
GPQA-Diamond	4.36	3.21
MMLU-Pro Physics	4.12	8.73
MMLU-Pro Engineering	0.63	3.87

Table 19: Pseudo-label accuracy, proportion of samples identified as confident, and self-consistency accuracy.

Dataset	Pseudo-label acc.	Confident sample proportion (n / total)	Full-set SC acc.
MATH-500	96.83	442 / 500	89.20
AIME 2024	94.44	18 / 30	59.67
AIME 2025	76.47	17 / 30	53.67
GPQA-Diamond	79.87	159 / 198	72.02
MMLU-Pro Physics	88.40	181 / 200	83.00
MMLU-Pro Engineering	83.03	165 / 200	72.33

we evaluate the pseudo-label accuracy and the proportion of confident samples in experiments using GPT-4.1-mini. We also report self-consistency accuracy as a reference.

As shown in Table 19, pseudo-label accuracy exceeds full-set self-consistency accuracy across all benchmarks, indicating that agreement concentrates on more reliable predictions and provides substantially cleaner supervision for memory construction.

To further evaluate the reliability of the pseudo-labels, we conduct an experiment on AIME 2025 using GPT-4.1-mini. We train the selection heads using only truly correct confident samples while holding all other settings fixed. The results are shown in Table 20.

Both settings achieve comparable performance and are statistically indistinguishable, indicating that confidently incorrect samples do not adversely affect the learned memory. Training on only truly correct samples slightly reduces variance, while removing memory from confidently incorrect samples slightly lowers accuracy, likely due to reduced coverage. Moreover, because memory is constructed at the step level, an incorrect final answer does not imply that all intermediate steps are incorrect. Our manual analysis (Appendix F.12) confirms that confidently incorrect cases retain non-trivial step-level quality.

F.11 Further robustness analysis

To further evaluate the robustness of MILES, we conduct experiments by (1) shuffling the sample order on AIME 2024 and AIME 2025 and (2) sampling different subsets from MMLU-Pro Physics and Engineering. We use GPT-4.1-mini as the backbone model. The results are reported in Tables 21 and 22. We note that the sample-order shuffling experiment differs from the sample-order analysis in Section 4.2.

Table 20: Performance comparison when selection heads are trained on all confident trajectories or only truly correct confident trajectories. Both settings achieve comparable performance.

Training data	Accuracy
All confident samples (MILES)	60.33 (1.35)
Truly-correct confident samples only	60.00 (1.12)

Table 21: Mean accuracy and 95% confidence intervals across different sample orders on AIME 2024 and AIME 2025.

Dataset	Accuracy
AIME 2024	66.67 (1.95)
AIME 2025	60.00 (1.59)

Table 22: Mean accuracy and 95% confidence intervals across different random subsets from MMLU-Pro Physics and Engineering.

Dataset	Ours	SC
MMLU-Pro Physics	84.63 (1.36)	83.00 (0.65)
MMLU-Pro Engineering	73.75 (1.89)	72.13 (1.77)

The results in both tables demonstrate the robustness of MILES. Across different subsets of MMLU-Pro Physics and Engineering, MILES consistently outperforms the self-consistency (SC) baseline, further supporting the effectiveness of our method.

F.12 Manual analysis

To evaluate the effectiveness of our modular memory, we conduct a manual analysis on the AIME datasets, with raters unaware of sample confidence and final-answer correctness. For each memory-application case, we evaluate the full pipeline—reasoning state $\rightarrow$ generated step-goal $\rightarrow$ selected sub-instruction $\rightarrow$ guided next step $\rightarrow$ resulting trajectory—along six dimensions scored on $\{0, 1, 2\}$: step-goal validity (whether the step-goal is a reasonable and useful immediate objective), sub-instruction applicability (whether the selected sub-instruction is relevant and applicable to the step-goal), instruction execution (whether the guided step substantively follows the instruction), local correctness, local coherence, and local contribution (whether the guided step makes meaningful progress).

To assess the reliability of our manual analysis, we first score 20 memory-application cases drawn from both confident and uncertain samples across AIME 2024 and AIME 2025, as shown in Table 23.

Cases from confident samples consistently receive higher scores than those from uncertain samples, indicating that these metrics meaningfully capture differences in reasoning quality.

To evaluate the effectiveness of the selection heads, we then score 20 memory-application cases from uncertain samples on AIME 2025, comparing MILES with the variant without selection heads (Exp. 2 in Table 4). The two methods differ only in their selection mechanism, and the results are shown in Table 24.

MILES improves the dimensions most directly related to effective reasoning composition—local contribution (1.30 vs. 0.80) and local coherence (1.50 vs. 1.25)—indicating that the selected guidance promotes meaningful and coherent progress rather than superficial contextual matching. We also find that step-goals associated with sub-instructions selected by the learned heads are of higher quality, providing further evidence that MILES selects guidance aligned with reasonable intermediate objectives rather than producing correct answers through unsound reasoning.

Table 23: Evaluation scores across six metrics on AIME 2024 and AIME 2025.

Metric	AIME24 Confident	AIME24 Uncertain	AIME25 Confident	AIME25 Uncertain
Step-goal validity	1.90	0.85	1.90	0.90
Sub-instruction applicability	1.50	1.20	1.10	0.95
Instruction execution	1.35	1.20	1.00	0.85
Local correctness	1.70	1.05	1.70	1.05
Local coherence	1.55	1.30	1.70	1.50
Local contribution	1.30	1.05	1.35	1.30

Table 24: Evaluation scores across six metrics with and without selection heads on AIME 2025.

Metric	Baseline (no heads)	MILES
Step-goal validity	0.60	**0.90**
Sub-instruction applicability	0.85	**0.95**
Instruction execution	0.85	0.85
Local correctness	1.00	**1.05**
Local coherence	1.25	**1.50**
Local contribution	0.80	**1.30**

Table 25: Evaluation scores across six metrics for confidently correct and confidently incorrect samples on AIME 2025.

Metric	Correct	Wrong
Step-goal validity	1.85	1.35
Sub-instruction applicability	1.30	1.05
Instruction execution	1.05	0.90
Local correctness	1.85	1.40
Local coherence	1.60	1.45
Local contribution	1.50	1.20

Since reasoning is a multi-step process, we further compare confidently correct and confidently incorrect samples on AIME 2025, as shown in Table 25.

Confidently correct cases score higher across all six dimensions, indicating that correct final answers are associated with consistently stronger intermediate reasoning. At the same time, confidently incorrect cases retain non-trivial local quality, supporting the step-level robustness of MILES: an incorrect final answer does not imply that all intermediate steps are invalid. We note that sub-instructions are intentionally formulated as high-level, generalizable guidance (e.g., “rewrite or simplify through substitution or factorization"), so their direct applicability may be moderate even when they provide useful guidance.

G Efficiency discussion

G.1 Reducing time delay via parallel data collection.

Our method relies on MCTS with multiple rollouts to collect teacher trajectories for training the selection heads, which can be time-consuming. To mitigate this overhead, we leverage parallel computation to reduce data collection latency.

Specifically, after the initial rollouts in MCTS, we estimate a pseudo ground-truth answer via majority voting and check for consistency across rollouts. If the answers are consistent, the LLM can directly return the final prediction, as additional data collection does not affect the outcome. In this case, trajectory collection can proceed asynchronously in the background, while the system handles new queries in parallel. If the answers are inconsistent, we instead invoke memory-guided tree search to continue reasoning and determine the final answer.

G.2 Applying KV-cache to reduce computation.

During both MCTS and memory-guided tree search, new branches are expanded from existing reasoning steps by prompting the LLM with the question, previous steps, and a selected sub-instruction to generate the next reasoning step. For open-source LLMs, the KV cache of previously processed tokens can be reused. As a result, the model avoids recomputing representations for earlier tokens, significantly reducing the computational cost of tree search.

Table 26: Detailed MCTS hyperparameters for experiments.

Dataset	Rollouts	Depth	Children Nodes	C
MATH-500	20	4	3	1.0
AIME 2024	40	5	3	1.0
AIME 2025	40	5	3	1.0
GPQA-Diamond	20	4	3	1.0
MMLU-Pro Physics	20	4	3	1.0
MMLU-Pro Engineering	20	4	3	1.0

H Broader impact

We propose a framework that learns to select sub-instructions at test time to improve reasoning performance. Our method does not require curated training datasets or process reward models (PRMs), enabling effective learning under limited data. Moreover, the memory construction process incrementally incorporates new sub-instructions as experience accumulates, allowing the system to adapt to diverse scenarios by updating memory rather than modifying LLM parameters.

As our approach relies on textual memory that is compatible across different LLMs, it enables knowledge transfer across models. In particular, trajectories collected from stronger models can be used to train selection policies for weaker models, potentially improving accessibility to high-quality reasoning capabilities without requiring large-scale training resources. Importantly, the teacher model does not need to be uniformly stronger across all domains. In specific areas, a smaller but more specialized model can provide high-quality trajectories to train selection policies for other models, enabling flexible and domain-specific knowledge transfer.

However, since the selection policy is learned from collected trajectories, it may inherit biases or errors present in the data, potentially leading to misleading or suboptimal outputs. Additionally, the accumulation of memory over time may amplify such biases if not properly controlled. To mitigate these risks, the use of textual memory allows for transparent inspection, as well as LLM-based verification and feedback. Future work may explore mechanisms for monitoring, filtering, and refining memory and selection policies through self-checking or external supervision.

I Implementation details

I.1 MCTS details

The detailed hyperparameters of MCTS are reported in Table 26. For more complex datasets, such as AIME 2024 and AIME 2025, we use greater search depth and more rollouts to collect training trajectories, while keeping the same hyperparameters for other datasets. The exploration constant C in Eq. (19) is set to 1.0 across all experiments.

I.2 Memory construction details

When clustering (step, step-goal) embeddings, instead of fixing the number of clusters in k-means, we impose a minimum cluster size to ensure that sub-instructions are summarized from sufficiently diverse reasoning steps, resulting in more reusable abstractions. Specifically, we set the minimum cluster size in the range of $[10, 20]$, depending on the similarity among queries within each dataset.

For math tasks, we set the α_{eff} to 2.0, as sub-instructions tend to generalize well across queries. In contrast, for academic tasks, we use a lower threshold in the range of $[1.0, 1.5]$, since sub-instructions are less transferable across different queries.

I.3 Training details

We use a 3-layer MLP with 16 hidden units as the selection head. For each sub-instruction, we train 10 independent classifiers and average their predicted application probabilities to improve robustness. We periodically update the classifiers during test-time processing: after every 2 questions,

Table 27: Detailed memory-guided tree search hyperparameters for experiments.

Dataset	Stage	Rollout per stage	Candidate instructions	β_{c2f}
MATH-500	5	4	16	1.0
AIME 2024	6	4	16	1.0
AIME 2025	6	4	16	1.0
GPQA-Diamond	5	4	16	0.4-0.6
MMLU-Pro Physics	5	4	16	0.4-0.6
MMLU-Pro Engineering	5	4	16	0.4-0.6

and whenever the number of training trajectories increases by 20. For smaller datasets such as AIME 2024 and AIME 2025, we instead update the selection heads after each question.

The context prototypes correspond to the cluster centers obtained from embeddings of contextual conditions and step-goals. We set the number of context prototypes (i.e., the number of clusters in k-means clustering) to $\min(N_i^{\text{app}} \cdot 0.2, 512)$, where N_i^{app} denotes the number of application cases for the i-th sub-instruction. The number of context prototypes increases gradually as the application cases become more diverse. We cap the maximum number at 512 to avoid overly complex context codebooks while maintaining clustering efficiency.

I.4 Memory guided tree search details

In our method, the application of memory-guided tree search is determined by the majority-answer proportion, i.e., answer consistency. Since different models exhibit different levels of answer consistency, we set the consistency threshold slightly lower than the zero-shot CoT accuracy measured on a small validation subset. We set the number of routing rollouts to 10 for all experiments. The detailed hyperparameters of memory-guided tree search are reported in Table 27. For more complex datasets, such as AIME 2024 and AIME 2025, we use greater search depths and more rollouts to enable broader application of sub-instructions during reasoning.

The parameter β_{c2f} in Eq. (3) controls the balance between selection and semantic relevance. We set $\beta_{\text{c2f}} = 1.0$ for math tasks, so that selection is determined solely by the selection heads, and use values in the range $[0.4, 0.6]$ for academic tasks to place greater weight on semantic relevance.

I.5 Decoding details

For all experiments using GPT-4.1-mini and GPT-4.1, we use the OpenAI API service. The decoding temperature is set to 0.0 for all experiments. The maximum response length is determined according to task complexity, ranging from 2048 to 16384 tokens. For experiments with GPT-4.1-mini, we use the `GPT-4.1-mini-2025-04-14` checkpoint.

For experiments using open-source models, we employ the vLLM framework [20] for local decoding and the OpenRouter API for online decoding. For all Qwen3 models, we follow the decoding settings described in the original paper [47]. For the GPT-OSS-20B model, we set the decoding temperature to 1.0 for all experiments.

All experiments use `text-embedding-3-small` from the OpenAI API service for embedding computation. The embedding dimension is fixed at 1536 across all experiments.

I.6 Hyperparameter tuning

We divide the hyperparameters into two groups: memory construction and memory application. We then examine three stages of the reasoning process: (1) step-goal generation, (2) two-layer selection, and (3) rollout and correctness evaluation. Based on these observations, we identify which components contribute to errors and adjust the corresponding hyperparameters. Specifically, if a suitable memory unit exists but the two-layer selection mechanism selects an inappropriate one, we tune the selection hyperparameters. If no suitable memory unit is available, we consider improving the diversity and generalizability of memory units or adjusting the selection criteria to allow fallback to the zero-shot baseline.

J Computation resources and assets

For experiments using API-based models, we train the selection heads on a single RTX 3090 GPU and access the models via LiteLLM `https://docs.litellm.ai/`. For open-source models, we either (1) deploy the models locally on a server with 8 A100 GPUs, or (2) access inference services through OpenRouter `https://openrouter.ai/`.

Open-sourced model URLs:

- Qwen3-30B-Instruct: `https://huggingface.co/Qwen/Qwen3-30B-A3B-Instruct-2507`.
- Qwen3-4B-Instruct: `https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507`.
- GPT-OSS-20B: `https://huggingface.co/openai/gpt-oss-20b`.

Dataset URLs:

- AIME datasets: `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`.
- MATH-500: `https://huggingface.co/datasets/HuggingFaceH4/MATH-500`.
- GPQA-Diamond: `https://huggingface.co/datasets/fingertap/GPQA-Diamond`.
- MMLU-Pro: `https://huggingface.co/datasets/TIGER-Lab/MMLU-Pro`.

K Case study

To illustrate the effectiveness of sub-instructions, we present two case studies from the algebra and geometry categories of MATH-500 during memory-guided tree search. For each case, we first show the response generated by Zero-shot CoT, followed by the response obtained by applying a selected sub-instruction at an intermediate reasoning step, together with the corresponding sub-instruction. Selected sub-instructions are highlighted in red, while shared responses between the two reasoning trajectories are highlighted in purple.

K.1 Algebra case

Case 1: Algebra

Question: Let x, y, and z be positive real numbers such that $xyz = 2$. Find the minimum value of $x^4 + 4y^2 + 4z^4$.

Zero-shot CoT:

Step 1: We can apply the AM-GM inequality to the terms x^4, $2y^2$, $2y^2$, $2z^4$, and $2z^4$ to relate their sum to their product, taking into account the constraint $xyz = 2$.

Step 2: Let's rewrite the expression as follows:

$$x^4 + 4y^2 + 4z^4 = x^4 + 2y^2 + 2y^2 + 2z^4 + 2z^4$$

Now, we have five terms: x^4, $2y^2$, $2y^2$, $2z^4$, $2z^4$.

Step 2: By the AM-GM inequality, we have

$$x^4 + 2y^2 + 2y^2 + 2z^4 + 2z^4 \geq 5\left(x^4 \cdot (2y^2)^2 \cdot (2z^4)^2\right)^{1/5}.$$

Step 3: Let's simplify the product inside the root:

$$x^4 \cdot (2y^2)^2 \cdot (2z^4)^2 = x^4 \cdot 4y^4 \cdot 4z^8 = 16x^4y^4z^8.$$

Step 4: Recall the constraint $xyz = 2$. We can write $z = \frac{2}{xy}$, so

$$z^8 = \left(\frac{2}{xy}\right)^8 = \frac{256}{x^8y^8}.$$

Thus,

$$16x^4y^4z^8 = 16x^4y^4 \cdot \frac{256}{x^8y^8} = 16 \cdot 256 \cdot x^4y^4/(x^8y^8) = 4096 \cdot \frac{1}{x^4y^4}.$$

Step 5: Substitute this back into the AM-GM inequality:

$$x^4 + 4y^2 + 4z^4 \geq 5\left(4096 \cdot \frac{1}{x^4y^4}\right)^{1/5} = 5 \cdot 4096^{1/5} \cdot (x^4y^4)^{-1/5}.$$

Step 6: Compute $4096^{1/5}$. Since $4096 = 2^{12}$, $4096^{1/5} = 2^{12/5} = 2^{2.4}$.

Step 7: Now, let's try to achieve equality in AM-GM, which occurs when all terms are equal:

$$x^4 = 2y^2 = 2y^2 = 2z^4 = 2z^4.$$

So $x^4 = 2y^2 = 2z^4$.

Step 8: Set $x^4 = 2y^2$ and $x^4 = 2z^4$. Then $y^2 = \frac{x^4}{2}$ and $z^4 = \frac{x^4}{2}$, so $z^2 = \left(\frac{x^4}{2}\right)^{1/2} = \frac{x^2}{\sqrt{2}}$.
Step 9: Now, use the constraint $xyz = 2$:

$$x \cdot y \cdot z = 2.$$

Recall $y = \frac{x^2}{\sqrt{2}}$ and $z = \frac{x^2}{\sqrt{2}}$, so

$$x \cdot \frac{x^2}{\sqrt{2}} \cdot \frac{x^2}{\sqrt{2}} = 2 \implies x \cdot \frac{x^4}{2} = 2 \implies \frac{x^5}{2} = 2 \implies x^5 = 4 \implies x = 4^{1/5} = 2^{2/5}.$$

Step 10: Now, $y = \frac{x^2}{\sqrt{2}} = \frac{(2^{2/5})^2}{\sqrt{2}} = \frac{2^{4/5}}{2^{1/2}} = 2^{4/5-1/2} = 2^{(8-5)/10} = 2^{3/10}$.
Similarly, $z = 2^{3/10}$.

Step 11: Now, compute the minimum value:

$$x^4 + 4y^2 + 4z^4 = (2^{2/5})^4 + 4(2^{3/10})^2 + 4(2^{3/10})^4.$$

Calculate each term: - $x^4 = (2^{2/5})^4 = 2^{8/5}$ - $y^2 = (2^{3/10})^2 = 2^{6/10} = 2^{3/5}$ - $z^4 = (2^{3/10})^4 = 2^{12/10} = 2^{6/5}$
So,

$$x^4 + 4y^2 + 4z^4 = 2^{8/5} + 4 \cdot 2^{3/5} + 4 \cdot 2^{6/5}$$

Step 12: This is the minimum value. Thus, the answer is $\boxed{2^{8/5} + 4 \cdot 2^{3/5} + 4 \cdot 2^{6/5}}$

Apply sub-instruction

<Sub-instruction>
Rewrite or simplify algebraic expressions or equations by substituting variables, factoring, or using properties of roots and exponents to transform them into more manageable forms.
</Sub-instruction>

Step 2: Substitute $z = \frac{2}{xy}$ into the expression to rewrite it as $x^4 + 4y^2 + 4\left(\frac{2}{xy}\right)^4$.

Step 3: Simplify the expression:

$$x^4 + 4y^2 + 4\left(\frac{2}{xy}\right)^4 = x^4 + 4y^2 + 4 \cdot \frac{16}{x^4y^4} = x^4 + 4y^2 + \frac{64}{x^4y^4}$$

Step 4: Let $a = x^4$ and $b = y^2$, so $x = a^{1/4}$, $y = b^{1/2}$, and $z = \frac{2}{xy} = \frac{2}{a^{1/4}b^{1/2}}$.

Step 5: Substitute these into the constraint $xyz = 2$:

$$xyz = a^{1/4}b^{1/2} \cdot \frac{2}{a^{1/4}b^{1/2}} = 2$$

So the constraint is satisfied for all positive a, b.

Step 6: The expression to minimize is:

$$a + 4b + \frac{64}{ab^2}$$

Step 7: Let's use symmetry and try $a = 2b$ (since the terms a and b appear with different powers, but the last term is symmetric in a and b^2). Set $a = kb$, so:

$$a + 4b + \frac{64}{ab^2} = kb + 4b + \frac{64}{kb^3}$$

Let's try $k = 2$:

$$2b + 4b + \frac{64}{2b^3} = 6b + \frac{32}{b^3}$$

Now, minimize $6b + \frac{32}{b^3}$ for $b > 0$.

Step 8: Take the derivative with respect to b and set to zero:

$$\frac{d}{db}\left(6b + \frac{32}{b^3}\right) = 6 - \frac{96}{b^4} = 0 \implies 6 = \frac{96}{b^4} \implies b^4 = 16 \implies b = 2$$

Step 9: Substitute $b = 2$ back:

$$6b + \frac{32}{b^3} = 6 \times 2 + \frac{32}{8} = 12 + 4 = 16$$

Step 10: Recall $a = 2b = 4$, so $x^4 = 4 \implies x = \sqrt{2}$, $y^2 = 2 \implies y = \sqrt{2}$, and $z = \frac{2}{xy} = \frac{2}{2} = 1$.

Step 11: Check the constraint: $xyz = \sqrt{2} \cdot \sqrt{2} \cdot 1 = 2$, which is satisfied.

Step 12: Compute the original expression:

$$x^4 + 4y^2 + 4z^4 = 4 + 4 \times 2 + 4 \times 1 = 4 + 8 + 4 = 16$$

Final answer: $\boxed{16}$

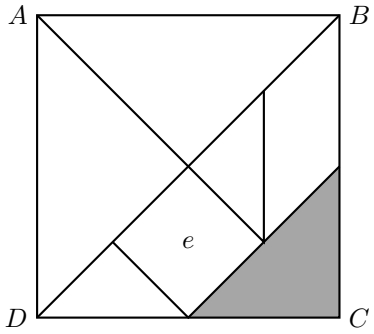
In the Zero-shot CoT response, the LLM attempts to solve the problem using the AM-GM inequality. However, this formulation leads to complicated computations and ultimately produces an incorrect answer. In contrast, the selected sub-instruction guides the LLM to first transform the expression into

a more manageable form. As a result, the LLM simplifies the formula and successfully reaches the correct answer.

K.2 Geometry case

Case 2: Geometry

Question: Quadrilateral $ABCD$ is a square with area 16 square inches. The figure represents the pieces of a Chinese tangram in which all the triangles are isosceles and piece "e" is a square. What is the area of the gray piece, in square inches?



Zero-shot CoT:

Step 1: Since the area of square $ABCD$ is 16 square inches, its side length is $\sqrt{16} = 4$ inches.

Step 2: The tangram divides the square into 7 pieces: 5 triangles, 1 parallelogram, and 1 small square (piece "e"). All triangles are isosceles right triangles, and piece "e" is a square.

Step 3: The gray piece in the diagram is the small triangle at the bottom right, adjacent to C and below piece "e". We need to find its area.

Step 4: Let's analyze the construction. The classic tangram divides the square into the following pieces (by area, as fractions of the whole square):
- 2 large triangles (each $\frac{1}{4}$ of the area)
- 1 medium triangle ($\frac{1}{8}$)
- 2 small triangles (each $\frac{1}{16}$)
- 1 square ($\frac{1}{8}$)
- 1 parallelogram ($\frac{1}{8}$)

Step 5: The gray triangle is one of the two smallest triangles, each with area $\frac{1}{16}$ of the total area.

Step 6: Since the total area is 16 square inches, the area of the gray piece is $16 \times \frac{1}{16} = 1$ square inch.

Final answer: $\boxed{1}$

Apply sub-instruction

<Sub-instruction>
Calculate the area of a triangle with sides a, b, and c by first finding the semi-perimeter $s = \frac{a+b+c}{2}$, then applying Heron's formula: $A = \sqrt{s(s-a)(s-b)(s-c)}$.
</Sub-instruction>

Step 3: To find the area of the gray triangle with vertices at $(2,0)$, $(4,0)$, and $(4,2)$, first compute the side lengths: $a = 2$ (from $(2,0)$ to $(4,0)$), $b = 2$ (from $(4,0)$ to $(4,2)$), and $c = \sqrt{(4-2)^2 + (2-0)^2} = \sqrt{4+4} = 2\sqrt{2}$ (from $(2,0)$ to $(4,2)$).

Step 4: The triangle is a right triangle with legs of length 2 (from $(2,0)$ to $(4,0)$ and from $(4,0)$ to $(4,2)$), so its area is $\frac{1}{2} \times 2 \times 2 = 2$ square inches.

Step 5: Therefore, the area of the gray piece is $\boxed{2}$ square inches.

In the Zero-shot CoT response, the LLM directly estimates the area of the gray region based on the partition of the square. However, it identifies the wrong triangle, which leads to an incorrect final answer. In contrast, the selected sub-instruction guides the LLM to compute the area of the gray region using Heron's formula. Consequently, the LLM first computes the lengths of the edges of the gray region in the following step. Although the LLM does not explicitly apply Heron's formula afterward, the computed edge lengths help it derive the correct area.

From the above cases, we observe that the selected sub-instructions guide the LLM toward more promising reasoning directions, ultimately leading to correct final answers. These results suggest that our method effectively achieves the objective of selecting appropriate sub-instructions to maximize the probability of generating correct solutions, demonstrating the effectiveness of our approach.

L All prompts

To ensure generalizability across tasks, we use a unified set of prompts for all experiments. To maintain compatibility with smaller models such as Qwen3-30B-Instruct and GPT-OSS-20B, we keep the prompts simple and straightforward.

Generate step-goal. This prompt generates a step-goal for the next reasoning step based on the current reasoning state. The resulting step-goal is used to select a sub-instruction by computing cosine similarity between its embedding and the stored sub-goals. The selected sub-instruction then guides the generation of the next reasoning step.

Generate step-goal

Instruction: Given the Question and Reasoning so far, output the immediate next-step objective that directly contributes toward achieving the final goal based on current progress. Respond in one sentence only.

Question:
<question>

Reasoning so far:
<reasoning so far>

Generate next step with sub-instruction. This prompt guides the LLM to generate the next reasoning step conditioned on the selected sub-instruction and the current reasoning state. In both MCTS and memory-guided tree search, generating the next reasoning step corresponds to expanding a new branch from the existing reasoning state.

Generate next step with sub-instruction

Instruction: Apply the given instruction to generate the NEXT reasoning step only.
- Output a single step, one sentence.
- Be precise. Use exact numbers, variables, or terms from the question and prior steps.
- Do not restate irrelevant details.
- Keep it in the same style as the previous steps.

- If the reasoning is complete, present the final answer as (\boxed{X}), where X is the correct answer.

Question:
<question>

Reasoning so far:
<reasoning so far>

Relevant instruction:
<relevant instruction>

Generate next step without sub-instruction. When no sub-instruction is selected, we prompt the LLM to generate the next reasoning step to achieve the given step-goal. The resulting (step-goal, reasoning step) pair is then stored for summarizing new sub-instructions.

Generate next step without sub-instruction

Instruction: Generate the Next reasoning step that best fulfills the given Objective of the next step in the context of the Question and Reasoning so far.
- Output a single step, one sentence.
- Be precise. Use exact numbers, variables, or terms from the question and prior steps.
- Do not restate irrelevant details.
- Keep it in the same style as the previous steps.
- If the reasoning is complete, present the final answer choice as (\boxed{X}), where X is the correct answer.

Question:
<question>

Reasoning so far:
<reasoning so far>

Objective of the next step:
<objective of the next step>

Complete reasoning. This prompt directly asks the LLM to complete the reasoning from the current reasoning state, corresponding to the simulation phase in MCTS. In memory-guided tree search, we perform completion after each expansion. Specifically, after applying a sub-instruction to generate a new reasoning step, we prompt the LLM to complete the remaining reasoning and produce the final solution.

Complete reasoning

Instruction: You are given a question and a partially completed reasoning chain. Continue the reasoning step by step, each step starts with 'Step i:', where i is the step number, until the solution is complete. Do not restart from the beginning.
Present the final answer as (\boxed{X}), where X is the correct answer.

Question:
<question>

Reasoning so far:
<reasoning so far>

Summarize sub-instructions. This prompt asks the LLM to extract common reasoning patterns from multiple similar reasoning steps and summarize them into a sub-instruction. The resulting sub-instruction is then used to guide the generation of new reasoning steps when selected.

Summarize sub-instructions

Instruction: You are given several reasoning steps that are semantically similar.
Your task is to summarize them into a general instruction that captures the shared logic or operation behind these steps.

Guidelines:
- Abstract away task-specific numbers or entities (replace them with variables or placeholders).
- Keep the instruction short and action-oriented (like a recipe step).
- If there are small differences across the steps, generalize to a pattern that covers them.
- Output only one generalized instruction.

Reasoning Steps:
<Reasoning Steps>

Format:
<general_instruction>
[one-sentence generalized instruction here, using variables or placeholders if needed]
</general_instruction>

Identify contextual conditions. This prompt extracts the contextual conditions for a reasoning step from the question and preceding responses. The extracted conditions are used to build a codebook that identifies the most relevant context for applying the corresponding sub-instruction.

Identify contextual conditions

Instruction: Try to identify the INPUT CONDITIONS for the NEXT reasoning step only (not the whole solution) from Question and the Reasoning so far.
- Output the identified INPUT CONDITIONS only.

Question:
<question>

Reasoning so far:
<reasoning so far>

Next reasoning step:
<next reasoning step>

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our claims in the abstract and introduction accurately reflect the contribution and scope of this work: we propose MILES to improve test-time reasoning through memory construction and compositional memory learning.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The conclusion section (Section 5) includes a discussion of the limitations of our work as well as directions for future research.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: All derivations, along with their corresponding assumptions and insights, are provided in Appendix A

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide method details, algorithms, and hyperparameters essential for reproducing our results in the Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: All datasets used in this work are publicly available. Our code will be released at `https://artificer-ai-lab.github.io/MILES` before the conference.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: All experimental settings and hyperparameter configurations are provided in the main text and Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report 95% confidence intervals computed over independent runs, using the same number of runs for all methods within each benchmark. However, given the relative stability of majority voting, we omit confidence intervals for the experiments with comparable token costs (Appendix F.1).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The Appendix includes details regarding the computational resources used in our experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our work aims to improve test-time reasoning on LLM reasoning tasks through modular step-wise memory. It is not intended to generate harmful content or recover private user information.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We include a discussion of broader impact in Appendix H.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our method uses publicly available LLMs and datasets. Therefore, this concern is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All datasets and code used in our work are publicly available. We have included the corresponding URLs in Appendix J and comply with the terms of use specified in their respective licenses.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [No]

    Justification: Although the assets are not released at the time of submission, we plan to release the code at `https://artificer-ai-lab.github.io/MILES`. The release will include comprehensive documentation and scripts to facilitate reproducibility.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: Our work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Our work does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: Our work evaluates and improves reasoning in pretrained LLMs used for inference in the experimental pipeline. We describe the LLMs, inference settings, and prompting configurations used in our experiments in the main text and Appendix.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.