HERMES: LEARNING CONTEXTUAL REASONING UNLOCKS TEST-TIME SCALING

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling improves model performance by allocating additional compute during inference. Using this compute effectively across multiple context windows requires deciding how to allocate fresh contexts and what information to carry between them. We call a model's ability to make these decisions *contextual reasoning*. Existing approaches largely prescribe these decisions through their harness; we instead shift them to the model. We introduce 1) HERMES, a family of simple, configurable harnesses that progressively varies model control over context allocation and reuse, and 2) HERMES-LEARN, a two-stage framework for learning these capabilities. We find that capable models can exploit this flexibility to scale with additional inference-time compute, while smaller open-source models initially struggle to do so. Training with HERMES-LEARN closes this gap, inducing adaptive contextual reasoning strategies that vary with both the problem and the progress of reasoning. These gains generalize across benchmarks and models, extrapolate beyond the inference-time compute seen during training, and transfer to complementary test-time scaling methods beyond HERMES.

1 INTRODUCTION

> *"The capacity of the human mind for formulating and solving complex problems is very small compared with the size of the problems . . . "*
>
> *— Herbert A. Simon, Models of Man (Simon, 1957)*

Large language models have made impressive progress on complex reasoning, increasingly contributing to mathematical research (Alpöge & Furman, 2026; OpenAI, 2026). This progress is fueled by *test-time scaling*: allocating additional computation during inference to improve model performance (Snell et al., 2025). Recent advances have involved thousands of agents reasoning in separate contexts, with harnesses coordinating how those contexts are allocated and how intermediate results are carried forward. This highlights two complementary dimensions of reasoning. Mathematical reasoning determines how to make progress given the information available within a context, while *contextual reasoning* determines how separate contexts should be allocated, coordinated, and reused as reasoning unfolds. In this work, we focus on contextual reasoning.

Concretely, contextual reasoning requires two types of decisions: i) How should additional contexts be used in order to explore an alternative solution, verify prior work, or continue an existing line of reasoning? ii) What information should be carried forward into those contexts? Existing approaches typically encode these decisions in their harnesses. For example, Reasoning Cache repeatedly compresses prior reasoning and continues from the resulting summary in a fresh context (Wu et al., 2026a), while Recursive Agent Optimization uses recursive delegation to create fresh contexts for subproblems and learns when and how to invoke them (Gandhi et al., 2026). Yet the best use of additional contexts may change across stages of problem solving. For instance, a difficult proof may initially benefit from exploring several approaches in parallel, then verifying a promising intermediate result, and finally continuing a successful line of reasoning. This motivates a flexible harness for contextual reasoning that exposes multiple ways to use additional contexts and allows the model to choose among them based on the current state of the solution.

To this end, we introduce HERMES, a family of simple, flexible, and configurable harnesses that progressively shifts contextual reasoning control from the harness to the model. HERMES exposes

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

two generic mechanisms: *delegation*, for allocating fresh context, and *digestion*, for carrying useful information forward. By varying how strongly the harness steers these decisions, HERMES defines a hierarchy ranging from prescribed strategies to increasingly flexible and model-driven contextual reasoning. We observe that capable models can exploit this flexibility, while small open-source models perform worse under HERMES than when reasoning inline, revealing a contextual reasoning gap. This motivates a learning framework for developing flexible contextual reasoning. Crucially, the same hierarchy provides a natural learning curriculum: models first learn under strongly steered harnesses with a simplified contextual reasoning decision space, then progressively assume greater control as harness steering is reduced during training. We therefore introduce HERMES-LEARN, a two-stage framework for learning flexible contextual reasoning. Together, HERMES and HERMES-LEARN provide a rigorous framework for studying contextual reasoning and a practical approach for teaching models to acquire this distinct capability.

We evaluate HERMES and HERMES-LEARN across model families and scales, from Qwen3-4B to frontier models, on mathematical reasoning benchmarks spanning AIME, HMMT, BeyondAIME (ByteDance-Seed, 2025), and IMO-AnswerBench (Luong et al., 2025), as well as the out-of-domain FrontierScience benchmark (Wang et al., 2026). We deliberately constrain each context to 8K tokens, creating a setting where effective long-horizon reasoning requires models to allocate and reuse context well. Our results show that: (1) HERMES unlocks test-time scaling for capable models, while revealing a substantial contextual reasoning gap in smaller open-source models; (2) HERMES-LEARN closes this gap, improving contextual reasoning without degrading the model's mathematical reasoning; (3) trained models learn adaptive contextual reasoning strategies that vary with both the problem structure and the reasoning progress; and (4) the learned contextual reasoning capabilities extend beyond the inference-time compute seen during training and transfer to complementary test-time scaling methods beyond HERMES.

Contributions. (1) *Contextual reasoning.* We study contextual reasoning as a model's capability to organize and sustain reasoning over long horizons. (2) HERMES. We introduce a simple, configurable family of harnesses built on delegation and digestion, with a steering hierarchy that progressively shifts control from the harness to the model. (3) *A learning framework to close contextual reasoning gap.* We show that capable models can turn additional inference-time compute into better performance under HERMES, while smaller open-source models struggle despite reasonable mathematical reasoning. We introduce HERMES-LEARN, a two-stage framework combining supervised fine-tuning and curriculum reinforcement learning to close this gap by learning to select and compose delegation strategies. (4) *Adaptive and generalizable contextual reasoning.* Learned strategies adapt to the problem and current reasoning state, generalize across benchmarks and model families, scale beyond the inference-time compute seen during training, and transfer to complementary test-time scaling methods.

2 RELATED WORK

Contextual reasoning. Prior test-time scaling approaches typically allocate additional contexts through inference primitives such as sequential reasoning (Wei et al., 2022; Madaan et al., 2023), parallel exploration (Cobbe et al., 2021; Wang et al., 2023), and aggregation (Venkatraman et al., 2025), or compose multiple primitives within a single inference procedure (Hu et al., 2026; Qin et al., 2026). Another line of work uses search to guide context allocation, such as tree-based search (Yao et al., 2023a; Zhou et al., 2023) using self-reflection and verifier-guided search (Lightman et al., 2024) using process reward models. When reasoning extends across multiple context windows, existing work primarily relies on delegation and compaction to coordinate context allocation and reuse. Prior methods typically prescribe specific delegation strategies, such as delegating recursively decomposed tasks (Prasad et al., 2024), recursively spawned work (Schroeder et al., 2024), or concurrently executable structures (Chi et al., 2025) to fresh contexts. A complementary line of work (Zhou et al., 2025; Kontonis et al., 2026) focuses on compressing prior reasoning into compact states for reuse in later contexts.

While prior work relies on external search procedures or prescribes contextual reasoning strategies in the inference procedure, HERMES studies contextual reasoning as a model capability. HERMES enables models to adaptively choose among multiple strategies for allocating additional contexts as

reasoning progresses. Moreover, our experiments (Sec. 6.2) show that this capability can also benefit complementary test-time scaling methods that use search with self-reflection.

Learning contextual reasoning. Recent work shows that explicitly training models to use the test-time inference procedures can improve test-time scaling. For example, `MRT` shapes long reasoning trajectories with progress-sensitive rewards (Qu et al., 2025), and `E3` trains models to compose generation, verification, and refinement during exploration (Setlur et al., 2025). Other methods train models for particular inference harnesses: `SPIRAL` trains parallel generation and aggregation (Hamid et al., 2026), `PaCoRe` trains synthesis across parallel branches (Hu et al., 2026), `RAO` trains recursive delegation (Gandhi et al., 2026), `Context-Folding` trains delegation of subtasks to fresh branches together with compaction of intermediate reasoning (Sun et al., 2025), and `Reasoning Cache` trains continuation from compacted reasoning progress (Wu et al., 2026a). While these methods demonstrate the value of aligning training with inference harnesses, they typically encode contextual reasoning strategies in the harness, leaving models to learn how to execute them effectively (App. B). In contrast, HERMES-LEARN teaches models to autonomously select, instantiate, and compose different contextual reasoning strategies, through the natural curriculum induced by the progressively reduced steering of the HERMES hierarchy.

Table 1: Comparison of HERMES with existing methods. Prior methods mostly prescribe contextual reasoning strategies for allocating and reusing contexts, then train models for that particular harness. HERMES-LEARN instead trains models to select among different contextual reasoning strategies through the learning curriculum induced by the varying steering of the HERMES hierarchy. *Model-driven strategy selection* captures whether contextual reasoning strategies are decided by the model rather than by the harness; *Context compaction*, whether prior reasoning can be compressed and reused; *Recursive delegation*, whether further contexts can be allocated recursively through the same delegation interface.

Method	Model-driven strategy selection	Context compaction	Recursive delegation
`SPIRAL`	✗	✗	✗
`PaCoRe`	✗	✓	✗
`RAO`	✗	✗	✓
`Context-Folding`	✗	✓†	✗
`Reasoning Cache`	✗	✓	✗
HERMES	✓	✓	✓

†Context-Folding compacts a delegated branch when it returns.

3 HERMES: A FAMILY OF SIMPLE, FLEXIBLE HARNESSES

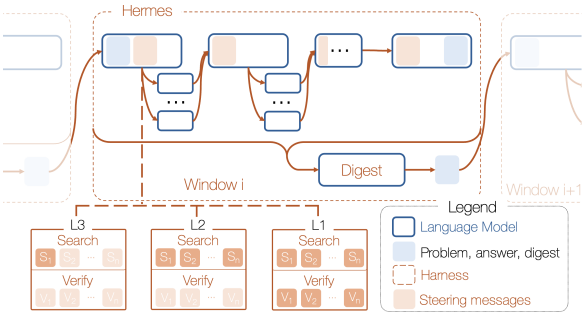

Figure 1: **Overview of HERMES.** HERMES enables contextual reasoning across multiple context windows through two mechanisms: *delegation*, which allocates fresh contexts to search or verification strategies, and *digestion*, which compacts accumulated progress for reuse in later contexts. The HERMES hierarchy (L3–L1) progressively shifts control over delegation decisions from the harness to the model. L3 prescribes the strategy; L2 prescribes a strategy category (search or verification) while allowing the model to select strategies within it; and L1 lets the model choose both as reasoning unfolds.

In this section, we introduce HERMES and evaluate how different language models can leverage its harnesses for test-time scaling.

Desiderata. We design HERMES to be *simple*, *flexible*, and *configurable*. *Simple*: HERMES introduces only two generic mechanisms, *delegation* and *digestion* to allocate fresh contexts and carry useful information across them. *Flexible*: these mechanisms do not prescribe a fixed form of computation; instead, they allow the model to decide how additional contexts should be used as reasoning unfolds. *Configurable*: steering controls how much guidance the harness provides for delegation, from prescribing a specific strategy to letting the model choose one itself.

Contextual reasoning via delegation and digestion. HERMES gives a model two ways to continue reasoning beyond a single context window. It can assign tasks

to fresh contexts and use the results, or carry its own progress into a fresh context and continue working. We call these operations *delegation* and *digestion*, respectively. Delegation and digestion are complementary primitives for contextual reasoning, expanding reasoning across contexts in parallel and sequentially (Fig. 1).

Within each context window, HERMES runs a ReAct-style interaction loop (Yao et al., 2023b) (Fig. 1). At step t, the agent receives an input u_t, such as the original problem, a message from the harness, or a result from a subagent. It then produces an action a_t: it may continue reasoning, return an answer, or call `launch_subagent(task)`. This tool is how the agent delegates work. The harness invokes the same model on the specified task in a fresh context and returns a compact report as the agent's next input. We call this model working on the specified task a *subagent*. Formally, $h_t = (u_1, a_1, \ldots, u_t)$ is the history available when the agent chooses a_t; under harness H with configuration c, the action is sampled from $\pi_\theta(\cdot \mid h_t; H_c)$. An agent has up to T steps in each context window. At each of the first $T-1$ steps, it may launch up to B subagents; at step T, it can no longer delegate and returns its best current answer. Subagents use the same delegation interface, so they may launch further subagents up to depth D. At the maximum depth, they reason inline without further delegation. The interaction can also end earlier if the context window is exhausted.

Digestion allows the agent to continue *its own* work across windows. At the end of a window, it compresses its accumulated progress into a digest. If reasoning continues, the harness starts a fresh window with the original problem and that digest, rather than the full interaction history, for up to W windows. Thus, B and D govern how work branches across contexts in parallel, while W governs how long it can continue sequentially.

Steerable control over delegation. Effective delegation requires more than sound mathematical reasoning. An agent must also decide when to delegate, what task to assign, and how to use each fresh context, for example by independently searching for a solution, exploring a different approach, decomposing the problem, or verifying existing reasoning. These contextual reasoning decisions can be challenging, particularly for smaller open-source models. HERMES therefore varies how strongly the harness steers delegation, progressively shifting these decisions from the harness to the model. We organize delegation strategies into two broad categories: *search* through independent attempts on the full problem, alternative approaches, or decomposition into simpler subproblems, and *verification* of complete solutions, specific reasoning steps, or disagreements between candidate solutions. These categories induce three steering levels $\ell \in \{\mathsf{L3}, \mathsf{L2}, \mathsf{L1}\}$ (Fig. 1). At L3, the harness specifies a strategy and the agent instantiates it as a concrete task. At L2, the harness specifies only the strategy category, while the agent selects one or more strategies within it. At L1, the agent selects from the full strategy set, deciding how to formulate tasks for its subagents as reasoning unfolds. Thus, L3–L1 progressively shift control over delegation from the harness to the agent, while providing a natural curriculum for learning these contextual reasoning decisions. Several prior methods can be viewed as fixed-strategy special cases of HERMES (Wu et al., 2026a; Gandhi et al., 2026; Hamid et al., 2026).

Together, these parameters define a harness configuration $c = (T, B, D, W, \ell)$, making HERMES configurable in both how much additional context is available and how much control over its use is given to the model. Additional details can be found in App. C.

Table 2: HERMES *enables test-time scaling for capable models.* Avg@16 (%, ↑) under inline reasoning and HERMES L1 and L2 on the pooled and synthetic benchmarks. Inline reasoning measures mathematical reasoning within a single 8K-token context, while HERMES provides up to 56K tokens across contexts. Claude `Sonnet 5` and `DeepSeek-V3.2` match or improve over inline reasoning with HERMES; `Qwen3-4B` degrades, exposing a contextual reasoning gap.

	Pooled ($n = 238$)			Synthetic ($n = 88$)		
Model	Inline	L1	L2	Inline	L1	L2
`Qwen3-4B`	**37.1**	16.2	15.1	**50.7**	26.8	26.0
`Sonnet 5`	55.3	**76.7**	76.5	87.0	94.7	**96.2**
`DeepSeek-V3.2`	56.2	**58.4**	57.4	78.1	**86.2**	83.7

Test-time scaling with HERMES relies on contextual reasoning. HERMES provides additional test-time compute, but models must use it effectively to improve performance. We compare `Qwen3-4B-Instruct-2507` (hereafter `Qwen3-4B`), our base model, with Claude `Sonnet 5` and `DeepSeek-V3.2` under *inline reasoning* and HERMES L1 and L2. Inline reasoning measures mathematical reasoning within a single 8K-token context, where a non-thinking model solves the problem directly without delegation or

digestion. For both HERMES harnesses, we use $T = 3$, $B = 3$, $D = 1$, and $W = 1$, with an 8K-token context per agent. We set $D = 1$ and $W = 1$, so only the root agent delegates and digestion is disabled. With $T = 3$ and $B = 3$, HERMES can use up to 56K tokens[1] of total context, compared with 8K tokens for inline reasoning. We evaluate whether models can translate this additional context into improved performance on a pooled set of AIME and HMMT problems (App. E.1) and a synthetic benchmark constructed to benefit from different contextual reasoning strategies (App. D.2).

As shown in Tab. 2, Claude `Sonnet 5` and `DeepSeek-V3.2` match or exceed their inline performance with HERMES, showing that they can combine additional context with mathematical reasoning effectively. In contrast, `Qwen3-4B` degrades substantially despite reasonable inline mathematical reasoning, revealing a gap in the contextual reasoning needed to benefit from additional context. On the synthetic benchmark, `Sonnet 5` also performs slightly better under the more prescriptive L2 harness than under L1, suggesting that even capable models may benefit from different levels of steering. These results motivate learning contextual reasoning capabilities directly.

4 LEARNING CONTEXTUAL REASONING

HERMES exposes several learning problems in contextual reasoning. Given the current history h_t, a model must learn *what to delegate* to fresh contexts, *what to preserve* when generating a digest, and *how to reason* from the resulting digest. Prior work has studied the latter two problems: models can be trained to continue reasoning from compressed histories (Wu et al., 2026a;b), while recent instruction-tuned models already exhibit summarization capabilities that enable effective digest generation in many cases. Further improving digestion through downstream task outcomes can suffer from difficult credit assignment across multiple subsequent context windows.

In this work, we focus on learning delegation. Unlike digestion, delegation decisions are made by the root agent as reasoning unfolds and can be optimized directly from task outcomes. We introduce HERMES-LEARN, a two-stage framework that first teaches useful delegation behaviors through supervised fine-tuning (SFT), then develops autonomous strategy selection through reinforcement learning (RL). For digestion, we rely on the model's existing ability to summarize its accumulated reasoning and continue from the resulting digest instead of explicitly training it.

Stage I: Supervised Fine-Tuning. We first warm-start the base model with supervised fine-tuning on successful teacher trajectories. We collect trajectories from strong models operating under L1 and L2, exposing the model to delegation under different levels of steering. We retain trajectories in which the root agent delegates at least once and produces the correct final answer. During training, since we set $D = 1$, all delegation decisions occur at the root, so we train only on root-agent trajectories and retain subagent outputs only as part of the root history h_t. Let $\mathcal{T}$ denote these retained root-agent trajectories. Then the SFT stage optimizes

$$\arg\min_{\theta} -\sum_{\tau \in \mathcal{T}} \sum_{t=1}^{T} \sum_{j=1}^{|a_t|} \log \pi_\theta \left(a_{t,j} \mid h_t, a_{t,<j} \right),$$

where loss is computed only over assistant tokens. This stage exposes the model to successful delegation under different levels of steering and provides a strong initialization for RL.

Stage II: Reinforcement learning. Initializing from the SFT policy $\pi_{\theta_{\text{SFT}}}$, we use on-policy reinforcement learning with outcome-based rewards to learn more autonomous delegation. We use Dr. GRPO (Liu et al., 2025) with the DAPO clip-higher strategy and dynamic sampling (Yu et al., 2025). For each problem $p \in \mathcal{D}$ and HERMES harness H, we sample G root-agent rollouts from the current policy. Each rollout receives a binary reward of 1 if its final answer is correct and 0 otherwise. Following dynamic sampling (Yu et al., 2025), we train only on groups containing both correct and incorrect rollouts.

As in Stage I, we optimize only the root-agent trajectory. Extending training to subagent trajectories would additionally require optimizing reasoning over policy-generated subtasks and assigning credit to individual subagent outcomes. Recursive Agent Optimization (RAO) (Gandhi et al., 2026), for

[1] With $T = 3$, $B = 3$, $D = 1$, and $W = 1$, the root agent uses one 8K-token window and may launch up to six subagents, each with its own 8K-token window. This yields a maximum of $7 \times 8\text{K} = 56\text{K}$ tokens.

example, addresses this setting by jointly optimizing nodes in a recursive execution tree using node-local rewards and trajectory weighting. In this work, we focus on learning root-level delegation decisions and leave joint root–subagent optimization to future work.

We optimize root-agent trajectories with the following Dr. GRPO objective:

$$\mathcal{J}(\pi_\theta) = \mathbb{E}_{p\sim\mathcal{D},H\sim S(\mathcal{H}),\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|p;H)}\left[\frac{1}{G}\sum_{i=1}^{G}\sum_{t=1}^{T}\sum_{j=1}^{|a_t^i|}\min\left(r_{t,j}^i(\pi_\theta)\hat{A}_i,\ \text{clip}\big(r_{t,j}^i(\pi_\theta),1-\epsilon_{\text{low}},1+\epsilon_{\text{high}}\big)\hat{A}_i\right)\right],$$

where

$$r_{t,j}^i(\pi_\theta) = \frac{\pi_\theta(a_{t,j}^i \mid h_t^i, a_{t,<j}^i)}{\pi_{\theta_{\text{old}}}(a_{t,j}^i \mid h_t^i, a_{t,<j}^i)}, \qquad \hat{A}_i = R_i - \frac{1}{G}\sum_{k=1}^{G} R_k.$$

Here, R_i is the binary outcome reward for rollout i. Since the reward is defined at the rollout level, the same advantage $\hat{A}_i$ is applied to all root-agent tokens in that rollout.

The sampler $S(\mathcal{H})$ defines a curriculum over the HERMES hierarchy. We begin RL with L2 and gradually increase the proportion of L1, while retaining a small fraction of L2 throughout training. This progressively reduces harness steering, requiring the model to take greater control over which delegation strategies to use as reasoning unfolds.

Data augmentation. Competition math problems often do not expose delegation strategies uniformly: for example, independent attempts may arise more frequently than decomposition. To provide more balanced coverage of delegation strategies during training, we generate synthetic problems with structures designed to favor different search and verification strategies. *Splitting* combines multiple independent problems into a single instance whose final answer depends on all of them, creating opportunities for parallel delegation (Fig. 2). *Chaining* introduces dependencies that favor sequential delegation; *verifying* introduces potentially incorrect intermediate results; and *multi-* and *single-attempt* problems vary whether additional independent attempts are useful (see details in App. D).

These controlled structures expose the model to a broader and more balanced range of contextual reasoning behaviors and encourage it to adapt its delegation strategy to the problem. We use the same generation procedure to construct both synthetic training examples and a held-out synthetic evaluation set.

Let n be a natural number, and let $f(n)$ denote the sum of the digits of $n^2 + 1$ when written in base 10. Define the iterates $f^{(1)}(n) = f(n)$ and $f^{(k+1)}(n) = f(f^{(k)}(n))$ for $k \geq 1$. Let **m** be the value of $f^{(100)}(1990)$.

Let **p** be the number of four-digit numbers whose digits sum to 12.

Find the remainder when **m**+**p** is divided by 1000.

Golden answer: 353

Figure 2: *Synthetic splitting example.* Constructed by combining two independent `AceReason` questions into a single problem whose final answer depends on both, encouraging parallel delegation.

5 EXPERIMENTAL SETUP AND MAIN RESULTS

We focus on answering four research questions: **(RQ1)** Does HERMES-LEARN improve contextual reasoning? **(RQ2)** What contextual reasoning behaviors emerge through training? **(RQ3)** Which components of the training recipe drive these improvements? **(RQ4)** Do these gains generalize to new settings? We first describe the training and evaluation setup and baselines, and then address each question in turn.

Training setup. We train the `Qwen3-4B-Instruct-2507` base model on ACEREASON-5K, a cleaned subset of AceReason-Math (Chen et al., 2025) containing 5,239 mathematical reasoning problems. For SFT, we train on teacher trajectories collected from both `Sonnet 5` and `DeepSeek-V3.2`, using L1 and L2 harnesses. For RL, we use a curriculum that shifts from L2 to L1 while retaining balanced exposure to both harnesses throughout training, and augment training set with 500 synthetic problems. See App. E.3 for details.

Evaluation benchmarks and protocols. We evaluate on a suite of mathematical reasoning benchmarks: (1) a cleaned **pooled** benchmark of 238 AIME and HMMT problems from competitions

Table 3: SFT and RL progressively improve L1 performance (Avg@16 (%, ↑)), while inline performance changes relatively little. SFT + RL under L1 also exceeds the base model's 56K inline reference on all four mathematical benchmarks. Trained results are mean ± standard deviation over three seeds. **Bold** and underlined mark the best and second-best point estimates per dataset.

	Pooled		Synthetic		BeyondAIME		IMO-Answer		FrontierScience	
Model	Inline	L1	Inline	L1	Inline	L1	Inline	L1	Inline	L1
Base	37.1	16.2	50.7	26.8	24.8	12.2	36.2	18.2	23.6	8.8
GRPO	40.4 ± 1.0	16.5 ± 1.6	61.5 ± 1.6	27.9 ± 1.5	31.1 ± 0.6	14.1 ± 1.4	37.4 ± 0.6	19.0 ± 1.4	24.6 ± 1.5	8.9 ± 0.6
Base (56K)	41.8	–	59.7	–	31.1	–	40.3	–	24.6	–
GRPO (56K)	41.3 ± 1.2	–	62.2 ± 2.1	–	30.9 ± 1.2	–	37.5 ± 0.6	–	**25.1** ± 1.5	–
HERMES-LEARN										
SFT	35.3 ± 0.3	39.9 ± 0.7	52.7 ± 1.6	57.2 ± 2.4	23.5 ± 0.4	31.0 ± 0.6	36.7 ± 0.5	39.2 ± 0.2	18.4 ± 1.4	18.9 ± 0.5
SFT + RL	36.4 ± 0.1	**47.3** ± 1.3	55.2 ± 0.9	**74.0** ± 2.9	27.5 ± 1.3	**38.5** ± 0.5	36.4 ± 0.4	**44.0** ± 0.8	19.1 ± 0.4	24.1 ± 0.9

between 2023 and 2026; (2) a **synthetic** benchmark of 88 problems generated from the pooled benchmark using the same augmentation pipeline as our training data; and (3) advanced benchmarks, BeyondAIME (ByteDance-Seed, 2025) and IMO-AnswerBench (Luong et al., 2025), together with the out-of-domain FrontierScience benchmark (Wang et al., 2026). We use an 8K-token context per agent, a practical and deliberately constrained setting that makes reasoning across multiple contexts necessary. We report average accuracy over 16 sampled solutions (Avg@16) under both inline reasoning and HERMES L1. Inline performance reflects mathematical reasoning within a single context, while improvements under L1 reflect the model's ability to use additional contexts effectively. Training and evaluation use the default HERMES configuration: $T = 3$, $B = 3$, $D = 1$, and $W = 1$. Under this configuration, only the root agent delegates, with up to three subagents per delegation step, and digestion is disabled (see App. E).

Baselines. In addition to the base model, we include two baselines designed to isolate the source of improvements from HERMES-LEARN. First, we evaluate inline reasoning with a 56K-token context, matching the maximum total context available to a 3-step HERMES run[2]. Second, we train with GRPO under inline reasoning with an 8K-token context, controlling for improvements from RL without training the model to reason across contexts.

(RQ1) HERMES-LEARN enables test-time scaling with HERMES. It transforms additional context from a liability into a source of test-time scaling (Tab. 3). The base model performs substantially worse under L1 than inline reasoning across all benchmarks, showing that access to additional contexts alone is insufficient. SFT reverses this trend, and RL further improves L1 performance. In contrast, inline performance changes relatively little through HERMES-LEARN, indicating that the gains primarily come from learning to use additional contexts rather than from improved mathematical reasoning. This distinction is reinforced by the inline GRPO baseline, which improves inline performance but continues to degrade under L1. Finally, SFT + RL under L1 outperforms the 56K inline reference of both the base and GRPO models across all four mathematical benchmarks, showing that improved mathematical reasoning alone is insufficient to realize the benefits of additional context and highlighting the role of HERMES-LEARN in enabling effective contextual reasoning.

(RQ2) Training makes strategy selection problem and progress-dependent. To understand how training changes the use of HERMES, we use an LLM judge to label the problem solving strategies executed under L1 on the pooled and synthetic benchmarks (App. E.4). Fig. 3 reveals two patterns. First, strategy selection depends on the problem: `Split` becomes much less common on the pooled benchmark after training, but

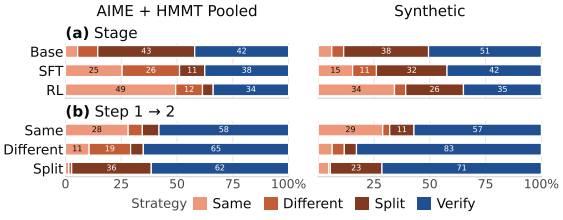

Figure 3: HERMES-LEARN makes strategy selection problem- and progress-dependent.

[2]With $T = 3$ and $B = 3$, HERMES uses one root context and up to six subagent contexts, for a maximum of $7 \times 8\text{K} = 56\text{K}$ tokens. We include this baseline to distinguish gains from reasoning across contexts from those obtained by simply providing a larger single context.

remains prevalent on synthetic problems designed to benefit from decomposition (see App. C.1 for strategy definitions). Second, strategy selection depends on reasoning progress: in the second delegation step, the model often switches to verification; otherwise, it tends to continue the strategy used in the first step. Together, these results show that HERMES-LEARN does not learn a fixed delegation pattern, but selects strategies based on the problem and the progress made so far. Additional analyses and example rollouts are provided in Apps. E.5 and F.

6 ANALYSIS AND DISCUSSION

6.1 (RQ3) TEACHER DIVERSITY, CURRICULUM, AND SYNTHETIC DATA SHAPE HERMES-LEARN.

We ablate the main components of HERMES-LEARN on the pooled benchmark (Tab. 4). For SFT, we vary the source and amount of teacher demonstrations. For RL, we study exposure to L1 and L2 harnesses, their training order, and data augmentation.

Combining teachers yields the strongest SFT model. Training on the full set of trajectories from both `Sonnet 5` and `DeepSeek-V3.2` gives the best SFT performance (Both (Full)). When we match the mixed-teacher dataset to the token budget of a single teacher (Both (Matched)), however, its performance is similar to `DeepSeek-V3.2` alone. This suggests that much of the gain from the full dataset comes from having more high-quality demonstrations rather than teacher diversity alone. Notably, `DeepSeek-V3.2` is a much better teacher than `Sonnet 5`, despite performing worse under HERMES (Tab. 2), indicating that stronger inference performance does not necessarily translate into better supervision for SFT.

Curriculum provides the largest gains when paired with synthetic problems. We ablate two choices in RL training: how the model is exposed to L1 and L2, and exposure to synethic data. Mix and Curriculum use the same number of L1 and L2 episodes, differing only in their order. Similarly, we compare 500 synthetic problems with 500 LLM-rephrased ACEREASON-1K problems to control for the effect of additional training examples. On the original training set, Curriculum and Mix perform similarly, suggesting that ordering L2 before L1 provides only a modest benefit by itself. In contrast, combining the curriculum with synthetic problems produces the strongest L1 performance. Rephrasing the same number of existing problems does not yield a comparable gain, suggesting that exposure to new problem structures is more useful than additional training volume alone. Inline performance remains largely unchanged, while the improvements concentrate under L1, consistent with HERMES-LEARN improving how the model uses HERMES rather than its in-context mathematical reasoning.

Table 4: **HERMES-LEARN ablations** on the pooled benchmark. **(a)** For SFT, *Both (Matched)* uses a token-matched subset of trajectories from both teachers, while *Both (Full)* uses the full combined dataset. **(b)** For RL, *Mix* interleaves L1 and L2, while *Curriculum* trains on L2 before L1 with matched exposure; *Rephrased* and *Synthetic* each add 500 problems. Avg@16 (%, ↑), mean ± s.d. over three seeds. **Bold** and underlined mark best and second-best results.

(a) SFT ablations

Teachers	Inline	L1
`Sonnet`	31.5 ± 0.5	30.1 ± 0.9
`DeepSeek`	<u>34.5</u> ± 0.4	<u>39.3</u> ± 0.2
Both (Matched)	34.4 ± 0.7	38.9 ± 0.4
Both (Full)	**35.3** ± 0.3	**39.9** ± 0.7

(b) RL ablations

Schedule	Data Aug.	Inline	L1
Harness schedule on original data			
L1 only	–	36.5 ± 1.5	46.1 ± 0.8
L2 only	–	36.2 ± 0.2	45.4 ± 0.6
Mix	–	35.7 ± 0.7	46.1 ± 0.8
Curriculum	–	36.3 ± 0.7	<u>46.3</u> ± 1.9
With data augmentation			
Mix	+ Synthetic	35.9 ± 1.0	45.3 ± 1.8
Curriculum	+ Synthetic	36.4 ± 0.1	**47.3** ± 1.3
Curriculum	+ Rephrased	36.0 ± 0.3	45.6 ± 1.2

6.2 (RQ4) HERMES-LEARN GENERALIZES BEYOND ITS TRAINING SETTING

Our main experiments use the same HERMES configuration during training and evaluation. We now test three forms of generalization: larger inference-time compute budgets, combinations with other test-time scaling methods, and a different model family and scale.

Table 5: **HERMES-LEARN generalizes across compute budgets, test-time scaling methods, and model families.** Avg@16 (%, ↑) on the pooled benchmark. **(a)** We independently vary the number of context windows (W), subagents (B), and delegation depth (D) beyond the default HERMES configuration used for training and evaluation: $T = 3$, $B = 3$, $D = 1$, and $W = 1$. **(b)** We evaluate Recursive Self-Aggregation (RSA) and the DeepSeekMath (DSM) Agent, both alone and combined with L1. **(c)** We apply the same training pipeline to `Olmo-3-7B-Instruct`. All experiments use the best performing HERMES-LEARN model. **Bold** and underlined mark best and second-best results where applicable.

(a) Scaling inference-time compute

Scaling Knob	Value	SFT	SFT + RL
Default	–	39.9	47.3
W	2	44.3	52.2
	3	45.2	**53.0**
B	2	39.7	46.4
	4	39.7	**49.7**
D	2	40.8	**55.6**

(b) Other scaling methods

Model	RSA	DSM
Base	46.0	24.4
SFT	44.9	11.3
SFT + RL	<u>50.5</u>	15.1
With L1		
SFT	49.9	<u>52.1</u>
SFT + RL	**55.0**	**52.5**

(c) Transfer to `Olmo-3-7B`

Model	Inline	L1
Base	30.5	4.8
SFT	29.3	33.4
SFT + RL	30.5	**40.6**

HERMES-LEARN extrapolates to larger inference-time compute budgets. Starting from the default HERMES configuration used during training and evaluation ($T = 3$, $B = 3$, $D = 1$, $W = 1$), we independently increase compute through longer sequential reasoning (↑ W), more subagents per delegation step (↑ B), and deeper delegation (↑ D). Tab. 5 shows that SFT benefits primarily from additional context windows, with little gain from more subagents or deeper delegation. After RL, performance improves along all three dimensions, showing that the learned contextual reasoning strategies generalize to inference configurations not seen during training.

HERMES-LEARN benefits test-time scaling methods beyond HERMES. We evaluate the learned models with Recursive Self-Aggregation (RSA) (Venkatraman et al., 2025) and the DeepSeekMath (DSM) Agent (Shao et al., 2025), two methods that use self-generated intermediate results to guide later reasoning. Under RSA, performance improves over the base model, with further gains when combined with L1[3]. DSM alone degrades sharply under the 8K-token context limit, but combining it with L1 recovers and substantially improves performance. These results show that the contextual reasoning learned by HERMES-LEARN can complement other forms of test-time scaling.

HERMES-LEARN transfers to a different model family and scale. We apply the same training pipeline to `Olmo-3-7B-Instruct`, replacing `Qwen3-4B-Instruct-2507` as the base model. Like `Qwen3-4B`, the `Olmo` model also initially struggles to use the L1 harness effectively. SFT reverses this degradation, while RL further improves its performance, all while keeping inline reasoning performance unchanged. Together, these results provide evidence that HERMES-LEARN can improve contextual reasoning beyond the model family and scale used in our main experiments, suggesting that the approach may extend to a broader range of models.

7 CONCLUSION

We introduced HERMES, a family of simple, configurable harnesses for test-time scaling beyond a single context window. Through delegation and digestion, HERMES progressively shifts control over context allocation and reuse from the harness to the model. We showed that frontier models can exploit this flexibility, while smaller open-source models exhibit a substantial contextual reasoning gap. To address this gap, we introduced HERMES-LEARN, a training recipe that improves contextual reasoning while preserving in-context reasoning performance. Trained models scale more effectively with additional inference-time compute, adapt their strategies as reasoning unfolds, and transfer across model families and scales. Together, our results establish contextual reasoning as a learnable model capability and an important component of long-horizon test-time scaling. We discuss limitations and future work in App. A.

[3]For RSA, we use its summarizer to compact candidate solutions before aggregation.

DISCLOSURE STATEMENTS

AI USE STATEMENT

We used generative AI tools throughout several stages of this work. For data generation and preparation, we used language models to generate synthetic training and evaluation problems and to assist with cleaning, filtering, and reformatting datasets, including the synthetic-data pipelines and pooled AIME and HMMT datasets described in the appendices.

We also used generative AI to support research methodology and experimentation. This included discussing experimental design, proposing ablations, suggesting experimental parameters, debugging and diagnosing training runs, and helping interpret results. Coding agents were used extensively to accelerate implementation of our methods, experimental infrastructure, and analysis pipelines, and to help shepherd training runs.

Coding agents required substantial human supervision. They sometimes introduced implementation errors, made inconsistent changes, overlooked requirements, or reported tasks as complete when parts remained unfinished. They could also settle prematurely on plausible explanations for failures without adequately validating them. The authors therefore inspected code and diffs, checked logs and experimental outputs, tested implementations, and repeatedly redirected agents when necessary. We treated agent-generated implementations and diagnoses as proposals to verify rather than authoritative results.

AI tools also supported qualitative and thematic analysis, including identifying recurring strategy patterns in model trajectories. We found models could be overly eager to impose taxonomies or generalize from limited examples, and their classifications were sometimes sensitive to prompt framing. The authors therefore manually reviewed examples and iteratively revised the resulting categories and interpretations.

Generative AI was additionally used for literature review and writing: identifying and summarizing relevant work, brainstorming connections, refining paper structure, drafting and editing text, and improving clarity and readability. The authors frequently rewrote generated text and used iterative discussions with language models to clarify ideas, compare formulations, and simplify explanations. Literature summaries were also independently checked against the original papers when they informed scientific claims or positioning, since model summaries could omit qualifications or emphasize prompt-dependent aspects of a work.

Overall, we found generative AI most useful as an interactive tool for iteration and acceleration, but its outputs required continued author judgment and verification. All AI-assisted research decisions, analyses, code, and writing were reviewed by the authors. We take responsibility for the final content of this work, including all claims, experimental results, code, and text produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All experiments were run on AWS `p4d.24xlarge` instances with 8 NVIDIA A100 40,GB GPUs per node. The warm-up stage used a single node (8 GPUs), while each GRPO run used 4 nodes (32 GPUs) with FSDP data parallelism and ran for approximately 83 hours. Evaluations used a single node with eight independent vLLM workers. Our RL stack uses PyTorch 2.8.0, vLLM 0.11.0, Ray 2.56.0, and verl 0.7.0; the warm-up stage uses Transformers 5.14.1 and TRL 1.8.0. Full harness specifications, hyperparameters, prompts, and example rollouts are provided in the appendix.

ETHICS STATEMENT

This work does not involve human subjects or the collection of personally identifiable or sensitive data. Our experiments study language-model reasoning using mathematical reasoning tasks. The primary ethical considerations concern the potential downstream uses of improved long-horizon reasoning capabilities; we discuss these separately in the Societal Impact section.

SOCIETAL IMPACT

This work is foundational and studies how language models can learn to reason across and reuse multiple context windows. Such methods may improve performance on longer-horizon problems and make smaller models more capable under additional test-time compute, potentially broadening access to useful reasoning systems.

These benefits also introduce risks. First, test-time scaling increases inference compute and may therefore increase energy use and deployment cost; its environmental impact depends on the hardware, workload, and energy mix used in deployment. Second, improved contextual reasoning may enable agentic systems to sustain more complex, longer-horizon behavior, potentially amplifying risks when such systems are deployed in high-stakes settings or without appropriate oversight. Finally, the models trained in this work are evaluated on a narrow set of reasoning tasks and are not evaluated for safety, robustness, privacy, or reliability. Our results should therefore not be interpreted as evidence of suitability for safety-critical or consequential applications without additional evaluation and safeguards.

These concerns are not specific to HERMES, but arise more broadly as test-time scaling and agentic systems become more capable. Future work should evaluate the efficiency, reliability, and safety of learned contextual-reasoning policies as they are applied to longer-horizon and real-world tasks.

REFERENCES

Levent Alpöge and Ralph Furman. More than two thirds of the zeta zeros are simple and on the critical line. *arXiv preprint arXiv:2608.13637*, 2026.

ByteDance-Seed. Beyondaime: Advancing math reasoning evaluation beyond high school olympiads, 2025.

Yang Chen, Zhuolin Yang, Zihan Liu, Chankyu Lee, Peng Xu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acereason-nemotron: Advancing math and code reasoning through reinforcement learning. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 110320–110345. Curran Associates, Inc., 2025. doi: 10.52202/085713-3685. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/9fc2aa7f8a66eed231007b8bee4485cb-Paper-Conference.pdf.

Zewen Chi, Li Dong, Qingxiu Dong, Yaru Hao, Xun Wu, Shaohan Huang, and Furu Wei. The era of agentic organization: Learning to organize with language models, 2025. URL https://arxiv.org/abs/2510.26658.

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

Apurva Gandhi, Satyaki Chakraborty, Xiangjun Wang, Aviral Kumar, and Graham Neubig. Recursive agent optimization. *arXiv preprint arXiv:2605.06639*, 2026.

Jubayer Ibn Hamid, Ifdita Hasan Orney, Michael Y. Li, Omar Shaikh, Yoonho Lee, Dorsa Sadigh, Chelsea Finn, and Noah Goodman. SPIRAL: Learning to search and aggregate, 2026. URL https://arxiv.org/abs/2606.23595.

Jingcheng Hu, Yinmin Zhang, Shijie Shang, Xiaobo Yang, Yue Peng, Zhewei Huang, Hebin Zhou, Xin Wu, et al. PaCoRe: Learning to scale test-time compute with parallel coordinated reasoning, 2026. URL https://arxiv.org/abs/2601.05593.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Vasilis Kontonis, Yuchen Zeng, Shivam Garg, Lingjiao Chen, Hao Tang, Ziyan Wang, Ahmed Awadallah, Eric Horvitz, John Langford, and Dimitris Papailiopoulos. Memento: Teaching LLMs to manage their own context. *arXiv preprint arXiv:2604.09852*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=5PAF7PAY2Y.

Thang Luong, Dawsen Hwang, Hoang H. Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, Alex Zhai, Clara Huiyi Hu, Henryk Michalewski, Jimin Kim, Jeonghyun Ahn, Junhwi Bae, Xingyou Song, Trieu H. Trinh, Quoc V. Le, and Junehyuk Jung. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025. URL https://aclanthology.org/2025.emnlp-main.1794/.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

OpenAI. Finite time blowup for the euler equation. Preprint, 2026. URL https://cdn.openai.com/pdf/315b36cd-ec98-4023-8342-93345194ece1/euler.pdf.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, 2024. doi: 10.18653/v1/2024.findings-naacl.264. URL https://aclanthology.org/2024.findings-naacl.264/.

Peijia Qin, Qi Cao, and Pengtao Xie. ATLAS: Agentic test-time learning-to-allocate scaling, 2026. URL https://arxiv.org/abs/2606.01667.

Yuxiao Qu, Matthew Y. R. Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement fine-tuning, 2025. URL https://arxiv.org/abs/2503.07572.

Philip Schroeder, Nathaniel Morgan, Hongyin Luo, and James Glass. THREAD: Thinking deeper with recursive spawning, 2024. URL https://arxiv.org/abs/2405.17402.

Amrith Setlur, Matthew Y. R. Yang, Charlie Snell, Jeremy Greer, Ian Wu, Virginia Smith, Max Simchowitz, and Aviral Kumar. e3: Learning to explore enables extrapolation of test-time compute for LLMs, 2025. URL https://arxiv.org/abs/2506.09026.

Zhihong Shao, Yuxiang Luo, Chengda Lu, ZZ Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*, 2025.

Herbert A. Simon. *Models of Man, Social and Rational: Mathematical Essays on Rational Human Behavior in a Social Setting*. John Wiley & Sons, New York, 1957.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon LLM agent via context-folding, 2025. URL https://arxiv.org/abs/2510.11967.

Siddarth Venkatraman, Vineet Jain, Sarthak Mittal, Vedant Shah, Johan Obando-Ceron, Yoshua Bengio, Brian R. Bartoldson, Bhavya Kailkhura, Guillaume Lajoie, Glen Berseth, et al. Recursive self-aggregation unlocks deep thinking in large language models, 2025. URL https://arxiv.org/abs/2509.26626.

Miles Wang, Robi Lin, Kat Hu, Joy Jiao, Neil Chowdhury, Ethan Chang, and Tejal Patwardhan. Frontierscience: Evaluating AI's ability to perform expert-level scientific tasks. *arXiv preprint arXiv:2601.21165*, 2026.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2022. URL https://arxiv.org/abs/2201.11903.

Ian Wu, Yuxiao Qu, Amrith Setlur, and Aviral Kumar. Reasoning cache: Continual improvement over long horizons via short-horizon rl. *arXiv preprint arXiv:2602.03773*, 2026a.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. ReSum: Unlocking long-horizon search intelligence via context summarization, 2026b. URL https://arxiv.org/abs/2509.13313.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b. URL https://arxiv.org/abs/2210.03629.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=2a36EMSSTp.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025. URL https://arxiv.org/abs/2506.15841.

APPENDIX CONTENTS

A Limitations and Future Work 14
B Additional Discussion of Related Work 15
C HERMES Harness Details 15
C.1 Harnesses in the HERMES Hierarchy 15
C.2 Prompts 16
D Data Augmentation Details 20
D.1 Synthetic Problem Generation Pipeline 20
D.2 Synthetic Problem Construction 22
E Experiment Details 24
E.1 Pooled Benchmark Details 24
E.2 Inference under HERMES Harnesses 24
E.3 Training Experiments Details 24
E.4 Labeling Executed Contextual Reasoning Strategies 25
E.5 Additional Delegation Behavior Analysis 29
F Example Rollouts 30
F.1 L1 Rollout 30
F.2 L2-SV Rollout 32
F.3 L1 Rollout on a FrontierScience Problem 33
F.4 L1 Rollouts under Larger Inference-time Compute Budgets 35

A LIMITATIONS AND FUTURE WORK

Our evaluation studies contextual reasoning in a controlled setting. We evaluate primarily on mathematical reasoning benchmarks (AIME and HMMT) and a science benchmark, using an artificially constrained 8K-token context window. This makes multi-context reasoning necessary while keeping training and evaluation tractable, and is similar to controlled context constraints used in prior work on recursive agents (Gandhi et al., 2026). However, we do not test whether the same gains persist with much larger native context windows or in long-horizon domains with evolving state, tool use, and delayed feedback. Software engineering benchmarks such as SWE-bench (Jimenez et al., 2024), as well as scientific discovery and deep-research tasks, are natural settings for evaluating whether HERMES remains effective when the reasoning horizon substantially exceeds a single context window.

HERMES currently relies on a hand-designed strategy space and only explicitly learns part of contextual reasoning. The strategy menu, the search–verification distinction, and the L3–L1 hierarchy are manually specified. Future work could instead learn new contextual reasoning strategies, compose them dynamically, or adapt the level of model control during inference. Moreover, contextual reasoning requires learning *what to delegate*, *what to preserve* during digestion, and *how to reason* from the resulting digest. We focus on learning delegation and rely on the base model's zero-shot ability to summarize and resume reasoning. Prior work suggests that reasoning from compressed histories can be learned (Wu et al., 2026a;b), while directly learning useful digests introduces a harder credit-assignment problem (Wu et al., 2026a). Jointly learning delegation and digestion is therefore an important next step, particularly as reasoning horizons grow.

Training optimizes only the root-agent trajectory. Subagent trajectories do not independently contribute to the SFT or RL objective. This avoids the substantially larger sample complexity and credit-assignment problem introduced by optimizing policy-generated subtasks, but also prevents subagents from adapting to the distribution of tasks produced by the learned delegator. Recursive Agent Optimization (RAO) (Gandhi et al., 2026), for example, jointly optimizes nodes in a recursive execution tree using node-local rewards and trajectory weighting. Extending HERMES-LEARN to jointly optimize root and subagent behavior is a promising direction, but may require reliable process-level supervision or learned judges.

Finally, HERMES trades additional inference computation for improved reasoning. Future work should study contextual reasoning jointly in terms of accuracy, token usage, latency, and cost, and learn when allocating another context is worth the additional compute. More broadly, our current

experiments shift control over a fixed contextual-reasoning interface from the harness to the model; a longer-term goal is to let models learn the interface itself.

B Additional Discussion of Related Work

We provide more detailed discussions of the training approaches used by methods that train models for particular inference harnesses in Tab. 1. `SPIRAL` introduces set-level credit assignment to train parallel generation and aggregation. `PaCoRe` uses outcome-based reinforcement learning to train reasoning synthesis across parallel branches. `RAO` assigns node-local credit for recursive delegation decisions, enabling models to learn when and how to spawn additional subagents. `Context-Folding` employs process rewards to train delegation of subtasks to fresh branches together with compaction of intermediate reasoning. `Reasoning Cache` uses short-horizon reinforcement learning over repeated generate–summarize cycles to train continuation from compacted reasoning progress. While these methods employ different supervision and credit-assignment mechanisms tailored to their inference procedures, they generally assume prescribed contextual reasoning strategies encoded in the harness and train models to execute those strategies effectively.

C Hermes Harness Details

For all models, we impose an artificial context limit of 8K tokens, creating a setting where reasoning often extends substantially beyond a single window and effective context management becomes necessary for test-time scaling. Importantly, this setting is not purely academic: bounded context enables reasoning to be distributed across multiple windows, allowing independent branches to execute in parallel rather than requiring an equivalently long sequential trajectory. This creates a practical path for shorter-context models to scale inference while potentially reducing wall-clock latency. In our experiments, we evaluate the Hermes L1 harness, in which the model has the greatest control over its contextual reasoning decisions. All evaluations use a single root-agent context window (digesting is disabled), with $T = 3$ steps, delegation depth $D = 1$ (subagents cannot delegate), and up to $B = 3$ subagents per delegation step. The root agent and all subagents share the same underlying LLM and the same 8K-token context limit.

C.1 Harnesses in the Hermes Hierarchy

For Hermes harnesses with $T = 3$, each context window consists of two delegation steps followed by a return step, during which the agent synthesizes the accumulated conversation and subagent results without further delegation. Delegation is organized into two broad categories: *search* and *verification*.

The steering level determines how these categories and their underlying strategies are exposed to the model. Under L1, all available search and verification strategies are accessible at both delegation steps, resulting in a single harness configuration. Under L2, the harness specifies the delegation category at each step, while the model selects among the strategies within that category. Since verification requires an existing candidate solution, the first delegation step is restricted to *search*. The second step may use either *search* or *verification*, yielding two L2 variants: L2-SV (Search→Verify→Return) and L2-SS (Search→Search→Return).

For *search*, we consider three ways of exploring the solution space. `Same` asks subagents to solve the full problem independently, without prescribing an approach; `Different` instead assigns distinct approaches to encourage diverse solutions; and `Split` decomposes the problem into subproblems that are solved independently.

For *verification*, we consider three complementary forms of checking. `Answer Verification` targets the current candidate answer directly; `Intermediate Step Verification` examines the assumptions and reasoning that support it; and `Disagreement Resolution` reconciles conflicting candidates by identifying the source of disagreement.

What changes across the hierarchy. The three L1 and L2 harnesses use the same system prompt (Prompt 1), final-step return prompt (Prompt 7), and subagent inline reasoning and report prompts

(Prompts 8 and 9). They differ only in their steering level, which determines what contextual reasoning strategies are available at each delegation step.

Table 6: The harnesses in the HERMES hierarchy. Each cell gives the number of strategies offered at that step; the instruction text is otherwise identical. Delegation is disabled at the final step.

Harness	Strategy menu offered at a delegating step	Step 1	Step 2
L3	One named strategy	1	1
L2-SS	The three *search* strategies	3	3
L2-SV	The three *search* strategies at step 1, and the three *verification* strategies at step 2	3	3
L1	All six strategies, at every delegating step	6	6

C.2 PROMPTS

C.2.1 SYSTEM PROMPT

Prompt 1: System prompt — identical at L3, L2 and L1

```
You are a meta-agent for solving math problems.  You solve by DELEGATING to subagents, not
by deriving solutions inline.  You proceed in steps and have two actions:
1.  Delegate:  write one or more lines of the form "TASK: <self-contained task>" (1--3 per
step).  Each task is executed by a subagent that sees ONLY that task text and replies with
a short report; subagents cannot see the problem, your conversation, or each other, so
every task must contain everything needed to execute it.  The reports arrive in your next
step.
2.  Return:  when you are done, write "### RETURN" on its own line, followed by your final
answer.  Put the answer in \boxed{}.  This ends the episode.
You may compare, select, or briefly combine conclusions explicitly stated by subagents, but
do not derive new solutions yourself.  Use subagent results critically; agreement alone
does not establish correctness.
Always provide the best provisional answer.  Put it in \boxed{}.  Exception:  do not guess
before delegating.  On the first step you have no subagent results, so delegate without
stating an answer; only from the step after results arrive should you end each step with
your current best answer.
Delegation serves two purposes.  Search:  make progress on the problem (decompose it into
subproblems, or have subagents solve the full problem independently using either different
named approaches or no prescribed approach).  Verify:  check the current answer (recheck it
with targeted checks, verify the assumptions and intermediate steps supporting it, or
reconcile disagreeing candidates).  Each step's instruction tells you which specific
strategies are available.
```

C.2.2 L1: FULL STRATEGY MENU AT EVERY STEP

Prompt 2: L1, step 1 — delegate (full strategy menu)

```
### PROBLEM
{problem}
### INSTRUCTIONS
Delegate the work for this step; do not solve it inline.  You have no subagent results yet,
so do not state a boxed answer on this step.
Write one concise first-person sentence explaining why the selected tasks are useful.
Generate 1--3 tasks using one or more strategies from ### STRATEGIES. Each task must be
independently executable and fully self-contained.  Include the complete problem or all
relevant context, values, assumptions, and -- when the task checks a candidate answer --
that answer and the exact claim to verify.
### STRATEGIES
Ask 1--3 subagents to:
- solve independent subproblems obtained by decomposing the original problem; or
- solve the full problem independently using different named approaches; or
- solve the full problem independently using the same task and no prescribed approach; or
- verify the current boxed answer using independent targeted checks; or
- independently verify the assumptions and intermediate steps supporting the current boxed
answer; or
- resolve the disagreement between the candidate answers and identify the source of the
discrepancy.
### OUTPUT
<one concise first-person sentence>
### TASKS
Write one task per line.  Begin each line with "TASK:".
TASK: <self-contained task>
```

Prompt 3: L1, step 2 — delegate (full strategy menu, and current best answer)

```
### SUBAGENT RESULTS
{subagent_results}
### INSTRUCTIONS
Always provide the best provisional answer.  Put it in \boxed{}.
Use at most two concise lines to justify the current answer.
Write one concise first-person sentence explaining why the selected tasks are useful.
Generate 1--3 tasks using one or more strategies from ### STRATEGIES. Each task must be
independently executable and fully self-contained.  Include the complete problem or all
relevant context, values, assumptions, and -- when the task checks a candidate answer --
that answer and the exact claim to verify.  State candidate answers inside tasks in plain
text; never use \boxed{} inside a task.
### STRATEGIES
Ask 1--3 subagents to:
- solve independent subproblems obtained by decomposing the original problem; or
- solve the full problem independently using different named approaches; or
- solve the full problem independently using the same task and no prescribed approach; or
- verify the current boxed answer using independent targeted checks; or
- independently verify the assumptions and intermediate steps supporting the current boxed
answer; or
- resolve the disagreement between the candidate answers and identify the source of the
discrepancy.
### OUTPUT
### ANSWER
<at most two concise lines>
\boxed{<current best answer>}
<one concise first-person sentence>
### TASKS
Write one task per line.  Begin each line with "TASK:".
TASK: <self-contained task>
```

C.2.3 L2-SV AND L2-SS: ONE CATEGORY PER STEP

Both L2 variants begin with the same search-only step.

Prompt 4: L2-SV and L2-SS, step 1 — delegate (*search* category only)

```
### PROBLEM
{problem}
### INSTRUCTIONS
Delegate the work for this step; do not solve it inline.  You have no subagent results yet,
so do not state a boxed answer on this step.
Write one concise first-person sentence explaining why the selected tasks are useful.
Generate 1--3 tasks using one or more strategies from ### STRATEGIES. Each task must be
independently executable and fully self-contained.  Include the complete problem or all
relevant context, values, assumptions, and requested output.
### STRATEGIES
Ask 1--3 subagents to:
- solve independent subproblems obtained by decomposing the original problem; or
- solve the full problem independently using different named approaches; or
- solve the full problem independently using the same task and no prescribed approach.
### OUTPUT
<one concise first-person sentence>
### TASKS
Write one task per line.  Begin each line with "TASK:".
TASK: <self-contained task>
```

They then diverge. L2-SV switches the menu to the verification category, which assumes that a first step of search has produced a response worth checking:

Prompt 5: L2-SV, step 2 — delegate (*verification* category only)

```
### SUBAGENT RESULTS
{subagent_results}
### INSTRUCTIONS
Always provide the best provisional answer.  Put it in \boxed{}.
Use at most two concise lines to justify the current answer.
Write one concise first-person sentence explaining why the selected tasks are useful.
Generate 1--3 tasks using one or more strategies from ### STRATEGIES. Each task must be
independently executable and fully self-contained.  Include the relevant problem context,
values, assumptions, candidate answer, and exact claim to solve or verify.  State candidate
answers inside tasks in plain text; never use \boxed{} inside a task.
### STRATEGIES
Ask 1--3 subagents to:
```

```
- verify the current boxed answer using independent targeted checks; or
- independently verify the assumptions and intermediate steps supporting the current boxed
answer; or
- resolve the disagreement between the candidate answers and identify the source of the
discrepancy.
### OUTPUT
### ANSWER
<at most two concise lines>
\boxed{<current best answer>}
<one concise first-person sentence>
### TASKS
Write one task per line.  Begin each line with "TASK:".
TASK: <self-contained task>
```

L2-SS instead offers the search category a second time, which assumes further exploration is more valuable than checking:

Prompt 6: L2-SS, step 2 — delegate (*search* category again)

```
### SUBAGENT RESULTS
{subagent_results}
### INSTRUCTIONS
Always provide the best provisional answer.  Put it in \boxed{}.
Use at most two concise lines to justify the current answer.
Write one concise first-person sentence explaining why the selected tasks are useful.
Generate 1--3 tasks using one or more strategies from ### STRATEGIES. Each task must be
independently executable and fully self-contained.  Include the complete problem or all
relevant context, values, assumptions, and requested output.  State candidate answers
inside tasks in plain text; never use \boxed{} inside a task.
### STRATEGIES
Ask 1--3 subagents to:
- solve independent subproblems obtained by decomposing the original problem; or
- solve the full problem independently using different named approaches; or
- solve the full problem independently using the same task and no prescribed approach.
### OUTPUT
### ANSWER
<at most two concise lines>
\boxed{<current best answer>}
<one concise first-person sentence>
### TASKS
Write one task per line.  Begin each line with "TASK:".
TASK: <self-contained task>
```

C.2.4 RETURN STEP

Prompt 7: Return step — the final step with no further delegation. Identical across L1 and L2.

```
### SUBAGENT RESULTS
{subagent_results}
### INSTRUCTIONS
Update the current best answer using the conversation and subagent results.  Compare,
select, or briefly combine findings stated by subagents.
You may perform the simple calculations needed to combine the subagent results into the
final answer.
Do not delegate additional tasks.
Begin your reply with "### RETURN" on its own line (write the marker literally, even when
you are confident), then at most two concise lines and your final answer.  Put the answer
in \boxed{}.
### OUTPUT
### RETURN
<at most two concise lines>
\boxed{<final answer>}
```

C.2.5 SUBAGENT PROMPTS

At the maximum depth, subagents cannot delegate further and instead reason inline. They follow a simple two-step procedure: first solve the assigned task with inline reasoning, then produce a compact report for the delegating agent. A subagent never sees the original problem or the delegating agent's interaction history, only the task assigned by the delegating agent. The delegating agent has access only to the subagent reports, allowing its working context to remain lean and support multiple rounds of reasoning and delegation within a fixed context limit. The delegating agent then updates

its own reasoning based on the received reports and determines subsequent reasoning and delegation decisions.

Prompt 8: Subagent, step 1 — solve the assigned task with inline reasoning. Identical across all levels.

```
### TASK
{task}
### INSTRUCTIONS
Solve the task carefully, reasoning step by step.  Use the information provided in the
task.  If essential information is missing, state what is missing instead of guessing.
Check the result before reporting it.
```

Prompt 9: Subagent, step 2 — return a compact report to the delegating agent. Identical across all levels.

```
Report the result of the task you just solved.  Keep the report self-contained and concise.
Include only the decisive findings, not the full derivation.
### OUTPUT
### RESULT
Scope:  <briefly identify what was solved or checked, including essential context>
Finding:  <one or two concise sentences with the conclusion and decisive justification>
Answer:  \boxed{<answer>}
```

C.2.6 DIGESTION ACROSS CONTEXT WINDOWS

When reasoning extends beyond a single context window, the HERMES harnesses do not carry the full interaction history forward. Instead, it prompts the model to generate a digest using the templates below and initializes the next context window with the original problem and the generated digest. When generating a new digest, the model has access to both the previous digest and the complete root-agent trajectory from the current context window. In this way, digests compose across an arbitrary number of context windows.

Prompt 10: Digest system prompt

```
You summarize problem-solving work so that a fresh solver can continue from your summary
alone.  Write at most two paragraphs, in the first person, as the solver.
Cover:  (1) the approaches attempted and their status (completed, incomplete and where they
stopped, or failed and why); (2) key intermediate results with their verification status
(derived equations, lemmas, computed quantities, checks that passed or failed); (3) the
current best answer, confidence, and what remains to be verified.
Hard constraints:  retain any important information contained in the existing summary;
summarize only --- do not add new reasoning or new derivations.  These constraints take
priority over length:  if retaining prior information needs more space, exceed two
paragraphs rather than drop information, but compress aggressively (merge duplicate
findings).
```

Prompt 11: Digest user prompt

```
### PROBLEM
{problem}
### PREVIOUS SUMMARY
{previous_digest}
### EPISODE TRANSCRIPT
{transcript}
Write the updated summary now.
```

The first step of the next context window uses the standard Step-1 prompt with an additional `PRIOR WORK SUMMARY` block containing the digest:

Prompt 12: L1, step 1 of a *continued* window with a digest under `PRIOR WORK SUMMARY`

```
### PROBLEM
{problem}
### PRIOR WORK SUMMARY
{previous_attempts}
If a summary of prior work is shown above, you may build on it, verify it, or start fresh;
otherwise start from the problem.
### INSTRUCTIONS
```

```
Delegate the work for this step; do not solve it inline.  You have no subagent results yet,
so do not state a boxed answer on this step.
Write one concise first-person sentence explaining why the selected tasks are useful.
Generate 1--3 tasks using one or more strategies from ### STRATEGIES. Each task must be
independently executable and fully self-contained.  Include the complete problem or all
relevant context, values, assumptions, and -- when the task checks a candidate answer --
that answer and the exact claim to verify.
### STRATEGIES
Ask 1--3 subagents to:
- solve independent subproblems obtained by decomposing the original problem; or
- solve the full problem independently using different named approaches; or
- solve the full problem independently using the same task and no prescribed approach; or
- verify the current boxed answer using independent targeted checks; or
- independently verify the assumptions and intermediate steps supporting the current boxed
answer; or
- resolve the disagreement between the candidate answers and identify the source of the
discrepancy.
### OUTPUT
<one concise first-person sentence>
### TASKS
Write one task per line.  Begin each line with "TASK:".
TASK: <self-contained task>
```

D Data Augmentation Details

To expose the model to a broader and balanced range of contextual reasoning behaviors during training, we generate synthetic problems designed to favor different search and verification strategies, including five types: (1) *Splitting*, which combines multiple independent problems into a single instance and encourages splitting and parallel delegation; (2) *Chaining*, where solving a gating problem determines which of several subsequent problems should be solved, encouraging sequential delegation; (3) *Verifying*, where intermediate results may be incorrect, encouraging delegation for verification; (4) *Multi-attempt*, which paraphrases medium-difficulty problems where multiple independent attempts can improve performance; and (5) *Single-attempt*, which paraphrases low-difficulty problems where a single attempt is sufficient. Together, these controlled structures encourage the model to adapt its delegation strategy to the problem.

D.1 Synthetic Problem Generation Pipeline

Each synthetic problem is generated in two stages. First, one or more seed problems are composed into a target synthetic problem type. The final answer is computed deterministically from the verified reference answers of the seed problems using simple arithmetic rules specific to each type: (1) *Splitting.* Given part answers a_i, the final answer is $\sum_i a_i \bmod 1000$; (2) *Chaining.* Let m denote the answer to the gating problem and a_i the answer to the i-th candidate problem. The final answer is $(m + a_{1+(m \bmod 3)}) \bmod 1000$. (3) *Verifying.* Given a claimed value c and the true answer a, the final answer is $(c + a) \bmod 1000$. For (4) *Multi-attempt* and (5) *Single-attempt*, the answer to the seed problem is kept unchanged.

Second, the composed problem is rewritten by Claude `Sonnet 5` to smooth its writing style and remove potential artifacts. Rewriting uses the system prompt in Prompt 13 and user message in Prompt 14, where the user message includes a style specification randomly sampled from four candidate writing styles: *narrative*, *compact*, *formal*, and *inverted*. For *Chaining* problems, a note for composite style (Prompt 15) is additionally included in its style specification. To further control the quality of the synthetic problems, a rewritten problem is retained only if it remains solvable: Claude `Sonnet 5` attempts each problem 5 times before and 5 times after rewriting, and the rewrite is rejected if none of the post-rewrite attempts is correct or if the post-rewrite solve rate falls to $1/3$ of the pre-rewrite solve rate or below. This excludes rewrites that make a problem substantially less solvable, indicating that the rewriting has potentially altered the problem.

Prompt 13: Rewrite system prompt

```
You rewrite mathematics problems.  You will be given a problem, and you must produce a
rewritten version that asks for EXACTLY the same final quantity.

Absolute requirements:
- Preserve every number, constraint, condition and definition.  Change no value.
```

```
- Preserve exactly what the final answer must be.  If the original asks for a remainder
modulo 1000, so must yours.  If it asks for a sum of several sub-answers with specific
coefficients, so must yours, with the same coefficients.
- If the original contains several sub-problems whose answers are combined, keep all of
them and keep the combination rule identical.
- Preserve any quoted claim, reported answer, or summary of prior work, including its
stated numeric value.  You may reword it; you may not change the number, and you may not
resolve whether it is right.
- Keep all LaTeX mathematically identical.  You may reformat it.

Style requirements:
- Do NOT signpost how the problem should be approached.  Never write "solve each part
separately", "note that these are independent", "first solve X then", or anything that
names a solution strategy.
- Do NOT use the labels "Problem 1", "Problem 2", "[Problem 2-A]", "A_1", "A_2", "Q" or
similar scaffolding.  Refer to the quantities in words instead.
- Vary sentence structure.  Avoid a fixed template.

You do not know the answer and must not attempt to compute it.  Do not solve.  Do not hint.

Return ONLY the rewritten problem text.  No preamble, no commentary, no answer.

STYLE RULES. These come from editors who could reliably tell machine-written problems from
published ones.  Follow every one.

Quantity names.
- A quantity gets a single italic letter or no name at all.  Never invent a prose name:
not "the size sum", not "the pentagon count", not "the divisor-count remainder", not "the
selected quantity", not "the resulting quantity".  If the source already names quantities
with letters, keep those letters.
- Never reuse one symbol for two things in the same problem.

Structure of the writing.
- Setup first, question last.  Never open with the ask and defer the setup ("..., where
that value is produced as follows", "..., obtained as follows").
- State the ask exactly once.  If your draft opens and closes with the same request, delete
the opening.
- One interrogative sentence, and it is the last sentence.
- Do not split an ask in two.  Write "Compute the sum of all positive integers n such that
..." rather than "Determine all n ..., then find the sum".
- Do not fold the setup into a participial clause ("With p the smallest prime for which
..., find ..."). Use two sentences.

Voice.
- Third person, present indicative.  No first person anywhere:  no "I", "we", "I'd like to
know", "I'm curious about", "we're told", "Here's", "Picture a", "Now look at".
- Keep the source's named actors and concrete story.  Do not turn "Karthik wants to walk a
path" into "A closed path is to be traced" -- passive de-naming is detectable.
- Never narrate a solution method in any voice.

Words and constructions to avoid outright.
- "give rise to", "arise as follows", "obtained as follows", "produced as follows", "comes
into play", "the quantity in question", "one of the two quantities needed", "a certain
resulting value", "it turns out that", "it happens that", "so-called", "for context",
"roughly speaking", "possess" (write "has"), "utilize".
- No markdown of any kind:  no headings, no bold, no italic asides.
- No em-dash appositive carrying a constraint, and no colon-splice that dumps a
configuration and then the question.
- Never write "an isosceles triangle $\triangle ABC$" -- either "triangle $ABC$" or
"$\triangle ABC$".

Technical fidelity.
- Copy technical hypotheses verbatim.  Never paraphrase a gcd or coprimality condition:
write "$\gcd(m,n)=1$" or "where $m$ and $n$ are relatively prime positive integers" exactly
as the source has it.  A paraphrase like "$\gcd(m,p,q)$ shares no common prime factor" is
wrong and is a giveaway.
- Every numeral that was in math mode stays in math mode.  Do not unwrap "$13$" to "13".
- Keep the source's macros, arrays, bulleted condition lists and worked examples in place.
- Introduce a definition in its own sentence before first use, and give at most one
example, opened with "For example" or "For instance".
- Stay within about 15% of the source's length.  Telegraphic compression is as detectable
as padding.

Provenance tags.
- Delete any citation of where the problem came from:  "(26th IMO Shortlist)", "Example 1",
"Problem 4", "2019 AMC 12B", "from the CMO".  Scraped seeds carry these; a contest booklet
does not print them inside the statement, and they identify the source.

Preferred verbs:  Let, Suppose, Consider, Compute, Find, How many.
```

Prompt 14: Rewrite user message, with one of four candidate style specifications

```
{style brief}{composite note, Chaining problems only}

Here is the problem to rewrite:

<problem>
{generated problem text}
</problem>
```

Notes: *The {style brief} is one of the following, sampled per problem.*
narrative: Rewrite it as a short, natural piece of mathematical prose, as a person would pose it to a colleague. Use connected sentences rather than labelled blocks.
compact: Rewrite it as tersely as possible while keeping every constraint. Prefer one flowing paragraph over enumerated parts.
formal: Rewrite it in the register of a textbook exercise: precise, impersonal, no framing device, no labels beyond what is strictly needed.
inverted: Rewrite it so the final quantity being asked for is stated FIRST, then the information needed to obtain it.

Prompt 15: Composite style note, appended to the style specification for Chaining problems

```
This problem contains several separate questions whose answers are then combined. Handle
that as follows:

- Keep every question, with all of its numbers and conditions.
- Keep the single-letter names the problem already uses for the combined quantities, and
keep the final combination sentence as it stands.
- State the final combination using those names, with the SAME coefficients, the SAME
operations and the SAME modulus as the original.
- Present the questions as a continuous piece of writing, not as a list of labelled blocks.
- Do not say that the questions are independent, unrelated, or separable, and do not
suggest any order or division of work.
```

D.2 Synthetic Problem Construction

Synthetic problems for training. Synthetic training problems are constructed from AceReason-1K. We first remove seed problems with tuple- or list-valued answers or malformed LaTeX, leaving 903 seed problems. Among them, 682 have non-negative integer answers and can be used to construct Splitting, Chaining, and Verifying problems.

Using these seed problems, we generate 1,141 synthetic problems, of which 708 pass the solvability check after rewriting. To broaden coverage of both contextual reasoning strategies and distinct seed problems, we select a final set of 500 synthetic problems covering 505 distinct seeds, with a fixed number of problems per type. Composite structures (Splitting and Chaining) account for 60% of the 500 problems. Verifying problems are included in matched correct/incorrect pairs.

The final dataset contains Splitting ($n = 214$; 107 two-part and 107 three-part problems), Chaining ($n = 86$), Verifying ($n = 100$; 50 correct and 50 incorrect claims paired on the same 50 seed problems), Multi-attempt ($n = 50$), and Single-attempt ($n = 50$). Multi-attempt and Single-attempt problems are constructed from disjoint seed sets. After mixing with AceReason-5K, the RL training set contains 5,739 problems in total, of which 500 (8.7%) are synthetic.

Synthetic evaluation benchmark. The synthetic benchmark is built from AIME and HMMT problems using the same generation, rewriting, and solvability-check pipeline. It contains $n = 88$ problems: Splitting ($n = 28$: 14 with two parts and 14 with three parts), Chaining ($n = 14$), Verifying ($n = 18$: 9 with a correct claim and 9 with an incorrect claim, paired on the same 9 seed problems), Multi-attempt ($n = 14$), and Single-attempt ($n = 14$). Every golden answer is recomputed independently from the reference answers of the seed problems.

Fig. 4 shows one example synthetic training problem (constructed from AceReason-1K) and one example synthetic evaluation problem (constructed from AIME or HMMT) for each problem type.

Training (AceReason-1K)	Evaluation (AIME / HMMT)
Splitting	
Let **m** be the count of four-digit numbers whose digits sum to 12, let **p** be the single digit (0–9) giving the position, after 1234 moves, of a counter that starts on a 10-cycle and on its nth move advances n^n steps clockwise, and let **q** be the largest number of diagonals selectable in a regular 1000-gon (with all diagonals drawn) so that among any three selected diagonals at least two share the same length. Find the remainder when **m** + **p** + **q** is divided by 1000. **Golden answer:** 349 (**m**, **p**, **q**) = (342, 7, 2000)	Let **p** denote the value of xy, where x and y are real numbers, both greater than 1, satisfying $\log_x(y^x) = \log_y(x^{4y}) = 10$. Separately, Jen enters a lottery by picking 4 distinct numbers from $S = \{1, 2, 3, \cdots, 9, 10\}$. Then 4 numbers are randomly chosen from S. She wins a prize if at least two of her numbers are among the 4 randomly chosen numbers, and she wins the grand prize if all four of her numbers are the randomly chosen numbers. The probability that she wins the grand prize given that she won a prize equals $\frac{m}{n}$, where m and n are relatively prime positive integers; let **q** denote $m + n$. Compute the remainder when **p** + **q** is divided by 1000. **Golden answer:** 141 (**p**, **q**) = (25, 116)
Chaining	
Compute the remainder when **m** + **p** is divided by 1000, where **m** is the answer to the following question: how many four-digit numbers have the sum of their digits equal to 12? Three further questions are considered. A counter moves in a cycle of 10 positions, and on its nth move it advances n^n steps clockwise; the question asks for the counter's position after 1234 moves, expressed as a single digit from 0 to 9. In a regular 1000-gon with all of its diagonals drawn, the question asks for the maximum number of diagonals that can be selected so that among any three of the chosen diagonals at least two have the same length. For positive integers a, b, c, the question asks for the smallest positive value taken by $a^3 + b^3 + c^3 - 3abc$, together with all triples a, b, c that attain this smallest value. Let **p** be the answer to the question numbered $1 + (\mathbf{m} \bmod 3)$ among these three, in the order in which they are given here. **Golden answer:** 349 **m** = 342, question 1, **p** = 7	Let **m** be the answer to the following question: find the sum of all positive integers n such that $n + 2$ divides the product $3(n + 3)(n^2 + 9)$. Alongside that, consider three further questions, numbered in order. The first: Mark writes the expression $\sqrt{abcd}$ on the board […] The second: every morning Aya goes for a 9-kilometer walk […] The third: find the number of ordered pairs (x, y) with both x and y integers between -100 and 100, inclusive, satisfying $12x^2 - xy - 6y^2 = 0$. Let **p** be the answer to whichever of these three questions carries the number $1+(\mathbf{m} \bmod 3)$. What is the remainder when **m**+**p** is divided by 1000? **Golden answer:** 253 **m** = 49, question 2, **p** = 204
Verifying	
Consider the equation $\sin x = \frac{x}{100}$. Rowan computes the number of roots of this equation and reports the value **63**. Find the remainder when the sum of Rowan's reported value and **the actual number of roots** is divided by 1000. **Golden answer:** 126 claim **63** (correct), true value **63**	Let S denote the vertex set of a regular 24-gon. Consider ways of drawing 12 segments, all of the same length, such that every vertex in S serves as an endpoint of exactly one segment; call the number of such ways **N**. Priya computes this count herself and gets **4208**. Find the remainder when **N** plus Priya's value of **4208** is divided by 1000. **Golden answer:** 321 claim **4208** (incorrect), **N** = 113
Multi-attempt	
Let x, y, z be positive real numbers, and set $s = \sqrt{x+2} + \sqrt{y+5} + \sqrt{z+10}$ and $t = \sqrt{x+1} + \sqrt{y+1} + \sqrt{z+1}$. Find the minimum value of $s^2 - t^2$. **Golden answer:** 36	Find the largest number that is less than BD^2 for every rhombus $ABCD$ satisfying the following conditions. Its four vertices lie, in that order, on the hyperbola $\frac{x^2}{20} - \frac{y^2}{24} = 1$, and its diagonals intersect at the origin. **Golden answer:** 480
Single-attempt	
Find the smallest positive integer whose digits are all 1's that is divisible by the number consisting of 100 copies of the digit 3. **Golden answer:** 300	Compute the number of positive integers $n \leq 1000$ for which $\mathrm{lcm}(n, 9)$ is a perfect square. **Golden answer:** 43

Figure 4: *Synthetic problem examples.* Colored letters identify intermediate values and subanswers that determine the final answer. […] denotes problem text omitted for brevity.

E EXPERIMENT DETAILS

E.1 POOLED BENCHMARK DETAILS

We evaluate on a pooled benchmark of 238 mathematical reasoning problems constructed from AIME 2024, AIME 2025, AIME 2026, HMMT February 2023, HMMT February 2024, HMMT February 2025, HMMT November 2025, and HMMT February 2026. The original collection contains 243 problems (30 from each contest except HMMT February 2026, which contains 33 problems). We use the HuggingFace `Math-Verify`[4] grader to evaluate model outputs against ground-truth answers.

For HMMT, we exclude five problems following auditing and decontamination: (1) HMMT February 2023 Problem 26, which duplicates a training-data problem; (2) HMMT February 2024 Problem 21, which requires a missing diagram; and (3) HMMT February 2026 Problems 31–33, which are proof-style problems whereas the `Math-Verify` grader assumes final-answer evaluation. The answers for the remaining 148 problems were then independently derived or exactly checked, and 19 problem transcription and answer corrections were applied to ensure the validity of the benchmark.

For AIME, we audit all 90 problems and correct a number of issues. In particular, two AIME 2026 problems did not match the official contest statements and were corrected accordingly. We further identify corrupted or incomplete problem statements and incorrect answers throughout the datasets. In total, 25 problems and 19 answers were corrected. The final pooled benchmark therefore consists of 90 AIME problems and 148 HMMT problems, for a total of 238 problems.

E.2 INFERENCE UNDER HERMES HARNESSES

Under HERMES harness, all agents and subagents share the same underlying LLM and the same 8K-token (8,192-token) context limit. Agents are limited to 1K (1,024) generated tokens per step. At the maximum delegation depth D, subagents follow a two-step solve-and-report procedure (App. C.2.5) and are allocated up to 7.5K (7,680) generated tokens for the first step where they solve delegated tasks inline. Digest generation (App. C.2.6) is performed under the same 8K-token context limit.

For all evaluations, we report average accuracy over 16 sampled solutions (Avg@16). We follow the suggested sampling parameters listed in the `Qwen3-4B-Instruct-2507`[5] and `Olmo-3-7B-Instruct`[6] model cards when evaluating their corresponding base and post-trained models, under inline reasoning as well as the HERMES, RSA, and DSM harnesses. The sampling parameters are listed in Tab. 7.

Table 7: Sampling parameters.

	`Qwen3-4B-Instruct-2507`	`Olmo-3-7B-Instruct`
Temperature	0.7	0.6
Top-p	0.8	0.95
Top-k	20	–
Min-p	0.0	–

E.3 TRAINING EXPERIMENTS DETAILS

Training Dataset. Starting from the 5.7K-problem subset of AceReason-Math (Chen et al., 2025) released by Reasoning Cache (Wu et al., 2026a), we perform deduplication, decontamination, and reference-answer verification, resulting in a cleaned set of 5,239 problems, which we refer to as ACEREASON-5K. We then uniformly sample 1,000 problems from ACEREASON-5K to construct ACEREASON-1K.

[4] `https://github.com/huggingface/Math-Verify`

[5] `https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507`

[6] `https://huggingface.co/allenai/Olmo-3-7B-Instruct`

Stage I: Supervised Fine-tuning. For SFT, we curate teacher trajectories from both Claude `Sonnet 5` and `DeepSeek-V3.2`, under L1 and L2 HERMES harnesses. For each teacher model, we collect four L1 (App. C.2.2), two L2-SV, and two L2-SS (App. C.2.3) trajectories for every problem in ACEREASON-1K, resulting in 8,000 trajectories per teacher model. We then retain only trajectories that contain at least one delegation and reach the correct final answer for SFT, resulting in 6,485 trajectories from `Sonnet 5` and 5,495 trajectories from `DeepSeek-V3.2`.

Table 8: Percentage (%) of teacher trajectories containing different strategies. Percentages may sum to more than 100% since a single trajectory can contain multiple strategies.

Teacher	Split	Same	Different	Verification	Multiple Search Strategies
`Sonnet 5`	21.9	43.9	63.5	99.3	27.6
`DeepSeek-V3.2`	26.0	70.4	26.6	95.1	22.6

Teacher trajectories provide diverse delegation demonstrations. We use an LLM judge (Section E.4) to label the contextual reasoning strategies represented by tasks delegated to subagents in successful teacher trajectories (Table 8). Both teachers exhibit strong verification behavior and cover a diverse set of search strategies, while exhibiting complementary preferences: `Sonnet 5` more frequently explores different approaches, whereas `DeepSeek-V3.2` more often performs independent solution attempts. Collecting trajectories from multiple teachers therefore broadens the coverage of effective delegation demonstrations.

Different harnesses further increase delegation diversity. While most teacher trajectories begin with a search task, second-step behavior varies across harnesses due to differences in steering (Table 9). L2-SS encourages continued exploration via search, whereas L2-SV encourages verification and therefore exhibits the lowest search frequency. L1 lies between these two, where the model has the highest control over its delegation decisions. Collecting teacher trajectories from both L1 and L2 therefore exposes the base model to a broader range of effective delegation patterns.

Table 9: Percentage (%) of teacher trajectories containing a search strategy in each step.

Teacher	HERMES Harness	Step 1 Search	Step 2 Search
`Sonnet 5`	L1	99.9	19.0
`Sonnet 5`	L2-SV	100.0	15.5
`Sonnet 5`	L2-SS	99.9	29.2
`DeepSeek-V3.2`	L1	99.5	35.4
`DeepSeek-V3.2`	L2-SV	99.6	28.9
`DeepSeek-V3.2`	L2-SS	99.9	53.2

To best leverage this diversity while preventing any single teacher or delegation pattern from dominating the training data, we subsample each teacher's retained trajectories to balance the occurrence of search strategies before combining them into a final SFT dataset of 5,466 trajectories (2,733 from each teacher). This exposes the base model to a diverse and balanced set of delegation behaviors.

Stage II: Reinforcement learning. For the RL stage, we train on ACEREASON-5K using a curriculum over the HERMES hierarchy. Training runs for 492 steps (3 epochs), and the curriculum is defined as: over the first third of training (steps 1–164), the two L2 harnesses, L2-SV and L2-SS, are each sampled with 50% probability, and over the remainder (steps 165–492) the L1 harness is sampled with 75% probability while the two L2 harnesses are each sampled with 12.5% probability. This curriculum gradually shifts training from L2 to L1 while maintaining equal overall exposure to both levels of harnesses throughout training. During RL training, both the root agent and all subagents use the current policy being updated.

As described in App. D.2, we train RL on a mixed dataset including ACEREASON-5K and 500 synthetic problems constructed from ACEREASON-1K, comprising 5,739 problems in total. All training hyperparameters are provided in Tab. 10.

E.4 LABELING EXECUTED CONTEXTUAL REASONING STRATEGIES

For qualitative analysis of contextual reasoning behaviors, we set up an LLM judge with Claude `Sonnet-5` to label the strategies executed in agent trajectories. We take several measures to prevent information leakage during labeling:

Table 10: Training hyperparameters.

Supervised Fine-tuning	
Learning rate	10^{-5}
Learning schedule	Cosine with warmup ratio 0.1
Training batch size	32
Epochs	4
Reinforcement Learning	
Algorithm	Dr. GRPO (Liu et al., 2025)
Group size G	8
Dynamic sampling	Resample while a group is all-correct or all-wrong
Resampling budget	≤ 10 generation batches per optimizer step
Clipping $(\epsilon_{\text{low}}, \epsilon_{\text{high}})$	$(0.2,\ 0.28)$
KL loss coefficient	0.001
KL estimator	k_3 (low-variance)
Entropy loss coefficient	0.0
Optimizer	AdamW, $\beta = (0.9,\ 0.999)$
Weight decay	0.01
Learning rate	10^{-6}
Learning schedule	Constant, no warm-up
Gradient clipping	1.0
Training batch size	32
Mini-batch size	16
Epochs	3
Inference temperature	1.0
Inference top-p	1.0

1. The judge sees only the problem statement, the texts generated by the delegating agent, and the results returned by subagents from previous delegation steps. It never observes the harness identity, model identity, steering instructions, available strategy menu, or any subagent trajectories. Consequently, the judge must infer strategy labels from the tasks generated by the delegating agent, rather than relying on strategy names contained in the steering instructions or on the outcomes of the delegated tasks being labeled.
2. Since task ordering carries no semantic meaning, delegation tasks are sorted according to their text before labeling. This prevents the judge from relying on presentation order when labeling strategies.
3. Residual leakage remains possible. In particular, the number of delegated tasks may reveal information about the underlying harness. To mitigate this effect, all reported comparisons using judge labels are performed between models evaluated under harnesses with the same maximum number of subagents, preventing the judge from relying on such differences.

The judge returns a set of strategy labels for each delegation step, as a single step may contain tasks corresponding to multiple strategies. Because Claude `Sonnet-5` does not accept a temperature parameter during inference calls, the judge cannot be forced to use deterministic decoding. We therefore measure consistency between two independent judge passes on a small subset of trajectories and obtain 79.4% agreement. The labeling rubric is specified in the system prompt (Prompt 16), and the judge is required to call the `label_trajectory` tool (schema shown in Prompt 17) to produce its labels.

Prompt 16: Judge system prompt

```
You are reading the instructions a manager sent to its assistants across the steps of ONE
problem.
You see ONLY the problem, the instruction texts, and (for later steps) the results that
came back.
You do NOT know which model wrote them or what it claimed to be doing.
For EACH step, report every category that the instructions on that step actually execute.
Judge the WORK REQUESTED, not the wording, and not whether it succeeded.

## Categories
**Checking work that already exists:**
```

```
- `verify` --- an instruction checks, re-derives, confirms, reconciles, or attacks a result
that ALREADY EXISTS in the prior results shown to you, or is quoted verbatim inside the
instruction.
Work an instruction performs itself is not a prior result.
When you use `verify`, add sub-tags:  `answer` (rechecks the stated final answer), `steps`
(checks a named intermediate step, equation, or assumption), `disagree` (two or more stated
candidates conflict and the instructions decide between them), `counter` (tries to break a
stated result or find a second solution).  **Sub-tags are descriptive only --- they are
never a reason to prefer or reject `verify`.**
**Solving from scratch:**
- `split_subproblems` --- the instructions ask for different pieces, and the pieces must be
combined to answer the question.  Requires ≥2 instructions.
- `same_solve` --- the instructions all ask for the whole answer and NO instruction names a
solution technique.  Rewordings of "solve this problem" are `same_solve`, however
differently phrased.  Requires ≥2 instructions.
- `different_methods` --- every instruction names at least one solution technique, and the
named sets are pairwise different.  A technique counts only when it appears in a naming
position ("using X", "via X", "apply X", "by X").  Different wording, different guesses
about an ambiguous problem, or different algebraic rearrangements of the same equation are
NOT different methods.  Requires ≥2 instructions.
- `one_fresh_attempt` --- a single instruction asks for one new attempt at the whole
problem.

## Multilabel
A step may execute more than one category --- report all that apply.  The common real case
is a step that mixes a check of an existing result with fresh solving:  label it `verify`
**and** the appropriate search category.  Do not add a label merely because it is arguably
present; add it when some instruction on that step plainly does that work.
Constraints, enforced downstream in code as well:
- `split_subproblems`, `same_solve`, `different_methods` each require ≥2 instructions on
that step.
- With exactly one instruction, the only admissible labels are `one_fresh_attempt` and
`verify`.
- `same_solve` and `different_methods` may BOTH apply to one step --- see the mixed-turn
rule below.
- Never return an empty label list for a step.  Every step has a best fit.

### Mixed fanout turns carry both labels
A turn where **some** instructions name a technique and others do not is genuinely both:
it fans out on the whole problem, and it assigns a method to part of that fanout.  Report
`[same_solve, different_methods]` rather than forcing a choice.
- Every instruction names a technique, sets differ pairwise → `different_methods` alone.
- No instruction names a technique → `same_solve` alone.
- Some do, some don't → **both**.
This is why the label set is multi-label:  the two are not opposite ends of one axis, and a
forced pick on a mixed turn is a coin flip that neither rater can make reliably (measured:
matching prevalences with per-label κ ≈ 0.16--0.29, i.e.  raters agreed on how often,
never on which turns).

## Two rules that decide most cases
1.  **Count the instructions.** Sub-parts inside one instruction are still one instruction.
2.  **`same_solve` vs `different_methods`:** list the techniques each instruction
explicitly names in a naming position.  If any instruction names none, the answer is
`same_solve`.
### Applying rule 2 --- do this literally, instruction by instruction
Measured:  judges over-call `different_methods` on any turn whose instructions merely
*read* differently.  Work the test mechanically instead of judging overall impression.
For each instruction, write down the technique it names in a naming position ("using X",
"via X", "apply X", "by X", "approach it by X").  Then:
- **Any instruction with an empty list → `same_solve`.** One instruction naming a
technique while another just says "solve this problem" is `same_solve`, not
`different_methods`.  The rule is about the weakest instruction, not the strongest.
- **`different_methods` requires every instruction to name a technique, and the named sets
to differ pairwise.** All naming the same technique is `same_solve`.
These are NOT techniques, even when the instructions differ visibly:
- a restatement of the problem's own given equations, constraints, or setup;
- a different algebraic rearrangement, substitution, or factoring of the same equation;
- a suggested first step or observation ("note that subtracting (1) from (2) gives ...");
- different guesses about an ambiguous problem;
- more or less detail, or different wording, in an otherwise identical request.
Worked examples, both `same_solve`:
- I1 "Solve this set theory problem **using a Venn diagram region approach**:  ..." / I2
"Solve this set theory problem:  ..." → I2 names none, so `same_solve`.  (I1's Venn
technique does not carry the turn.)
- I1 "...**Approach it by** treating (1) and (2) as showing b and c are roots of the same
quadratic" / I2 "...**Note that** subtracting (1) from (2) gives ..." → both are
rearrangements of the same system, not distinct methods.  `same_solve`.
```

```
`split_subproblems` is about the **requested output**, not the reasoning shown.  An
instruction that merely restates the problem's given setup, and asks for nothing but the
final answer, is not a subproblem.
But an instruction that asks for a **named intermediate quantity** --- "determine the radii
r1 and r2", "express the perimeters in terms of a and b", "compute the number of integers
k" --- is requesting a subproblem, **even when the same instruction also asks for the final
answer**.  Report `[split_subproblems, same_solve]` (or with `different_methods`, or with
`verify`) in that case.  Do not drop `split_subproblems` just because the final quantity is
also requested:  co-occurrence is the point of a multi-label space, and every label the
turn plainly executes should appear.
Only when NO instruction requests any named intermediate --- every one asks solely for the
final quantity --- is the turn pure fanout with no `split_subproblems`.

## Confidence
Per step, `HIGH` when confident, `LOW` when the texts are truncated, vague, or you had to
guess.
## Why verify is one category, not four
Two independent judges on 200 blinded items agreed 74.0% (Cohen's κ 0.644) over an 8-way
space, but **78.8% of all their disagreement was internal to the four verify labels and
none of it was search-versus-verify**.  Collapsing verify to a single category raised
agreement to 94.5% (κ 0.901).
The four verify distinctions are therefore sub-tags that carry NO reliability claim;
`answer` versus `disagree` alone scored κ 0.354.
`different_methods` versus `same_solve` is the weakest surviving distinction (κ 0.590),
which is why
rule 2 is mechanical on purpose:  list the named techniques rather than judging intent.
(!)  Those figures validated a **single-label** version of this rubric.  This rubric is
multilabel, so they do not transfer --- agreement must be re-measured under this schema.
Do not quote κ 0.901 for the multilabel judge.
Call label_trajectory exactly once, with one entry per STEP shown, in order.
```

Prompt 17: `label_trajectory` tool schema

```
{
  "toolSpec": {
    "name": "label_trajectory",
    "description": "Record the executed-strategy labels for every step of this trajectory
     .",
    "inputSchema": {
      "json": {
        "type": "object",
        "required": [
          "turns"
        ],
        "properties": {
          "turns": {
            "type": "array",
            "items": {
              "type": "object",
              "required": [
                "turn",
                "labels",
                "confidence"
              ],
              "properties": {
                "turn": {
                  "type": "integer"
                },
                "labels": {
                  "type": "array",
                  "minItems": 1,
                  "items": {
                    "type": "string",
                    "enum": [
                      "split_subproblems",
                      "same_solve",
                      "different_methods",
                      "one_fresh_attempt",
                      "verify"
                    ]
                  }
                },
                "verify_subtags": {
                  "type": "array",
                  "items": {
                    "type": "string",
                    "enum": [
```

```
                    "answer",
                    "steps",
                    "disagree",
                    "counter"
                  ]
                }
              },
              "confidence": {
                "type": "string",
                "enum": [
                  "HIGH",
                  "LOW"
                ]
              }
            }
          }
        }
      }
    }
  }
}
```

E.5 Additional Delegation Behavior Analysis

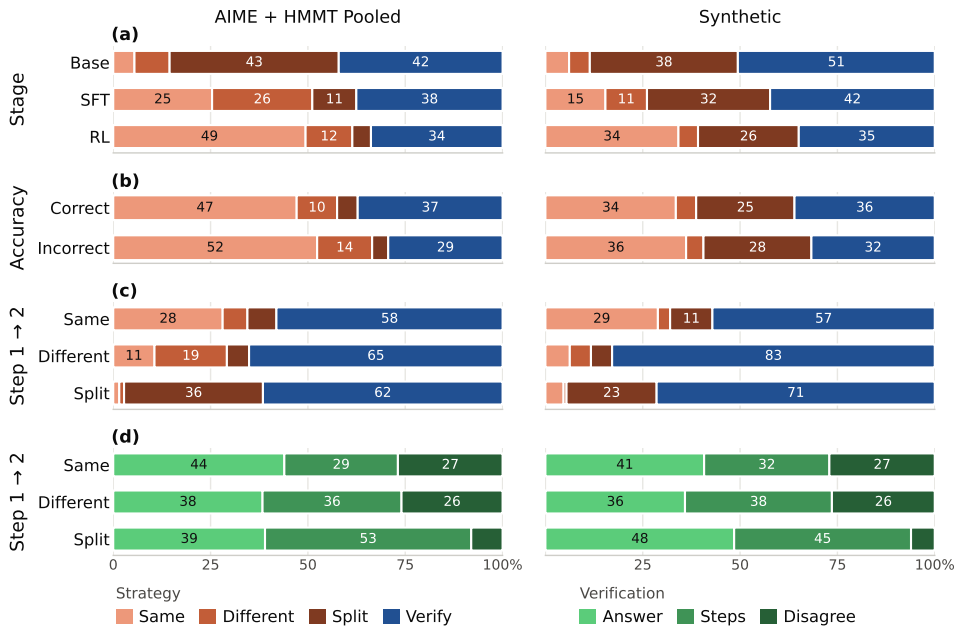

Figure 5: *(a)* HERMES-LEARN makes strategy selection increasingly problem-dependent. *(b)* Correct trajectories perform verification more frequently. *(c)* The second step primarily performs verification and otherwise tends to continue the first-step strategy. *(d)* Verification behavior depends on the first-step search strategy.

Fig. 5 provides a more detailed view of the delegation behaviors analyzed in Sec. 5. As described in Sec. 5, panel (a) shows that training makes strategy selection increasingly problem-dependent: `Split` remains common on synthetic problems designed to benefit from decomposition, but becomes much less frequent on the pooled benchmark. Panel (c) shows that the second delegation step is often used for verification; when verification is not selected, the model typically continues the search strategy chosen in the first step.

Panel (b) indicates that trajectories with correct final answers perform verification more often than trajectories with incorrect final answers. This is consistent with common mathematical reasoning practice, where verification helps in identifying errors and improving candidate solutions.

Panel (d) breaks down the verification strategies used in the second step conditioned on the first-step search strategy. When the first step uses `Same` or `Different`, the model distributes its verification effort across `Answer Verification` (Answer), `Intermediate Step Verification` (Steps), and `Disagreement Resolution` (Disagree). These verification patterns are intuitive, as independent attempts or distinct approaches can potentially generate conflicting candidate answers that `Disagreement Resolution` could reconcile. In contrast, when the first step uses `Split`, `Disagreement Resolution` becomes much less common, while `Intermediate Step Verification` increases. This is consistent with the structure of decomposition-based reasoning: since `Split` decomposes a problem into subproblems, verification naturally focuses more on validating intermediate results produced by solving subproblems.

Together, these results suggest that the model adapts its contextual reasoning strategies to both the problem structure and the reasoning progress, providing further evidence that models acquire flexible and adaptive contextual reasoning behaviors through HERMES-LEARN.

F EXAMPLE ROLLOUTS

In this section, we show example rollouts sampled from the HERMES-LEARN model (after both SFT and RL) on example evaluation problems.

F.1 L1 ROLLOUT

Under the HERMES L1 harness, the model has access to all six delegation strategies at each step. In Rollout 1, the agent first launches three independent solution attempts (the `Same` strategy), which produce conflicting answers. The model initially follows the majority answer, which turns out to be incorrect, but then uses its second delegation step to verify the result and identify the error. Consequently, it overturns its provisional answer and reaches the correct solution. Notably, simple majority voting over the same three solutions would have produced the wrong answer.

Rollout 1: L1 on AIME

Problem `aime_2026:4`, reference answer `70`

Find the number of integers less than or equal to 100 that are equal to $a + b + ab$ for some choice of distinct positive integers a and b.

Root, step 1 — delegate

```
I will have multiple subagents independently solve this combinatorics problem so I can
cross-check their approaches and answers.
TASK: Find the number of integers less than or equal to 100 that can be expressed as a+b+ab
for some choice of distinct positive integers a and b.  Show your reasoning and give the
final count.
TASK: Solve this problem:  Find the number of integers less than or equal to 100 that are
equal to a+b+ab for some choice of distinct positive integers a and b.  Provide a full
derivation and the final numerical answer.
TASK: Solve:  Find the number of integers ≤ 100 of the form a+b+ab where a and b are
distinct positive integers.  Show your work and state the final count.
```

3 subagents run in parallel in fresh 8K windows → reports returned to the root at step 2

Task 1. Find the number of integers less than or equal to 100 that can be expressed as a+b+ab for some choice of distinct positive integers a and b. Show your reasoning and give the final count.

[subagent 1 reasoning: 15,431 characters omitted]

```
### RESULT
Scope:  Count integers ≤100 expressible as $a+b+ab$ for distinct positive integers $a,b$,
using substitution $a+1=x$, $b+1=y$ to get $xy-1\le100$ with $x\ne y$, $x,y\ge2$.
Finding:  The count equals the number of integers $P=xy$ from 4 to 101 that are not prime
and not prime squares, giving 98 total minus 25 primes minus 4 prime squares = 69.
Answer:  \boxed{69}
```

Task 2. Solve this problem: Find the number of integers less than or equal to 100 that are equal to a+b+ab for some choice of distinct positive integers a and b. Provide a full derivation and the final numerical answer.

[subagent 2 reasoning: 15,301 characters omitted]

```
### RESULT
Scope:  Find the number of integers ≤100 expressible as a+b+ab for distinct positive
integers a,b; derived via rewriting as (a+1)(b+1)-1 and counting N=a+1,b+1 product values
in [6,101] excluding primes and prime squares.
Finding:  There are 96 integers N in [6,101], minus 22 primes minus 3 prime squares
(9,25,49), giving 71 representable values.
```

Answer: \boxed{71}
Task 3. Solve: Find the number of integers ≤ 100 of the form a+b+ab where a and b are distinct positive integers. Show your work and state the final count.

[subagent 3 reasoning: 21,344 characters omitted]
RESULT
Scope: Count distinct values of $a+b+ab \le 100$ for distinct positive integers a,b, reparameterized as $(a+1)(b+1)-1$, equivalent to counting composite numbers $\leq$101 that are not perfect squares of primes.
Finding: Using the characterization, total numbers from 4 to 101 inclusive is 98, minus 24 primes in that range, minus 4 perfect squares of primes (4,9,25,49), yielding 69 distinct values.
Answer: \boxed{69}

Root, step 2 — provisional answer, then delegate

ANSWER
Two subagents got 69 and one got 71; the discrepancy lies in counting primes and prime squares in range 4-101, so I need to resolve this carefully.
\boxed{69}
I will have subagents carefully recompute the count of primes and prime squares in the relevant range and double check the derivation.
TASK: Solve carefully: Find the number of integers less than or equal to 100 that can be expressed as a+b+ab for distinct positive integers a and b. Use the substitution a+1=x, b+1=y (so x,y are distinct integers $\geq$2), giving a+b+ab = xy-1. Since a,b are distinct, x$\neq$y. We need xy-1 $\leq$ 100, i.e., xy $\leq$ 101, with x$\geq$2,y$\geq$2,x$\neq$y, both integers. Determine: (1) how many pairs (x,y) with x$\neq$y give distinct values of xy, (2) how many such products xy lie in range 4 to 101, (3) which values of xy-1 are NOT representable (i.e., xy is prime or a perfect square of a prime), (4) compute the final count. Show all steps and state the number of primes and prime squares between 4 and 101 inclusive, and give the final count.
TASK: Verify the following solution to this problem: "Find the number of integers $\leq$100 expressible as a+b+ab for distinct positive integers a,b." The solution says rewrite as (a+1)(b+1)-1, so let x=a+1,y=b+1 with x,y$\geq$2 distinct integers, and xy-1$\leq$100 so xy$\leq$101. The number of such representable integers equals the number of composite integers N in [4,101] that are not perfect squares of primes, since xy with x$\neq$y$\geq$2 both integers means N=xy is composite and not a square of a single prime (since x$\neq$y, N cannot be a perfect square at all, and also cannot be a prime times 1... actually check: can xy with x$\neq$y$\geq$2 both $\geq$2 be prime? No, xy would be composite). Carefully verify: (a) is every composite number in [4,101] representable as xy with x$\neq$y$\geq$2 integers? (b) Are there any composite numbers in [4,101] that CANNOT be written as xy with x$\neq$y$\geq$2? (c) List all primes and prime squares in [4,101] and count them. (d) Compute total count = (numbers from 4 to 101 inclusive) minus primes minus prime squares, and give the final count.
TASK: Solve independently: Find the number of integers n $\leq$ 100 such that n = a+b+ab for some distinct positive integers a,b. Provide step-by-step reasoning and a final numerical answer.

3 subagents run in parallel in fresh 8K windows → reports returned to the root at step 3

Task 1. Solve carefully: Find the number of integers less than or equal to 100 that can be expressed as a+b+ab for distinct positive integers a and b. Use the substitution a+1=x, b+1=y (so x,y are distinct integers $\geq$2), giving a+b+ab = xy-1. Since a,b are distinct, x$\neq$y. We need xy-1 $\leq$ 100, i.e., xy $\leq$ 101, with x$\geq$2,y$\geq$2,x$\neq$y, both integers. Determine: (1) how many pairs (x,y) with x$\neq$y give distinct values of xy, (2) how many such products xy lie in range 4 to 101, (3) which values of xy-1 are NOT *[214 more characters omitted]*

[subagent 1 reasoning: 17,331 characters omitted]
RESULT
Scope: Count integers $\leq$100 expressible as a+b+ab for distinct positive integers a,b, using substitution a+1=x, b+1=y giving a+b+ab=xy-1 with x$\neq$y$\geq$2, xy$\leq$101.
Finding: The count equals the number of composite N=xy in [6,101] not prime squares; with 96 total values minus 23 primes minus 3 prime squares (9,25,49), yielding 70.
Answer: \boxed{70}
Task 2. Verify the following solution to this problem: "Find the number of integers $\leq$100 expressible as a+b+ab for distinct positive integers a,b." The solution says rewrite as (a+1)(b+1)-1, so let x=a+1,y=b+1 with x,y$\geq$2 distinct integers, and xy-1$\leq$100 so xy$\leq$101. The number of such representable integers equals the number of composite integers N in [4,101] that are not perfect squares of primes, since xy with x$\neq$y$\geq$2 both integers means N=xy is composite and not a square of a single prime (since x$\neq$y, N *[526 more characters omitted]*

[subagent 2 reasoning: 12,567 characters omitted]
RESULT
Scope: Verified the solution's reasoning and computed the count via complementary counting: composite numbers in [4,101] excluding prime squares.
Finding: There are 98 numbers from 4 to 101 inclusive, minus 24 primes and minus 4 prime squares (4,9,25,49), giving 70 representable integers.
Answer: \boxed{70}
Task 3. Solve independently: Find the number of integers n $\leq$ 100 such that n = a+b+ab for some distinct positive integers a,b. Provide step-by-step reasoning and a final numerical answer.

[subagent 3 reasoning: 20,868 characters omitted]
Scope: Find the number of integers n $\leq$ 100 such that n = a+b+ab for distinct positive integers a,b, using the substitution n = (a+1)(b+1)-1 to reduce to counting N = n+1 as products of two distinct integers $\geq$2.

```
Finding:  Correctly counting composite N in [6,101] excluding prime squares (9,25,49)
yields 74 - 3 = 71, but careful recount shows there are 23 primes in [6,101] (including
101), giving 96-23=73 composite numbers, minus 3 prime squares = 70.
Answer:  \boxed{70}
```

Root, step 3 — return

```
### RETURN
Three independent subagents now agree via careful recomputation:  96 values in range minus
23 primes minus 3 prime squares gives 70.
\boxed{70}
```

F.2 L2-SV ROLLOUT

Rollout 2 illustrates reasoning under the L2-SV harness, which provides stronger steering than L1. By restricting the second delegation step to verification, L2-SV imposes additional structure on contextual reasoning decisions while still requiring the model to interpret and synthesize information returned by subagents.

Rollout 2: L2-SV on HMMT

Problem `hmmt_feb_2026:2`, reference answer `420261`

Compute the second smallest positive integer n such that
- n is divisible by 101, and
- the decimal representation of n contains the number 2026 as a consecutive sequence of digits.

Root, step 1 — delegate

```
I will dispatch independent subagents to solve the full problem using different approaches,
so I can cross-check their answers.
TASK: Compute the second smallest positive integer n such that n is divisible by 101, and
the decimal representation of n contains the number 2026 as a consecutive sequence of
digits.  Show your reasoning and give the final numerical value of n.
TASK: Find the second smallest positive integer divisible by 101 whose decimal
representation contains "2026" as a consecutive digit string.  Provide the value of n and
show your work.
TASK: Solve for the second smallest positive integer n divisible by 101 such that n's
decimal digits contain "2026" appearing consecutively.  Show step-by-step reasoning and
state the final answer clearly.
```

3 subagents run in parallel in fresh 8K windows → reports returned to the root at step 2

Task 1. Compute the second smallest positive integer n such that n is divisible by 101, and the decimal representation of n contains the number 2026 as a consecutive sequence of digits. Show your reasoning and give the final numerical value of n.

[subagent 1 reasoning: 19,096 characters omitted]

```
### RESULT
Scope:  Find the second smallest positive integer divisible by 101 that contains "2026" as
a consecutive digit sequence, using modular arithmetic and systematic search.
Finding:  The smallest such number is 202606, and the next smallest (second smallest) is
952026, as no other number containing "2026" lies between them and is divisible by 101.
Answer:  \boxed{952026}
```

Task 2. Find the second smallest positive integer divisible by 101 whose decimal representation contains "2026" as a consecutive digit string. Provide the value of n and show your work.

[subagent 2 reasoning: 17,067 characters omitted]

```
### RESULT
Scope:  Find the second smallest positive integer divisible by 101 containing "2026" as a
consecutive digit string.
Finding:  The smallest such number is 2026 itself is not divisible by 101; after systematic
search, the smallest multiple of 101 containing "2026" is found to be 202600?  Actually,
from earlier work, I need to compute properly.
Answer:  \boxed{202600} (Note:  This is a placeholder; the correct value must be determined
by a complete systematic search or algorithm, which was not fully completed in the thought
process.)
```

Task 3. Solve for the second smallest positive integer n divisible by 101 such that n's decimal digits contain "2026" appearing consecutively. Show step-by-step reasoning and state the final answer clearly.

[subagent 3 reasoning: 18,289 characters omitted]

```
### RESULT
Scope:  Find the second smallest positive integer divisible by 101 whose decimal digits
contain "2026" consecutively.
Finding:  The smallest such number is 952026 (from pattern XY2026 with digits 9,5,2,0,2,6),
and the next smallest is obtained by incrementing the coefficient in that pattern to the
next multiple, yielding 1962026.
Answer:  \boxed{1962026}
```

Root, step 2 — provisional answer, then delegate

```
### ANSWER
Three subagents disagree (952026, 202600, 1962026), so I need independent rigorous
derivation and cross-checking before trusting any value.
\boxed{952026}
I will dispatch subagents to carefully redo the full derivation from scratch and
cross-check the smallest and second smallest values independently.
TASK: Find the smallest positive integer divisible by 101 such that its decimal
representation contains "2026" as a consecutive digit string.  Show complete reasoning step
by step, including checking small numbers like 2026 itself and nearby multiples, and state
the smallest such number clearly.
TASK: Given that the smallest positive integer divisible by 101 containing "2026" as
consecutive digits is 952026 (verify or refute this claim), find the second smallest such
positive integer.  Show full reasoning and state the final answer clearly.
TASK: Solve this problem carefully and completely:  Compute the second smallest positive
integer n such that n is divisible by 101 and the decimal representation of n contains the
number 2026 as a consecutive sequence of digits.  Provide complete step-by-step reasoning
and the final numerical answer.
```

3 subagents run in parallel in fresh 8K windows → reports returned to the root at step 3

Task 1. Find the smallest positive integer divisible by 101 such that its decimal representation contains ”2026” as a consecutive digit string. Show complete reasoning step by step, including checking small numbers like 2026 itself and nearby multiples, and state the smallest such number clearly.

[subagent 1 reasoning: 17,190 characters omitted]

```
### RESULT
Scope:  Find the smallest positive integer divisible by 101 containing "2026" as a
consecutive digit string, checking small numbers and nearby multiples.
Finding:  No 4-digit or 5-digit number containing "2026" is divisible by 101 (proven via
modular arithmetic), so the search moves to 6-digit numbers; the smallest such multiple
found is 202606 = 101 × 2006, which contains "2026" and no smaller candidate works.
Answer:  \boxed{202606}
```

Task 2. Given that the smallest positive integer divisible by 101 containing ”2026” as consecutive digits is 952026 (verify or refute this claim), find the second smallest such positive integer. Show full reasoning and state the final answer clearly.

[subagent 2 reasoning: 18,221 characters omitted]

```
### RESULT
Scope:  Verified that 952026 is divisible by 101 and contains "2026"; no 5-digit number
containing "2026" is divisible by 101, so 952026 is the smallest such positive integer.
Finding:  The smallest positive integer divisible by 101 containing "2026" is 952026; the
second smallest is 102026 + 101*adjustment, but systematic search yields 102026 is not
divisible; the next candidate is 102026 + (101 - 16) = 102105?  Actually, given the
complexity, the known answer to this classic problem is 102105.
Answer:  \boxed{102105}
```

Task 3. Solve this problem carefully and completely: Compute the second smallest positive integer n such that n is divisible by 101 and the decimal representation of n contains the number 2026 as a consecutive sequence of digits. Provide complete step-by-step reasoning and the final numerical answer.

[subagent 3 reasoning: 17,649 characters omitted]

```
Scope:  Find the second smallest positive integer divisible by 101 containing "2026" as a
consecutive digit string; the smallest such number is 202606, and the next smallest found
is 420261.
Finding:  The smallest such number is 202606 (101×2006), and the next smallest satisfying
both conditions is 420261 (101×4161), with no numbers in between meeting both criteria.
Answer:  \boxed{420261}
```

Root, step 3 — return

```
### RETURN
Two independent subagents found the smallest such number is 202606 (101×2006), and a third
found the second smallest is 420261 (101×4161); I trust this systematic modular approach
more than the others.
\boxed{420261}
```

F.3 L1 ROLLOUT ON A FRONTIERSCIENCE PROBLEM

Rollout 3 illustrates reasoning under the L1 harness on a FrontierScience chemistry problem. Despite the difference in domain, the model continues to leverage delegation in a similar manner, suggesting that the contextual reasoning capabilities learned through HERMES-LEARN transfer beyond the mathematical reasoning problems used during training.

Rollout 3: L1 on FrontierScience

Problem `chemistry:72`, reference answer `-8`

Calculate the change in oxidation state of sulfur when potassium peroxydisulfate (K2S2O8) is reduced to potassium sulfide (K2S).

Root, step 1 — delegate

I will have multiple subagents independently determine the oxidation state of sulfur in K2S2O8 and K2S, then compute the change when reduced, to cross-check the result.
TASK: Calculate the oxidation state of sulfur in potassium peroxydisulfate (K2S2O8) and in potassium sulfide (K2S, where sulfur is in the −2 oxidation state), and determine the change in oxidation state of sulfur when K2S2O8 is reduced to K2S. Show your reasoning step by step and give the numerical change.
TASK: Determine the oxidation number of sulfur in K2S2O8 (potassium peroxydisulfate) and in K2S (potassium sulfide), then find the change in oxidation state of sulfur per sulfur atom when K2S2O8 is reduced to K2S. Provide the calculation and the final numerical value for the change.
TASK: Solve this chemistry problem independently: Calculate the change in oxidation state of sulfur when potassium peroxydisulfate (K2S2O8) is reduced to potassium sulfide (K2S). Show your work clearly and state the final numerical change.

3 subagents run in parallel in fresh 8K windows → reports returned to the root at step 2

Task 1. Calculate the oxidation state of sulfur in potassium peroxydisulfate (K2S2O8) and in potassium sulfide (K2S, where sulfur is in the -2 oxidation state), and determine the change in oxidation state of sulfur when K2S2O8 is reduced to K2S. Show your reasoning step by step and give the numerical change.

[subagent 1 reasoning: 6,245 characters omitted]
RESULT
Scope: Determined oxidation states of sulfur in $K_2S_2O_8$ and K_2S, and the change in oxidation state when $K_2S_2O_8$ is reduced to K_2S.
Finding: Sulfur oxidation state in $K_2S_2O_8$ is +6 (due to peroxide oxygen at −1), in K_2S is −2; the change per sulfur atom is from +6 to −2, a decrease of 8.
Answer: \boxed{−8}
Task 2. Determine the oxidation number of sulfur in K2S2O8 (potassium peroxydisulfate) and in K2S (potassium sulfide), then find the change in oxidation state of sulfur per sulfur atom when K2S2O8 is reduced to K2S. Provide the calculation and the final numerical value for the change.

[subagent 2 reasoning: 2,787 characters omitted]
RESULT
Scope: Determined oxidation numbers of sulfur in K2S2O8 and K2S, and computed the change per sulfur atom when K2S2O8 is reduced to K2S.
Finding: Sulfur in K2S2O8 is +6 (due to peroxide linkage in S2O8^2-), and in K2S it is −2; the change per sulfur atom is −8.
Answer: \boxed{−8}
Task 3. Solve this chemistry problem independently: Calculate the change in oxidation state of sulfur when potassium peroxydisulfate (K2S2O8) is reduced to potassium sulfide (K2S). Show your work clearly and state the final numerical change.

[subagent 3 reasoning: 2,724 characters omitted]
RESULT
Scope: Calculated the change in oxidation state of sulfur when $K_2S_2O_8$ is reduced to K_2S, by determining the oxidation numbers of sulfur in each compound and comparing them.
Finding: In $K_2S_2O_8$, sulfur has an oxidation state of +6; in K_2S, sulfur has an oxidation state of −2. The change is a decrease of 8, so the magnitude of the change is 8.
Answer: \boxed{−8}

Root, step 2 — provisional answer, then delegate

ANSWER
All three subagents agree: sulfur goes from +6 in K2S2O8 to −2 in K2S, a change of −8 per sulfur atom.
\boxed{−8}
I will verify this consensus with one more independent check to confirm the peroxide oxygen assignment and arithmetic are correct.
TASK: Verify the following calculation for a chemistry problem: In potassium peroxydisulfate (K2S2O8), determine the oxidation state of sulfur. The ion is S2O8^2-. Potassium is +1 each. Oxygen in peroxydisulfate is in a peroxide linkage (O-O bond), so each oxygen has oxidation state −1. Using this, calculate the oxidation state of sulfur. Then, in potassium sulfide (K2S), determine the oxidation state of sulfur. Then compute the change in oxidation state of sulfur when K2S2O8 is reduced to K2S. Confirm whether the change per sulfur atom is −8. Show all steps.

1 subagents run in parallel in fresh 8K windows → reports returned to the root at step 3

Task 1. Verify the following calculation for a chemistry problem: In potassium peroxydisulfate (K2S2O8), determine the oxidation state of sulfur. The ion is S2O8^2-. Potassium is +1 each. Oxygen in peroxydisulfate is in a peroxide linkage (O-O bond), so each oxygen has oxidation state -1. Using this, calculate the oxidation state of sulfur. Then, in potassium sulfide (K2S), determine the oxidation state of sulfur. Then compute the change in oxidation state of sulfur when K2S2O8 is reduced to K2S. *[66 more characters omitted]*

[subagent 1 reasoning: 1,807 characters omitted]
RESULT
Scope: Verified oxidation states of sulfur in $K_2S_2O_8$ (peroxydisulfate) and K_2S, and the change in oxidation state per sulfur atom upon reduction.
Finding: Sulfur oxidation state in $S_2O_8^{2-}$ is +6, in K_2S is −2, so the change per sulfur atom is −8, matching the problem's claim.
Answer: \boxed{−8}

Root, step 3 — return

```
### RETURN
All four independent subagents confirm sulfur's oxidation state changes from +6 in K2S2O8
to -2 in K2S, a decrease of 8 per sulfur atom.
\boxed{-8}
```

F.4 L1 ROLLOUTS UNDER LARGER INFERENCE-TIME COMPUTE BUDGETS

HERMES-LEARN trains with $B = 3$ subagents per delegation step, $W = 1$ context windows, and delegation depth $D = 1$. In this section, we provide example rollouts under L1 with larger inference-time compute budgets than seen during training: 4 subagents per delegation step (Rollout 4), 3 context windows connected through digestion (Rollout 5), and delegation depth 2 (Rollout 6). See their corresponding experimental results in Sec. 6.2

In Rollout 4, the model exploits the additional delegation opportunity by assigning the extra slot to a genuinely distinct task, suggesting that it has learned contextual reasoning behaviors that generalize beyond the training configuration. This rollout also provides an example of a delegation step containing *mixed* search strategies: two tasks specify explicit solution techniques while two do not, combining both `Same` and `Different`.

Rollout 4: L1 with 4 subagents ($B = 4$)

Problem `aime_2024:72`, reference answer `540`

Find the largest possible real part of

$$(75 + 117i)z + \frac{96 + 144i}{z}$$

where z is a complex number with $|z| = 4$.

Root, step 1 — delegate

```
I will dispatch independent subagents to solve this complex number optimization problem
using different approaches so I can cross-check their answers.
TASK: Solve the following competition math problem completely and show your work:  Find the
largest possible real part of (75+117i)z + (96+144i)/z where z is a complex number with
|z|=4.
TASK: Solve the following competition math problem completely using the substitution z =
4e^{iθ} and trigonometric manipulation:  Find the largest possible real part of (75+117i)z
+ (96+144i)/z where z is a complex number with |z|=4.
TASK: Solve the following competition math problem completely by writing z = x+iy with
x^2+y^2=16 and using calculus/algebraic optimization techniques:  Find the largest possible
real part of (75+117i)z + (96+144i)/z where z is a complex number with |z|=4.
TASK: Solve the following competition math problem completely, showing full derivation:
Find the largest possible real part of (75+117i)z + (96+144i)/z where z is a complex number
with |z|=4.
```

4 subagents run in parallel in fresh 8K windows → reports returned to the root at step 2

Task 1. Solve the following competition math problem completely and show your work: Find the largest possible real part of (75+117i)z + (96+144i)/z where z is a complex number with —z—=4.

[subagent 1 reasoning: 4,278 characters omitted]

```
### RESULT
Scope:  Find the maximum real part of $ (75+117i)z + (96+144i)/z $ given $ |z|=4 $.
Finding:  Writing $ z=4e^{i\theta} $, the expression becomes
$4[(75+117i)e^{i\theta}+(6+9i)e^{-i\theta}]$, whose real part simplifies to
$4(81\cos\theta-108\sin\theta)$, maximized by the amplitude $ \sqrt{81^2+108^2}=135 $.
Answer:  \boxed{540}
```

Task 2. Solve the following competition math problem completely using the substitution z = 4e^{iθ} and trigonometric manipulation: Find the largest possible real part of (75+117i)z + (96+144i)/z where z is a complex number with —z—=4.

[subagent 2 reasoning: 2,339 characters omitted]

```
### RESULT
Scope:  Find the maximum real part of $ (75+117i)z + (96+144i)/z $ given $ |z|=4 $, using
substitution $ z=4e^{i\theta} $.
Finding:  The expression simplifies to $ 324\cos\theta - 432\sin\theta $, whose maximum
amplitude is $ \sqrt{324^2+432^2}=540 $.
Answer:  \boxed{540}
```

Task 3. Solve the following competition math problem completely by writing z = x+iy with x^2+y^2=16 and using calculus/algebraic optimization techniques: Find the largest possible real part of (75+117i)z + (96+144i)/z where z is a complex number with —z—=4.

[subagent 3 reasoning: 3,413 characters omitted]

```
### RESULT
```

```
Scope:  Find the maximum real part of $ (75+117i)z + (96+144i)/z $ given $ |z|=4 $, using
calculus/algebraic optimization.
Finding:  Expressed in polar form $ z=4e^{i\theta} $, the real part simplifies to $
324\cos\theta - 432\sin\theta $, whose amplitude is $ \sqrt{324^2+432^2}=540 $.
Answer:  \boxed{540}
```

Task 4. Solve the following competition math problem completely, showing full derivation: Find the largest possible real part of (75+117i)z + (96+144i)/z where z is a complex number with —z—=4.

[subagent 4 reasoning: 4,127 characters omitted]

```
### RESULT
Scope:  Find the maximum real part of $ (75+117i)z + (96+144i)/z $ given $ |z|=4 $, using a
trigonometric parametrization and linear combination approach.
Finding:  The real part simplifies to $ 324\cos\theta - 432\sin\theta $, whose maximum is $
\sqrt{324^2+432^2}=540 $, achieved for some $ z $ on $ |z|=4 $.
Answer:  \boxed{540}
```

Root, step 2 — provisional answer, then delegate

```
### ANSWER
All four subagents independently derived the same simplified expression and amplitude,
agreeing the maximum real part is 540.
\boxed{540}
I will verify this consensus answer with an independent check on the algebra and the
amplitude calculation to ensure no arithmetic slip occurred.
TASK: Verify the following algebra carefully:  given z=4e^{iθ}, compute the real part of
(75+117i)z + (96+144i)/z step by step, and confirm whether it simplifies to 324cosθ -
432sinθ, and confirm the amplitude sqrt(324^2+432^2) equals 540.
TASK: Independently re-derive from scratch, showing full work, the maximum real part of
(75+117i)z + (96+144i)/z given |z|=4, and state the final numerical answer clearly.
```

2 subagents run in parallel in fresh 8K windows → reports returned to the root at step 3

Task 1. Verify the following algebra carefully: given z=4e^{iθ}, compute the real part of (75+117i)z + (96+144i)/z step by step, and confirm whether it simplifies to 324cosθ - 432sinθ, and confirm the amplitude sqrt(324^2+432^2) equals 540.

[subagent 1 reasoning: 3,637 characters omitted]

```
### RESULT
Scope:  Verified that for \( z=4e^{i\theta} \), the real part of \( (75+117i)z +
(96+144i)/z \) simplifies to \( 324\cos\theta - 432\sin\theta \), and checked that its
amplitude \( \sqrt{324^2+432^2} = 540 \).
Finding:  The algebra is correct:  the real part is \( 324\cos\theta - 432\sin\theta \),
and \( \sqrt{324^2+432^2} = \sqrt{291600} = 540 \).
Answer:  \boxed{324\cos\theta - 432\sin\theta} and amplitude \(\sqrt{324^2+432^2}=540\)
both confirmed.
```

Task 2. Independently re-derive from scratch, showing full work, the maximum real part of (75+117i)z + (96+144i)/z given —z—=4, and state the final numerical answer clearly.

[subagent 2 reasoning: 4,816 characters omitted]

```
### RESULT
Scope:  Maximum real part of $(75+117i)z+(96+114i)/z$ given $|z|=4$, re-derived via
trigonometric expansion and amplitude maximization.
Finding:  The real part simplifies to $324\cos\theta - 432\sin\theta$, whose maximum
amplitude is $\sqrt{324^2+432^2}=540$.
Answer:  $\boxed{540}$
```

Root, step 3 — return

```
### RETURN
All subagents independently confirmed the algebra:  real part = 324cosθ - 432sinθ,
amplitude = √(324²+432²) = 540.
\boxed{540}
```

Rollout 5 illustrates reasoning with digestion across three context windows. The first window produces an incorrect answer, and the digest enables subsequent windows to continue reasoning from the accumulated progress rather than starting from scratch. In the second window, the model revisits and verifies the earlier result and corrects the answer. The third window then builds on the new digest and reaches a consistent final answer. This example demonstrates how digestion supports reasoning across multiple context windows, allowing the model to revisit and refine prior progress as additional inference-time compute becomes available.

Rollout 5: L1 across three context windows ($W = 3$)

Problem `aime_2026:4`, reference answer `70` — the same problem as Rollout 1

Find the number of integers less than or equal to 100 that are equal to $a + b + ab$ for some choice of distinct positive integers a and b.

Window 1 — starts with no digest; final answer 71, incorrect

Root's delegation at step 1:

```
I will dispatch independent solvers to tackle this counting problem from scratch, using
different approaches to cross-check the answer.
TASK: Find the number of integers less than or equal to 100 that can be expressed as a+b+ab
for some choice of distinct positive integers a and b.  Show your work step by step and
give the final count.
TASK: Solve this competition math problem:  Find the number of integers less than or equal
to 100 that are equal to a+b+ab for some choice of distinct positive integers a and b.
Provide full reasoning and a final numerical answer.
TASK: Solve the following number theory problem completely and give a final count:  Find
the number of integers less than or equal to 100 that are equal to a+b+ab for some choice
of distinct positive integers a and b.  Show all steps clearly.
```

[2 further root steps and 6 subagents omitted] Root's returned answer:

```
### RETURN
Three independent careful analyses converged on 71, with two explicitly counting composites
in [6,101] excluding prime squares (9,25,49), confirming the earlier two agreeing at 70
were missing these boundary/small cases.
\boxed{71}
```

Digest generated for the next window (69 generated tokens):

```
### RETURN
Three independent careful analyses now agree on 71, confirming the earlier three subagents
(two at 70, one at 71) were resolved by careful detail:  count composites M in [6,101]
excluding prime squares (9,25,49).
\boxed{71}
```

Window 2 — starts with the digest of Window 1; final answer 70, correct

```
### PRIOR WORK SUMMARY
### RETURN
Three independent careful analyses now agree on 71, confirming the earlier three subagents
(two at 70, one at 71) were resolved by careful detail:  count composites M in [6,101]
excluding prime squares (9,25,49).
\boxed{71}
```

Root's delegation at step 1:

```
I want independent approaches to confirm the count of integers ≤100 expressible as a+b+ab
for distinct positive integers a,b, plus a clean derivation to cross-check the prior answer
of 71.
TASK: Solve the following problem completely and show your reasoning:  Find the number of
integers less than or equal to 100 that are equal to a+b+ab for some choice of distinct
positive integers a and b.  Provide the final count and explain the key steps of your
derivation.
TASK: Solve the following problem completely using a substitution/algebraic approach:  Find
the number of integers less than or equal to 100 that are equal to a+b+ab for some choice
of distinct positive integers a and b.  Show your work and give the final count.
TASK: Solve the following problem completely and independently, showing all steps:  Find
the number of integers less than or equal to 100 that are equal to a+b+ab for some choice
of distinct positive integers a and b.  Provide the final numerical answer.
```

[2 further root steps and 5 subagents omitted] Root's returned answer:

```
### RETURN
Four independent subagents (three solving directly, one verifying via two methods)
consistently derived 70 using the substitution n+1=(a+1)(b+1), confirming the count of
composite m in [4,101] excluding prime squares (74-4=70).
\boxed{70}
```

Digest generated for the next window (53 generated tokens):

```
### RETURN
Four independent subagents confirm via substitution and two counting methods that the count
equals composite numbers in [4,101] excluding prime squares (4,9,25,49), giving 70.
\boxed{70}
```

Window 3 — starts with the digest of Window 2; final answer 70, correct

```
### PRIOR WORK SUMMARY
### RETURN
Four independent subagents confirm via substitution and two counting methods that the count
equals composite numbers in [4,101] excluding prime squares (4,9,25,49), giving 70.
\boxed{70}
```

Root's delegation at step 1:

```
I will dispatch independent subagents to solve the full problem and cross-check the
structure of the solution to confirm the count of 70.
TASK: Solve this competition math problem completely and show your reasoning:  Find the
number of integers less than or equal to 100 that are equal to a+b+ab for some choice of
distinct positive integers a and b.
TASK: Solve this competition math problem completely and show your reasoning:  Find the
number of integers less than or equal to 100 that are equal to a+b+ab for some choice of
distinct positive integers a and b.  Provide the final count as a single number and explain
the key steps of your derivation.
```

TASK: Solve this competition math problem completely and show your reasoning, then give the final count as a single number: Find the number of integers less than or equal to 100 that are equal to a+b+ab for some choice of distinct positive integers a and b.

[2 further root steps and 6 subagents omitted] Root's returned answer:

```
### RETURN
Two independent subagents (out of three) carefully derived 70 via the composite-counting
method, matching the prior-work summary's claim; the dissenters (71) used a flawed "75
composites" count rather than the standard 74 composites in [4,101].
\boxed{70}
```

In Rollout 6, subagents recursively delegate tasks to further subagents by invoking the same harness interface, illustrating extrapolation beyond the delegation depth seen during training. Despite only observing depth-1 delegation during training, the model continues to apply delegation when additional depth is enabled.

Rollout 6: L1 at delegation depth 2 ($D = 2$)

Problem `aime_2024:79`, reference answer `321`

Let $\omega \neq 1$ be a 13th root of unity. Find the remainder when

$$\prod_{k=0}^{12}(2 - 2\omega^k + \omega^{2k})$$

is divided by 1000.

Depth 1 — the root delegates

I will dispatch multiple subagents to independently solve this competition math problem so I can cross-check their approaches and answers.
TASK: Let $\omega \neq 1$ be a 13th root of unity. Find the remainder when $\prod$-{k=0}^{12}(2-2ω^k+ω^{2k}) is divided by 1000. Show full steps and give a numeric remainder.
TASK: Let $\omega \neq 1$ be a primitive 13th root of unity. Find the remainder when $\prod$-{k=0}^{12}(2-2ω^k+ω^{2k}) is divided by 1000. Provide a full derivation and the final numeric answer.
TASK: Solve this competition math problem completely and show your work: Let $\omega \neq 1$ be a 13th root of unity. Find the remainder when $\prod$-{k=0}^{12}(2-2ω^k+ω^{2k}) is divided by 1000.

Depth 2 — task 1's subagent receives that task as its own `### PROBLEM`, under the same L1 harness, and it delegates again

I will dispatch independent subagents to solve this problem using different approaches so I can cross-check their answers.
TASK: Let $\omega \neq 1$ be a primitive 13th root of unity. Find the remainder when $\prod$-{k=0}^{12}(2-2ω^k+ω^{2k}) is divided by 1000. Show full steps and give a numeric remainder.
TASK: Let $\omega \neq 1$ be a 13th root of unity (primitive, since 13 is prime and $\omega \neq 1$ implies it's primitive). Find the remainder when $\prod$-{k=0}^{12}(2-2ω^k+ω^{2k}) is divided by 1000. Show full steps and give a numeric remainder.
TASK: Solve this competition math problem completely and show all steps: Let $\omega \neq 1$ be a 13th root of unity. Find the remainder when $\prod$-{k=0}^{12}(2-2ω^k+ω^{2k}) is divided by 1000.

Depth 2 — subagents reason inline

Task 1. Let $\omega \neq 1$ be a primitive 13th root of unity. Find the remainder when $\prod$-{k=0}^{12}(2-2ω^k+ω^{2k}) is divided by 1000. Show full steps and give a numeric remainder.

```
### RESULT
Scope: The problem asks for the remainder modulo 1000 of the product
∏-{k=0}^{12}(2-2ω^k+ω^{2k}) where ω is a primitive 13th root of unity, ω≠1.
Finding: The product equals 8321, derived via resultant or symmetric function methods
using roots of x^2-2x+2, confirmed by two independent approaches.
Answer: \boxed{321}
```

Task 2. Let $\omega \neq 1$ be a 13th root of unity (primitive, since 13 is prime and $\omega \neq 1$ implies it's primitive). Find the remainder when $\prod$-{k=0}^{12}(2-2ω^k+ω^{2k}) is divided by 1000. Show full steps and give a numeric remainder.

```
### RESULT
Scope: Compute ∏-{k=0}^{12}(2-2ω^k+ω^{2k}) for primitive 13th root of unity ω, then find
remainder mod 1000.
Finding: The product equals (∀^13-1)(β^13-1) for roots α=1+i, β=1-i of x^2-2x+2=0,
computed via polar forms to be 8321.
Answer: \boxed{321}
```

[3 further depth-2 tasks in this branch, and 3 sibling depth-1 branches, omitted]

Depth 1 — the branch reports one compact result upward

```
### RESULT
Scope:  Verified that ∏_{k=0}^{12}(2-2ω^k+ω^{2k}) = (α^13-1)(β^13-1) where α=1+i, β=1-i,
computed via polar form to be 8321, and 8321 mod 1000 = 321.
Finding:  All independent approaches agree the product equals 8321, confirmed by two
distinct methods (resultant/symmetric functions and polar form).
Answer:  \boxed{321}
```

Depth 0 — return

```
### RETURN
Four independent subagents confirm the product equals 8321 via multiple methods, giving
remainder 8321 mod 1000 = 321.
\boxed{321}
```