SCALING TEST-TIME COMPUTE BY LEARNING TO ORCHESTRATE EXPLORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are increasingly used to solve hard reasoning problems with large search spaces, where success can require sustained exploration over substantial test-time compute. We study how today's LLMs use this compute and find that chain-of-thought (CoT) is *contractive*: as reasoning progresses, models devote increasingly less computation to novel directions and instead concentrate on approaches discovered earlier. Existing reasoning harnesses broaden exploration, but do not eliminate this contraction. We therefore study a harness in which an *orchestrator* explicitly decides which directions to explore and delegates them to parallel execution models. Current LLMs still struggle in this role, exhibiting redundant delegation, premature termination, and substantial problem-solving within the orchestrator itself. We hypothesize that this reflects a mismatch between standard reasoning training and the requirements of orchestration: models are trained to solve problems within their own reasoning trajectories, but receive little supervision for deciding where additional computation should be allocated. To address this, we introduce **`LOrE`** (*Learning to Orchestrate Exploration*), which first midtrains a model on synthetic exploration trajectories to establish orchestration priors, then applies multi-turn RL within the orchestrator-executor harness to optimize downstream task success. Applied to a 9B-parameter model, **`LOrE`** sustains broader exploration, scales effectively to millions of generated test-time tokens, and performs well on research-level benchmarks.

1 INTRODUCTION

While there has been substantial progress in training large language models (LLMs) to solve hard reasoning problems (Guo et al., 2025; Wang et al., 2026), many research-level problems at the frontier of science, mathematics, and related domains remain challenging. The hardest problems often present large search spaces, little indication of which one will work, and solutions that depend on discovering unexpected connections between different ideas (Feng et al., 2026). Solving them can therefore require substantial *exploration*: a model may need to consider several strategies, rule out unproductive ones, connect ideas across different approaches, and revise its approach as new information is uncovered. In these settings, scaling test-time compute requires not only spending more tokens developing a solution, but also using that compute to explore a broader set of approaches.

Does training models to generate longer chains of thought (CoT) suffice to support this sort of exploration? In principle, a CoT (Wei et al., 2022) can implement an arbitrarily long search process. In practice, however, current training objectives reward models for reaching a correct answer within a finite reasoning budget. This creates pressure to progressively concentrate computation on promising directions rather than continually opening new ones, consistent with prior observations (Xu et al., 2026). We refer to this tendency as *exploration contraction*. Intuitively, as reasoning progresses, a CoT can settle into the "basin" of a small set of high-level approaches it has already discovered, spending subsequent compute developing, verifying, or repairing reasoning within that region. In Section 3.1, we observe this behavior directly: the fraction of reasoning devoted to novel high-level directions falls sharply with depth, while later reasoning becomes more predictable overall and increasingly local, with recent context remaining substantially more informative than reasoning from much earlier in the trajectory. Contraction is useful, and perhaps necessary, for eventually committing to and completing a solution within a fixed budget. However, it also means that simply extending a CoT provides diminishing opportunities to reach qualitatively different parts of the search space, limiting test-time scaling on difficult, open-ended problems.

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.

A natural approach to addressing this limitation is to use *harnesses* that structure test-time compute across multiple model calls, for example through parallel sampling, compaction, or aggregation (Wu et al., 2026; Venkatraman et al., 2025). Such harnesses can broaden search beyond a single CoT trajectory. However, our analysis of two representative harnesses in Section 3.2 shows that they do not eliminate contraction: they increase the absolute number of directions explored, but the rate at which new directions are discovered still declines as more compute is spent. We hypothesize that this is because these harnesses still rely on new directions to emerge implicitly from CoTs. This motivates making the allocation of exploration itself explicit. We therefore consider a simple harness that utilizes an *orchestrator* that tracks what has already been explored, proposes complementary directions to investigate, delegates them in parallel to execution models, and uses the resulting findings to decide where subsequent test-time compute should be allocated.

Can current models, particularly open-weight models, effectively orchestrate exploration through this harness? In Section 3.3, we find that they struggle: models often spend substantial compute *solving* the problem themselves before delegating, propose redundant directions within and across rounds, and terminate exploration prematurely. We hypothesize that these behaviors reflect a mismatch between the demands of orchestration and the behaviors rewarded by standard RL training. An orchestrator must allocate computation across multiple investigations, including uncertain directions whose value may emerge only through subsequent exploration (Russell & Wefald, 1991). Standard RL training (even with harnesses above) instead rewards completing a solution within the model's own reasoning budget, encouraging it to concentrate effort on immediately promising approaches. Our observations suggest that these learned behaviors persist even when models are placed in an orchestrator role, limiting their ability to turn compute into sustained, coordinated exploration.

To address this, we introduce **LOrE**, a two-stage procedure for training LLMs to orchestrate exploration. First, we construct synthetic traces and midtrain a model to acquire useful priors for proposing directions, avoiding redundant work, and continuing to explore when useful work remains. We then perform end-to-end multi-turn RL within our orchestrator-executor harness, optimizing downstream task success together with an exploration bonus that regularizes the policy toward sustained exploration. This allows **LOrE** to move beyond imitating hand-designed exploration behaviors and learn which exploration strategies actually improve the performance of the full reasoning system. We train a 9B orchestrator with **LOrE** and show that it scales effectively to millions of test-time tokens across research-level benchmarks in computer science, mathematics, and the sciences. Our behavioral analysis shows that the trained orchestrator sustains exploration of new directions with limited redundancy and makes fuller use of its available compute budget, both by delegating more directions in parallel and by continuing exploration over more sequential rounds.

2 Related Work

Test-time compute and reasoning harnesses. Early approaches scale test-time compute through sampling and selection (Cobbe et al., 2021; Wang et al., 2023; Brown et al., 2024) or search over intermediate reasoning states (Yao et al., 2023; Hao et al., 2023; Zhou et al., 2024). Others use iterative refinement (Madaan et al., 2023; Qu et al., 2024; Kumar et al., 2025), budget forcing (Muennighoff et al., 2025), or combined sequential and parallel compute (Snell et al., 2025). Recent harnesses support longer or more structured inference (Cursor Research, 2026; Wu et al., 2026; Venkatraman et al., 2025; Shao et al., 2025; LM-Provers et al., 2026), but additional compute may not expand the set of *semantic* problem-solving directions explored (Zhu et al., 2026). RLMs (Zhang et al., 2025) provide greater flexibility by allowing models to recursively decompose tasks and execute parallel work. While **LOrE** trains orchestrators work to similarly delegate work to parallel executors, our focus is on whether the model can continually discover complementary semantic directions as test-time compute scales. We study reasoning problems where useful exploration does not admit an obvious decomposition into independent subtasks (Gandhi et al., 2026), and find that even strong open-weight models struggle to sustain broad exploration in this setting (Section 3).

Multi-agent RL and orchestration. Another line of work studies how multiple LLMs can jointly allocate compute to solve tasks, including multi-agent debate, mixture-of-agents, and multi-agent training (Du et al., 2024; Wang et al., 2025a; Motwani et al., 2025; Park et al., 2025). Recent methods learn when to spawn and join reasoning threads (Pan et al., 2025), train shared policies for recursive delegation and task solving (Gandhi et al., 2026), or support dynamic decomposition, parallel delegation, and tool selection (Kimi Team, 2026; Yuan et al., 2026; Su et al., 2025). Our approach uses similar infrastructure but targets a different capability. Typical agentic tasks assign

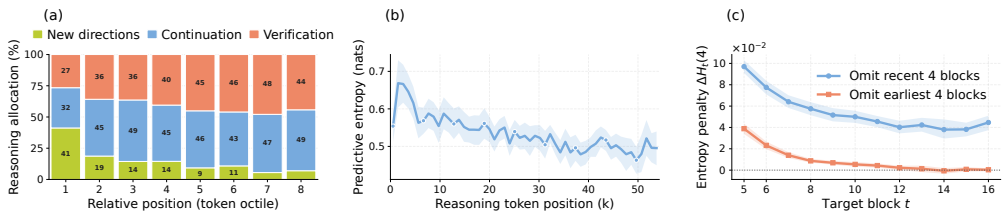


Figure 1: ***Reasoning contracts with depth.*** **(a)** Across TCSBench traces, the share of reasoning devoted to new directions declines over reasoning token octiles, while continuation and verification increase. **(b)** Next-token entropy, measured in 1,024-token windows, generally decreases with token position. **(c)** Removing either four recent or four earliest 4,096-token blocks incurs a diminishing entropy penalty with reasoning depth. Recent blocks remain more predictive than equally sized distant context. Shading denotes bootstrap 95% CI.

subagents well-defined, complementary subtasks; our non-agentic setting requires exploring speculative, overlapping, or competing directions whose value is initially uncertain. We therefore midtrain models to propose semantically diverse directions, then optimize their allocation with RL using a tailored reward and credit attribution scheme.

Exploration on hard problems. Recent work identifies exploration within CoTs as a bottleneck on hard reasoning tasks (Setlur et al., 2026b;a; Qu et al., 2026; Li et al., 2026). Prior work addresses this through prompt-length curricula (Setlur et al., 2026b), hint-guided training (Li et al., 2026; Qu et al., 2026; Zhang et al., 2026) or selectively optimizing high-entropy tokens within the CoT (Wang et al., 2025b). These methods improve exploration within individual CoTs without directly targeting coverage of distinct semantic directions. Xu et al. (2026) also find that uncertainty decreases as reasoning progresses, supporting our view that CoT reasoning becomes increasingly contractive. We instead study how to sustain exploration *across* coordinated trajectories, explicitly training an orchestrator to allocate additional test-time compute toward complementary semantic directions rather than repeatedly revisiting the same regions of the search space.

3 Exploration through Orchestration

Why can orchestration improve exploration beyond simply allocating more tokens to standard CoT reasoning? To answer this question, we study how exploration changes as test-time compute (TTC) scales under standard CoT and reasoning harnesses on research-level problems in theoretical computer science (TCS). We observe that CoTs exhibit a tendency we refer to as *exploration contraction*. At the semantic level, as a trajectory progresses, the model increasingly develops and verifies directions or solution approaches it has already discovered rather than opening new ones. At the token level, future computation becomes more predictable from recent context, with progressively less dependence on reasoning much earlier in the trajectory, suggesting a reduced tendency to revisit earlier directions. Such contraction is arguably useful, and perhaps necessary, for eventually committing to a direction, working out an answer fully, and receiving task reward. However, it also means that simply extending a CoT provides diminishing opportunities to explore new regions of the solution space. We further find that independent parallel sampling and compaction-based harnesses exhibit a similar contraction. Together, these observations motivate explicit orchestration as a mechanism for *actively* deciding where additional computation should explore.

3.1 Exploration within CoTs Tends to Contract

An appealing mental model is to view CoT reasoning as implementing a serialized search process under a finite token budget. For models trained with outcome-reward RL, this creates an incentive to balance exploration against completing a solution before the budget is exhausted. As reasoning progresses, a new approach has less remaining compute in which to yield a solution. This gives rise to pressure favoring continued investment in established directions that appear promising in lieu of exploration. We hypothesize that this encourages *exploration contraction*: the progressive concentration of computation on previously established approaches. In the following experiments, we investigate whether capable open-weight exhibit this behavior by examining how the introduction of new directions and token-level predictability change with reasoning depth. Our findings provide evidence of contraction and complement related observations on token entropy (Xu et al., 2026).

CoTs devote progressively less reasoning to new directions. We first examine how the exploration of new problem-solving directions changes with CoT depth. Such directions include adopting a different proof technique or introducing a major new line of argument, rather than continuing an estab-

lished approach or verifying previous claims. We sample Qwen3.5-9B (Qwen Team, 2026) rollouts on **TCSBench**, a rubric-based evaluation dataset of research-level TCS problems constructed for this work (Appendix A), and divide each rollout into eight equal-token segments. We use GPT-5.6-sol (`medium`) to identify high-level reasoning moves within each segment and estimate the proportion of reasoning devoted to three categories: **(a)** *New Directions*, encompassing distinct approaches and major developments within existing approaches; **(b)** *Continuation*, further developing established directions; and **(c)** *Verification*, checking previous work. Appendix D provides the annotation and aggregation details. The estimated proportion of reasoning devoted to *New Directions* falls sharply from 41% in the first octile to 7% in the last (Figure 1(a)). Thus, as reasoning progresses, models increasingly develop and verify established approaches rather than introduce new ones.

Later CoT predictions become more predictable and increasingly local. Our preceding analysis suggests that models introduce fewer new approaches as reasoning progresses. We next study a token-level signature of the same contraction: whether later reasoning is easier to predict, and how much its prediction depends on recent versus distant context. We partition each CoT into non-overlapping blocks of 4,096 tokens. For each target block b_i, we teacher-force the observed tokens and measure their average next-token entropy under the full preceding context. We then remove either the $g = 4$ immediately preceding blocks or the earliest g preceding blocks and recompute the entropy on exactly the same target tokens. We denote the resulting changes relative to the full-context entropy by $\Delta H_i^{\text{recent}}(g)$ and $\Delta H_i^{\text{early}}(g)$, respectively. The problem $\mathbf{x}$ is always retained; Appendix C provides definitions, implementation details, and results using other values of g.

Figures 1(b) and (c) reveal three complementary patterns. First, full-context entropy decreases with reasoning depth, indicating that later reasoning becomes increasingly predictable under the model. Second, removing recent blocks incurs a consistently larger entropy penalty than removing the same number of earliest blocks, showing that later predictions are more sensitive to their recent reasoning context than to equally sized distant context. This is consistent with the current subgoal, intermediate conclusions, and local reasoning state being represented primarily in recent blocks. Third, the effect of removing either recent or early context also decreases with depth: deleting a fixed amount of reasoning becomes less consequential later in the trajectory, even when the deleted blocks immediately precede the target. Thus, later reasoning is both more predictable overall and more robust to partial removal of its preceding context, consistent with the findings of Aghajohari et al. (2026). We observe the same qualitative trends using token NLL in Appendix C, showing that the pattern extends beyond predictive entropy to token sequences explicitly sampled from the model.

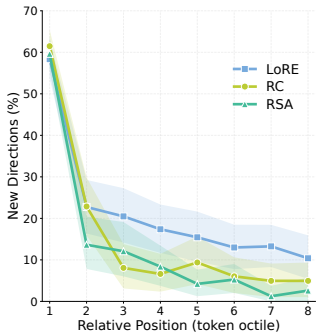


Figure 2: ***New directions on TCSBench.*** The proportion declines with reasoning depth (measured via turns/rounds in this case) across all three harnesses, but the decline is less pronounced with the **`LOrE`** harness.

Together with the direction-level analysis, these results support a picture of *exploration contraction*. As reasoning progresses, new semantic directions become rarer, while token-level prediction enters an increasingly concentrated and local "basin": recent context remains more influential than distant context, yet no fixed portion of the preceding trajectory is individually as consequential as it was earlier in reasoning. This is consistent with the model increasingly developing a well-established line of reasoning rather than moving to qualitatively different regions of the search space.

3.2 Do Harnesses Fix CoT Contraction?

A natural question is whether reasoning harnesses can overcome CoT contraction. To study this, we evaluate two harnesses representative of modern systems: sequential context compaction via Reasoning Cache (Wu et al., 2026) and parallel sampling followed by aggregation via RSA (Venkatraman et al., 2025); see Appendix F for details. As shown in Figure 2, both harnesses broaden exploration at the initial turn, producing more distinct directions than standard CoT. However, neither eliminates contraction: as additional test-time compute is spent across subsequent turns/rounds, the rate at which new directions are discovered still declines sharply. Thus, wrapping a pretrained model in a fixed harness can increase exploration in absolute terms, but does not quite change how coverage over high-level directions scales with compute. This is perhaps expected: while harnesses based on summarization, aggregation, or verification allow models to remain productive for longer, they still rely largely on the model's own CoTs to generate new directions, and those CoTs them-

selves eventually contract. This motivates making the allocation of exploration directions explicit and adaptive, rather than leaving it implicit within a fixed harness.

3.3 Orchestrating Exploration With The LOrE Harness

We call a model that directs exploration an *exploration orchestrator*. It can maintain an explicit search frontier, propose directions informed by prior exploration, and use the resulting findings to allocate subsequent test-time compute. Pairing an orchestrator with a problem-solving model ("executor") provides a natural mechanism for sustaining exploration across both breadth and depth. Crucially, this separation also reduces the pressure for the orchestrator itself to contract toward a solution: each executor can focus on developing a particular direction toward a concrete outcome, while the orchestrator remains free to allocate compute across multiple, potentially uncertain directions even if it consolidates its work on some directions. This raises a key question: can pretrained LLMs already serve as effective orchestrators when explicitly prompted to do so?

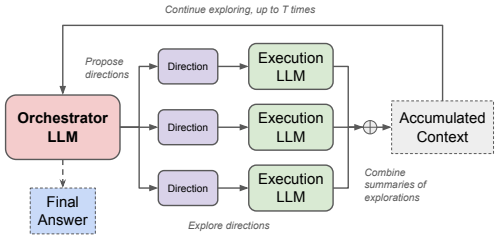


Figure 3: ***The LOrE harness.*** An orchestrator proposes directions and delegates them to executors running in parallel. The findings are summarized and added to an accumulated context, which informs subsequent exploration. Upon stopping or reaching the turn limit, it synthesizes a final answer from the accumulated findings.

The LOrE harness. Our harness, that we call the **LOrE** (learning to orchestrate exploration) harness employs an explicit *orchestrator* π and an *executor* μ, which may be the same or different LLMs. Given a problem $\mathbf{x}$, the orchestrator runs for up to T steps. At step t, it receives the problem together with an accumulated exploration context C_t (initialized as $C_1 = \varnothing$), and produces a reasoning trace followed by explicit exploration *directions*, $D_t = \{d_{t,1}, \ldots, d_{t,K_t}\} \sim \pi(\cdot \mid x, C_t, \mathcal{I}_\pi)$,where $\mathcal{I}_\pi$ is the orchestrator instruction prompt and K_t is the variable number of directions proposed at turn t. Each direction specifies a line of investigation, and is dispatched independently to the executor models μ. For each direction $d_{t,k}$, the executor also receives $\mathbf{x}$ and C_t, and is instructed to investigate that direction, $s_{t,k} \sim \mu(\cdot \mid x, C_t, d_{t,k}, \mathcal{I}_\mu)$, where $\mathcal{I}_\mu$ denotes the executor instruction prompt. The executor returns a compact summary $s_{t,k}$ of its findings, which is appended to the accumulated context, $C_{t+1} = C_t \oplus \left\{(d_{t,k}, s_{t,k})\right\}_{k=1}^{K_t}$, where $\oplus$ denotes concatenation. In this way, each turn exposes the orchestrator to the results of all previous exploration and allows subsequent directions to be conditioned on what has already been discovered.

At each turn, the orchestrator may either continue exploration or terminate. Upon termination, we replace the exploration instruction $\mathcal{I}_\pi$ with an answer instruction $\mathcal{I}_A$ and ask the orchestrator to produce a final answer using the accumulated context: $y \sim \pi(\cdot \mid x, C_t, \mathcal{I}_A)$. Termination may be selected by the orchestrator itself or forced at an arbitrary depth or a maximum number of turns T. The separation between orchestration and execution allows us to scale exploration at the orchestration level, by increasing the number of directions considered, while preserving the useful pressure for each executor to contract toward a solution within a given direction.

Open-weight LLMs are poor orchestrators of exploration. We now study whether current LLMs can effectively serve as orchestrators, and whether this capability improves by simply scaling the model. We evaluate several open-weight models on TCSBench and **ArxivMath-0626** (Dekoninck et al., 2026), a research-level, final answer-based mathematics benchmark. For each model, we use the same LLM as both orchestrator π and executor μ, allowing up to $K = 5$ directions per turn and $T = 8$ exploration turns. We compare against standard CoT reasoning and naïve parallel sampling under matched test-time compute. We sample $N = 3$ trajectories for all experiments and use GPT-5.6-Luna (medium) as the judge for ArxivMath, and GPT-5.6-Luna (high) as the judge for TCSBench. See Appendix N for additional evaluation details, and H.1 for our prompts.

Finding 1: Several LLMs fail to benefit from the LOrE harness. Both Qwen3.5 models and Inkling-Small obtain limited gains from our harness despite using substantially more test-time compute. Scaling within the Qwen3.5 family from 9B to 122B (Qwen Team, 2026), does not qualitatively change this, suggesting that effective orchestration does not emerge automatically with scale. DeepSeek-V4-Flash-0731 (DeepSeek-AI, 2026) obtains substantially larger gains. However, even

Table 1: ***Benchmark performance (left) and orchestration behavior (right) for various open-weight models and reasoning harnesses.*** *Solve* indicates significant problem solving, *Back-Gen* indicates back-generation, *Red* indicates redundant directions, while *Term* indicates early termination on incorrect trajectories.

	TCSBench				ArxivMath				Orch. Behavior on TCSBench			
Model	CoT	RC	RSA	`LOrE`	CoT	RC	RSA	`LOrE`	**Solve↓**	**Back-Gen↓**	**Red↓**	**Term↓**
Qwen3.5-9B	11.0	13.6	17.9	13.5	20.1	23.7	25.0	22.2	**21.7**	**14.2**	46.5	66.4
Qwen3.5-122B-A10B	21.9	24.9	32.3	23.5	28.5	30.6	38.6	30.6	22.7	17.7	31.7	68.3
Inkling-Small	36.6	43.4	39.5	38.6	37.5	41.7	47.9	37.5	40.4	21.8	44.0	98.3
DeepSeek-V4-Flash	50.4	63.9	67.7	61.3	46.5	61.9	68.8	55.9	62.6	33.7	**8.5**	**60.6**

for these models, the RSA parallel-sampling harness performs better (although it uses substantially more compute: see Appendix G), suggesting that they still do not orchestrate effectively.

Finding 2: LLMs exhibit complementary failure modes as orchestrators. We use GPT-5.6-Sol to annotate orchestration CoTs for three failure modes: *significant problem-solving*, where the orchestrator performs work that should be delegated to an excutor; *back-generation*, where it delegates directions it has already substantially explored itself; and *redundant directions* across or within rounds. See Appendix E for details. Qwen3.5 and Inkling-Small exhibit substantial direction repetition, while DeepSeek-V4-Flash is less redundant but frequently solves the problem within the orchestrator CoT and then back-generate directions, suggesting that its gains arise largely from standard CoT rather than effective orchestration. On problems the model fails to solve (rubric score below $5/7$), we observe frequent *early termination*: despite having exploration budget remaining, all models stop before the maximum number of rounds roughly 60% of the time, while Inkling-Small terminates nearly all rollouts by round six. These results suggest that current LLMs neither allocate exploration cleanly across executors nor reliably sustain it when additional compute is available.

4 Learning Exploration Priors via Synthetic Midtraining

Our results suggest that existing models lack the behavioral priors required to effectively orchestrate exploration. We therefore run *midtraining* on synthetic data to instill these priors before optimizing the orchestrator end-to-end with RL. Unlike training that targets improving general problem-solving capabilities, the goal of this midtraining phase is to teach general orchestration behaviors: proposing focused directions, maintaining diversity, avoiding redundant exploration, incorporating previous findings, and continuing exploration when useful work remains.

4.1 Constructing Synthetic Demonstrations of Effective Orchestration

Because existing models are themselves poor orchestrators, we cannot simply distill orchestration rollouts from a stronger teacher, particularly when their CoTs exhibit the same pathological behaviors we aim to correct. Therefore, unlike prior midtraining approaches (Wang et al., 2025c) that directly train on long CoT traces, we never train on the original CoT used to generate exploration directions. Instead, we use only final outputs from models together with LLM-based judgments to construct demonstrations of desirable exploration behavior. Our pipeline consists of three stages:

Propose and select directions. At each round, we generate candidate directions conditioned on a problem and the accumulated findings so far, using a mixture of proprietary and open-weight models (Appendix B). We then prompt GPT-5.6-Luna to filter out directions that overlap with previous or concurrent exploration and rank the survivors by expected utility. Note that we do not optimize directions for correctness: incorrect directions may still be useful for learning how to explore. We discard the CoTs of the direction proposer due to various pathologies from above.

Execute directions and accumulate findings. We use DeepSeek-V4-Flash to independently execute each selected direction. The resulting findings are summarized and added to the exploration context. We then return to Step 1 and continue proposing and executing directions until a separate judge (DeepSeek-V4-Flash) deems the accumulated exploration sufficient, based on its coverage of the problem's reference solution. At that point, we use DeepSeek to generate a final answer conditioned on the accumulated exploration context. We retain the DeepSeek CoTs from both the execution and final-answer generation steps and add them to our dataset.

Synthesize orchestration reasoning. For each step of a trace, we prompt DeepSeek with only the context available at that point, together with the candidate directions and the judgments indicating whether each was selected or rejected as redundant or lower in expected utility. DeepSeek then

synthesizes a forward-looking CoT that weighs the candidate directions and culminates in proposing the selected ones. Importantly, it sees neither the original CoT that generated the directions nor the outcomes of future rounds. We filter out synthesized CoTs that contain substantial problem-solving rather than orchestration reasoning, and retain the remainder for training. See Appendix B for details. We apply this synthetic midtraining pipeline to 4,500 problems, split roughly evenly between Olympiad-style proofs from FineProofs (Beeching et al., 2026) and theoretical computer science problems from TCSBench-Training (Appendix A.1). We generate one or two trajectories per problem, allowing up to eleven rounds before forcing final-answer generation. The resulting dataset contains 30k orchestration CoTs, 50k execution CoTs, and 6k final-answer CoTs. We midtrain Qwen3.5-9B on this dataset to obtain **LOrE**-MidT-9B (Appendix L).

4.2 Results: Midtraining Improves Exploration Orchestration

We assess whether **LOrE**-MidT-9B enables more effective test time scaling than the base model. As shown in Table 2, midtraining substantially increases gains from the harness: while the harness improves the base model by only 1-5% across benchmarks, it improves the midtrained model by 7-13%. On the proof-based tasks most closely aligned with our training data, TCS-Bench and **IMOProofBench** (Luong et al., 2025), midtraining improves even direct CoT performance. In contrast, direct CoT performance does not improve on Arxiv-Math, an answer-based benchmark less aligned with the training tasks, and on the out-of-domain **FrontierScience Olympiad**, which comprises challenging physics, chemistry and biology problems (Wang et al., 2026). Nevertheless, the performance on ArxivMath with our harness increases from 22.2% for the base model to 29.9% for the midtrained model. This suggests that the benefits of orchestration training are behavioral, and not purely knowledge or problem solving-based.

Table 2: ***Midtraining increases the gains from orchestration.*** Performance (%) of Qwen3.5-9B (Base) and **LOrE**-MidT-9B (MidT) under direct CoT and **LOrE** inference. Δ denotes the harness gain in percentage points.

Model		TCS Bench	Arxiv Math	IMO ProofB.	Frontier Sci.
Base	CoT	11.1	20.8	25.8	54.3
	LOrE	13.6	22.2	31.0	59.3
	Δ	+2.5	+1.4	+5.2	+5.0
MidT	CoT	20.3	19.4	40.1	55.0
	LOrE	27.3	29.9	52.8	62.3
	Δ	+7.0	+10.4	+12.7	+7.3

We repeat the behavioral analysis from Section 3.3 to examine how midtraining changes orchestration (see Table 4). We see a significant decrease in back-generation and redundant direction generation, while the incidence of substantial problem-solving within the orchestrator's CoT decreases more modestly, from 21.7% to 19.2%. Midtraining also reduces premature termination: among trajectories that ultimately produce incorrect if, early termination falls from 66% to 24%. We also repeat the direction analysis experiments conducted in Section 3.3, and compare the proportion of directions generated by our midtrained model and the base mode classified as *new directions*: see Figure 4. We find that the midtrained model generates new directions even late into reasoning, with nearly 30% of directions remaining new in the last octile of reasoning depth. Overall, our analysis indicates that **LOrE**-MidT-9B both sustains exploration for longer and explores more distinct approaches than its base counterpart. These behavioral changes provide a plausible explanation for the stronger orchestration performance attained by our model.

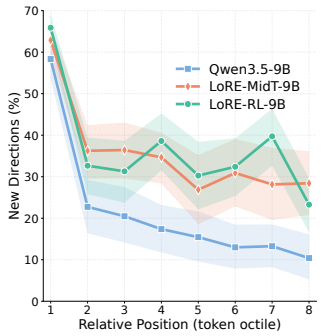


Figure 4: ***New directions on TCSBench.*** Both midtrained and RL models generate new directions even later into reasoning.

5 Improving the Orchestration of Exploration with RL

After imbuing the model with useful behavioral priors for exploration, we use multi-turn reinforcement learning (RL) to train it to use these behaviors for solving downstream tasks, moving beyond imitation of synthetic demonstrations. Our RL procedure builds on standard outcome-reward training, but introduces two critical design choices.

Orchestration-only multi-turn RL. We initialize the orchestrator policy from **LOrE**-MidT-9B and generate training rollouts using the **LOrE** harness, allowing up to $T = 8$ turns and $K = 5$ directions per turn. We assign outcome rewards based on the final answer and propagate them to all orchestration steps, optimizing direction selection and final-answer generation while leaving the executor

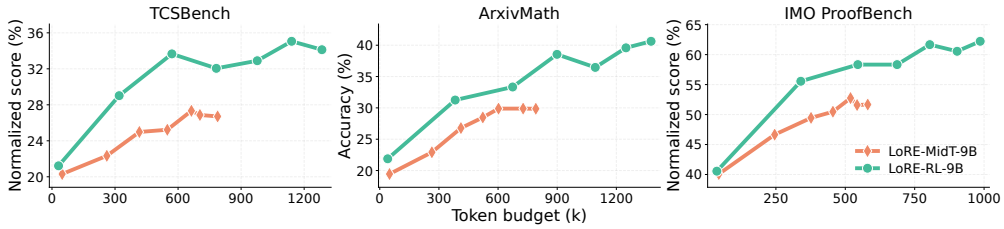


Figure 5: ***Test-time scaling after midtraining and RL.*** Each point corresponds to a maximum number of exploration turns, with a final answer forced at that limit unless the model terminates earlier. The horizontal axis reports cumulative generated tokens averaged over all problems, including those that have already terminated. **LOrE**-RL-9B generally achieves higher performance than the midtrained model at comparable compute.

unchanged. To isolate improvements in orchestration and simplify optimization over multiple model roles, we use a frozen copy of **LOrE**-MidT-9B as the executor throughout both training and evaluation. We optimize the orchestrator using CISPO (Chen et al., 2025). Our training data combines held-out problems from TCSBench-Training, and Olympiad-level proof problems from FineProofs (LM-Provers et al., 2026), and answer-based problems from ArXivMath-Training (Dekoninck et al., 2026). We evaluate rubric-based answers using GPT-5.6 Luna (`high`). After approximately 200 optimizer steps, we obtain **LOrE**-RL-9B. Appendices A and M provide further details on the training data and RL configuration. See Appendix K for a discussion on judge choices.

Table 3: ***Multi-turn RL further improves performance with the LOrE harness.*** TCSBench and IMO ProofBench report mean rubric scores; ArxivMath and FrontierScience Olympiad report pass@1 acc. All values are percentages. MidT and RL models use the **LOrE** harness.

Model	TCS Bench	Arxiv Math	IMO ProofB.	Frontier Sci.
Qwen3.5-9B (CoT)	11.1	20.8	25.8	54.3
Qwen3.5-9B (**LOrE**)	13.6	22.2	31.0	59.3
LOrE-MidT-9B	27.3	29.9	52.8	62.3
LOrE-RL-9B	**32.9**	**39.6**	**60.8**	**66.7**

Additional details. We introduce two non-standard mechanisms to stabilize RL. First, in preliminary experiments with standard multi-turn RL, we observed that the number of proposed directions gradually decreased, while an increasing fraction of directions verified previously explored work. To counteract this early collapse in exploration, we add a small *exploration bonus* during the first 100 training steps. At each orchestration turn, a judge evaluates proposed directions for *progress*, whether they request useful work likely to advance the investigation, and *redundancy*, whether they substantially overlap with previous or sibling directions. Each direction judged both progressive and non-redundant contributes 0.05 to the rollout reward, with the total bonus capped at 0.25 per turn. We remove the bonus after 100 steps, using it only as a training regularizer to establish broader exploration early on. Second, we maintain a *context buffer* of partial explorations, represented by their accumulated contexts and associated problems, initialized from trajectories generated by the midtrained model. Rather than starting every rollout from an empty context, we sample a buffered context and continue exploration from that state. Buffered contexts contain at most three completed exploration steps, and we sample context depths uniformly. As training produces new explorations, we update the buffer with their accumulated contexts. This trains the policy to build on findings from earlier trajectories, and reduces rollout cost by enabling reuse of completed exploration prefixes.

Results. We compare **LOrE**-RL-9B against **LOrE**-MidT-9B and base Qwen3.5-9B, all evaluated with the **LOrE** harness with $T = 8$, on the in-domain benchmarks TCSBench, ArxivMath, and IMO ProofBench, as well as the out-of-domain FrontierScience Olympiad benchmark. Appendix N provides evaluation details. As shown in Table 3, RL substantially improves over midtraining across all benchmarks. To test scaling beyond the train-time exploration horizon, we increase the maximum number of turns to $T = 12$ and plot performance against cumulative generated tokens in Figure 5. Performance continues to improve beyond 1M generated tokens on both TCSBench and ArxivMath, although marginal gains diminish at larger compute budgets.

Table 4: ***Orchestration behavior of LOrE-RL and LOrE-MidT*** on TCSBench with the **LOrE** harness.

	Orch. behavior				Exploration		
Model	Solve↓	Back↓	Red↓	Term↓	Dirs. / step	Steps (max 12)	Dirs. / prob.
Base	21.7	14.2	46.5	66.4	2.5	4.9	12.2
MidT	19.2	5.0	**13.6**	24.1	2.7	6.8	18.6
RL	**5.3**	**1.4**	20.3	**4.1**	4.5	7.6	34.4

How do orchestration behaviors change after RL? We repeat the behavioral analysis from Section 3.3 on **LOrE**-RL-9B trajectories from TCSBench and report the results in Table 4. We additionally measure the mean number of directions proposed per turn, the mean number of turns

before termination, and the mean total number of directions proposed per problem. Relative to the midtrained model, RL further reduces substantive problem-solving within the orchestrator and back-generation of directions. Redundancy increases slightly, likely because the model has become more exploratory and thus generates far more directions, but this nonetheless remains substantially below the base model's level. Early termination among trajectories with incorrect final answers also decreases. Meanwhile, the model proposes more directions per turn and continues for more steps on average, thereby making fuller use of orchestration, and the number of turns it is allowed to act over. Together, this indicates that RL encourages more orchestration and sustained exploration.

Our midtraining enables effective orchestration RL. Applying our multi-turn RL approach directly to Qwen3.5-9B, without our midtraining (row 6, Table 5), improves performance over standalone CoT but yields little benefit over placing the untrained base model within the **LOrE** harness. We observe that the resulting policy also terminates exploration earlier than the base model, and hypothesize that, without effective orchestration priors, the model obtains insufficient benefit from orchestration, leading RL to reinforce a preference for answering directly. These results highlight the importance of establishing useful exploration behaviors before optimizing orchestration through RL.

Table 5: ***Baseline results.*** Base: Qwen3.5-9B; MidT: **LOrE**-MidT-9B; DS-Dist.: DeepSeek-distilled 9B; Std.: single-trajectory RL. CoT uses no harness. Scores (%) are ArxivMath pass@1 and normalized rubric scores for TCSBench and IMO ProofBench. The final row is **LOrE**-RL-9B.

#	Init.	RL	Infer.	TCS	Arxiv	IMO
1	Base	Std.	CoT	14.9	20.8	30.6
2	MidT	Std.	CoT	28.6	23.6	45.9
3	DS-SFT.	Std.	CoT	28.6	23.5	43.5
4	MidT	Std.	**LOrE**	30.2	23.7	55.8
5	Base	RC	RC	29.9	22.9	51.1
6	Base	**LOrE**	**LOrE**	13.4	21.9	32.2
	Base	None	CoT	11.1	20.8	25.8
	MidT	**LOrE**	**LOrE**	**32.9**	**39.6**	**60.8**

Multi-turn RL improves performance beyond standard reasoning RL. We compare our multi-turn RL procedure against standard CoT RL on the same training data using three initializations: Qwen3.5-9B, **LOrE**-MidT-9B, and DeepSeek-Distilled-9B (rows 1–3, Table 5). The distilled model is obtained by finetuning Qwen3.5-9B on DeepSeek CoT solutions generated for our midtraining problems. These baselines test whether the gains of **LOrE**-RL-9B can be recovered by improving the underlying reasoner without explicitly optimizing multi-turn orchestration. We evaluate all three with standard CoT inference and find that each falls substantially short of **LOrE**-RL-9B. In particular, standard RL from the same **LOrE**-MidT-9B initialization yields weaker performance, showing that the midtrained initialization alone does not account for the gains. The distilled baseline likewise shows that strong-teacher CoT distillation followed by standard RL does not recover the performance of our approach. We additionally evaluate the standard-RL-trained **LOrE**-MidT-9B model with the **LOrE** harness (row 4). Harnessed inference improves its performance, but it remains below **LOrE**-RL-9B, particularly on ArxivMath. This comparison holds both the initialization and inference harness fixed, providing direct evidence that optimizing the orchestrator end-to-end across multiple exploration turns contributes beyond either stronger reasoning training or harnessed inference alone.

Orchestration outperforms compaction-based reasoning. We also train Qwen3.5-9B using the RC protocol of Wu et al. (2026) and evaluate it with the RC harness (row 5). RC training substantially improves over the base model, but remains below **LOrE**-RL-9B across all three benchmarks. This supports the effectiveness of our method relative to sequential scaling through compaction.

Reducing exploration breadth lowers LOrE performance. Finally, we evaluate **LOrE**-RL-9B while restricting execution to the first K directions proposed at each turn. On IMO ProofBench, performance decreases from 60.8% with up to five directions to 58.2% with $K = 3$ and 52.7% with $K = 1$. These results indicate that parallel investigations contribute to the model's performance, although reducing their number also reduces the available test-time compute to the model.

Conclusion and limitations. In this paper, we identify a limitation of test-time scaling: reasoning models increasingly revisit familiar approaches, while existing harnesses fail to sustain broader exploration even with explicit orchestrator roles. We introduce **LOrE**, combining synthetic exploration midtraining with multi-turn RL to learn effective orchestration. Our 9B-parameter orchestrator sustains broader exploration, improves research-level benchmark performance, and scales effectively beyond one million test-time tokens. Limitations include not jointly training the orchestrator and executor, limited credit assignment for delayed-value investigations, reliance on separate judges, and no study of context compaction that preserves useful findings while avoiding repetition.

Ethics Statement

This work investigates methods for improving language-model reasoning through learned orchestration of exploration. Potential benefits include supporting mathematical and scientific research, while potential risks include the misuse of improved reasoning capabilities, overreliance on incorrect model-generated conclusions, and the computational and environmental costs of large-scale training and inference. Our use of synthetic training data and LLM-based evaluation introduces additional risks of factual errors, inherited biases, and imperfect judgments of mathematical correctness. We describe our data construction, filtering, and evaluation procedures to support scrutiny of these limitations. Benchmark performance should not be interpreted as a guarantee of correctness or suitability for consequential applications; model-generated scientific and mathematical conclusions require independent verification.

Reproducibility Statement

We document the **LOrE** orchestration harness and training procedure in Sections 3, 4 and 5. Appendices A and B describe the construction of TCSBench and the synthetic midtraining data, including generation and filtering procedures. Appendices C, D and E detail the reasoning and orchestration analyses, Appendix F describes the baseline harnesses, and Appendix H provides prompts for inference and evaluation. Supervised fine-tuning and reinforcement learning configurations are reported in Appendices L and M, respectively, while Appendix N summarizes the benchmark evaluation configurations. As described in Appendices A and B, we plan to release the datasets, data-generation pipeline, and full synthetic-data generation prompts upon publication to facilitate reproduction and further investigation.

Disclosure of Generative AI Use

We used generative AI tools to assist with implementing experiments, generating synthetic datasets, drafting sections of the manuscript, and editing the text for clarity and readability. Research ideation and experimental design were carried out by the human authors. As described in the paper, LLMs were also used to construct benchmark problems, reference answers, and grading rubrics; generate and filter synthetic exploration trajectories; annotate reasoning and orchestration behaviors; and provide training rewards and evaluation scores. These uses are documented in the corresponding methods and appendix sections. We take responsibility for the final content of this work, including its claims, results, and artifacts produced with the assistance of generative AI.

References

Milad Aghajohari, Kamran Chitsaz, Amirhossein Kazemnejad, Sarath Chandar, Alessandro Sordoni, Aaron Courville, and Siva Reddy. The markovian thinker: Architecture-agnostic linear scaling of reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 46743–46765, 2026.

Edward Beeching, Jasper Dekoninck, Jia Li, Yuxiao Qu, Amrith Setlur, Ian Wu, Aviral Kumar, and Lewis Tunstall. Fineproofs sft dataset: Mathematical olympiad problems with chain-of-thought reasoning. https://huggingface.co/datasets/lm-provers/FineProofs-SFT, 2026.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.

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.

Cursor Research. Composer 2 technical report, 2026. URL https://arxiv.org/abs/2603.24477.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL https://arxiv.org/abs/2605.00674.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 11733–11763. PMLR, 2024.

Tony Feng, Trieu H Trinh, Garrett Bingham, Dawsen Hwang, Yuri Chervonyi, Junehyuk Jung, Joonkyung Lee, Carlo Pagano, Sang-hyun Kim, Federico Pasqualotto, et al. Towards autonomous mathematics research. *arXiv preprint arXiv:2602.10177*, 2026.

Apurva Gandhi, Satyaki Chakraborty, Xiangjun Wang, Aviral Kumar, and Graham Neubig. Recursive agent optimization, 2026. URL https://arxiv.org/abs/2605.06639.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.507.

Devvrit Khatri, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Sai Surya Duvvuri, Manzil Zaheer, Inderjit S. Dhillon, David Brandfonbrener, and Rishabh Agarwal. The art of scaling reinforcement learning compute for llms, 2025. URL https://arxiv.org/abs/2510.13786.

Kimi Team. Kimi K2.5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276.

Jon Kleinberg and Fan Wei. Language generation and identification from partial enumeration: Tight density bounds and topological characterizations, 2025. URL https://arxiv.org/abs/2511.05295.

Kacper Kluk, Hung Le, Wojciech Nadara, Marcin Pilipczuk, Hector Tierno, and Vinayak. A polynomial coreset for furthest neighbor in planar metrics, 2026. URL https://arxiv.org/abs/2603.28100.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M. Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=CjwERcAU7w.

Jiazheng Li, Hongzhou Lin, Hong Lu, Kaiyue Wen, Zaiwen Yang, Jiaxuan Gao, Yi Wu, and Jingzhao Zhang. QuestA: Expanding reasoning capacity in LLMs via question augmentation. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2507.13266.

LM-Provers, Yuxiao Qu, Amrith Setlur, Jasper Dekoninck, Edward Beeching, Jia Li, Ian Wu, Lewis Tunstall, and Aviral Kumar. QED-Nano: Teaching a tiny model to prove hard theorems, 2026. URL https://arxiv.org/abs/2604.04898.

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, 2025. URL https://arxiv.org/abs/2511.01846.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip H. S. Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. MALT: Improving reasoning with multi-agent LLM training. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2412.01928.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1025.

Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. Learning adaptive parallel reasoning with language models. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2504.15466.

Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman E Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. Maporl: Multi-agent post-co-training for collaborative large language models with reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30215–30248, 2025.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. In *Advances in Neural Information Processing Systems*, volume 37, pp. 55249–55285, 2024. URL https://arxiv.org/abs/2407.18219.

Yuxiao Qu, Amrith Setlur, Virginia Smith, Ruslan Salakhutdinov, and Aviral Kumar. POPE: Learning to reason on hard problems via privileged on-policy exploration, 2026. URL https://arxiv.org/abs/2601.18779.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial intelligence*, 49(1-3):361–395, 1991.

Amrith Setlur, Zijian Wang, Andrew Cohen, Paria Rashidinejad, and Sang Michael Xie. Reuse your FLOPs: Scaling RL on hard problems by conditioning on very off-policy prefixes, 2026a. URL https://arxiv.org/abs/2601.18795.

Amrith Setlur, Matthew Y. R. Yang, Charlie Snell, Jeremy Greer, Ian Wu, Virginia Smith, Max Simchowitz, and Aviral Kumar. e^3: Learning to explore enables extrapolation of test-time compute for LLMs. In *International Conference on Learning Representations*, 2026b. URL https://arxiv.org/abs/2506.09026.

Zhihong Shao, Yuxiang Luo, Chengda Lu, Z. Z. Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. DeepSeekMath-V2: Towards self-verifiable mathematical reasoning, 2025. URL https://arxiv.org/abs/2511.22570.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.03314.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhturina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. ToolOrchestra: Elevating intelligence via efficient model and tool orchestration, 2025. URL https://arxiv.org/abs/2511.21689.

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

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations*, 2025a.

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.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xiong-Hui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for LLM reasoning. In *Advances in Neural Information Processing Systems*, 2025b.

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://openreview.net/forum?id=1PL1NIMMrw.

Zengzhi Wang, Fan Zhou, Xuefeng Li, and Pengfei Liu. Octothinker: Mid-training incentivizes reinforcement learning scaling. *arXiv preprint arXiv:2506.20512*, 2025c.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Ian Wu, Yuxiao Qu, Amrith Setlur, and Aviral Kumar. Reasoning cache: Continual improvement over long horizons via short-horizon RL. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.03773.

Ting Xu, Xu He, Yupu Lu, Jiankai Sun, Dong Li, Wai Lam, and Jianye Hao. Unveiling the entropy dynamics of chain-of-thought reasoning, 2026. URL https://arxiv.org/abs/2606.02020.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Wenzhen Yuan, Wutao Xiong, Fanchen Yu, Shengji Tang, Ting Liu, Tao Chen, Peng Ye, Yuzhuo Fu, Wanli Ouyang, and Lei Bai. Small model as master orchestrator: Learning unified agent-tool orchestration with parallel subtask decomposition, 2026. URL https://arxiv.org/abs/2604.17009.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models, 2025. URL https://arxiv.org/abs/2512.24601.

Xuechen Zhang, Zijian Huang, Yingcong Li, Chenshun Ni, Jiasi Chen, and Samet Oymak. Bread: Branched rollouts from expert anchors bridge sft & rl for reasoning. *Advances in Neural Information Processing Systems*, 38:96726–96752, 2026.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 62138–62160. PMLR, 2024.

Deyao Zhu, Xin Zhou, Shengling Qin, Xuekai Zhu, Hangliang Ding, Shu Zhong, Zixin Wen, Zhonglin Xie, Chenhui Gou, Linxuan Ren, et al. Edgebench: Unveiling scaling laws of learning from real-world environments. *arXiv preprint arXiv:2607.05155*, 2026.

Appendices

A TCSBENCH

TCSBench is an evaluation dataset of research-level theoretical computer science (TCS) problems assembled as part of this work. It consists of proof-based problems and grading rubrics extracted from research papers using a custom-built pipeline. Our motivation is to provide high-quality evaluation problems whose difficulty and scope reflect reasoning at the frontier of mathematical and scientific knowledge. By drawing on results from recent research papers, our dataset evaluates exploration, discovery and problem-solving skills beyond standard textbook or competition exercises. Our pipeline converts these results into standalone problems and applies refinement and quality checks to support clear problem statements and consistent rubric-based evaluation. Beyond the present benchmark, the pipeline provides a reusable means of extracting problems from newly published literature, allowing the evaluation set to be refreshed as models and the research frontier advance.

A.1 TCSBENCH CONSTRUCTION

We construct TCSBench from theoretical computer science papers published at one of STOC[1], SODA[2] or FOCS[3]. Our pipeline converts results from these papers into standalone reasoning problems, each accompanied by a reference answer and a rubric for partial-credit evaluation. We will open source both our datasets and our data generation pipeline upon publication of this work.

Paper collection and theorem extraction. We collect paper metadata and available full-text PDFs from conference websites, and then convert the PDFs into Markdown for processing. We use Gemini-3.8-Flash to extract the mathematical components of each paper, including definitions, assumptions, theorems, lemmas, constructions, and proof techniques. We merge duplicate components within each paper and organize them into a directed dependency graph: this approach allows us to keep track of how different components of the paper relate to each other, which, as we discuss later, allows us to provide only the relevant components to the problem generating LLM. Having constructed the base graph, we then augment it with associated proofs, proof sketches, and citations from the main text and appendices. A subsequent review, conducted using GPT-5.6-Sol, checks for missing components, incorrect dependencies and other structural inconsistencies, and then proposes and applies corrections.

Generating standalone reasoning problems. We use the extracted graph to generate problems targeting substantive results from each paper. The pipeline supports two principal task types: *proof writing*, which asks for a proof of a specified result, similar to we can be found in traditional proof-based benchmarks; and *proof strategy*, which asks for a concrete research plan for tackling a more open-ended, high level research problem.

To generate problems, we sample a key mathematical component (the "target") from our dependency graph (usually a theorem or a lemma component). We also extract the target's surrounding dependency components (as specified by our graph), and pass all components to a generator LLM (GPT-5.6-Sol). The generator LLM is instructed to rewrite the theorem or lemma into a standalone problem, and to include the definitions, assumptions, and background necessary to attempt the problem without access to the source paper. To increase problem difficulty, we prompt the LLM to withhold intermediate constructions or results whose disclosure could substantially simplify the problem. We also prompt the generator to reject unsuitable targets, such as trivial corollaries or well-known results the base model will likely have seen before.

Reference answers and grading rubrics. Each generated problem is paired with a reference answer and a rubric totaling seven points. Reference answers are constructed using the extracted mathematical content and proof structure. The rubric is constructed using GPT-5.6-Sol based on this reference answer, and assigns partial credit to substantive components of a solution, identifies

[1] https://acm-stoc.org/

[2] https://www.siam.org/conferences-events/siam-conferences/soda27/

[3] https://ieee-focs.org/

common incorrect approaches, and allows appropriately justified alternative solutions. Depending on the task, the reference answer takes the form of a proof or a high-level a proof strategy.

Refinement and automated quality filtering. We refine our generated problems to remove excessive guidance, such as explicit proof outlines, revealed intermediate steps, or supplied results that provide overly-generour help in establishing the target. This stage is instructed to preserve the mathematical task and the information required to make it self-contained. We use GPT-5.6-Sol as our refiner model.

After refinement, we conduct an LLM audit to assess each question for clarity, self-containedness, fairness, and alignment with its reference answer. Questions are rejected when they contain missing assumptions, ambiguous targets, dependence on unstated source-specific knowledge, or insufficient information for a meaningful attempt. Lemma-identification questions are additionally assessed for whether their answer space is sufficiently constrained to support consistent grading. Only questions that receive a passing audit verdict are retained.

TCSBench and TCSBench-Training. Our final evaluation benchmark contains 120 problems drawn from 60 STOC 2026 and FOCS 2026 papers, with an even split between proof writing and proof strategy problems. We refer to this dataset as **TCSBench** throughout this work, and use it exclusively for evaluation.

In addition to TCSBench, we also generate a much larger training dataset using the same pipeline. This dataset contains approximately 3500 problems, split approximately evenly between proof writing and proof strategy, generated from SODA (2024, 2025, 2026) and FOCS (2025) papers. We use this dataset in our **LOrE**-midtraining pipeline to generate midtraining data, as well as for RL. We refer to this dataset as **TCSBench-Training**.

B LOrE MIDTRAINING DATA PIPELINE

In this section, we provide additional details of our midtraining data pipeline. We also provide abridged versions of the key prompts used in this pipeline. The full versions of these prompts will be accessible in our codebase, which we plan on releasing upon publication of our work.

Dataset. We source problems from TCSBench-Training (see Appendix A) and from FineProofs (Beeching et al., 2026). Both data sources consist of proof-based, rubric-graded problems. More specifically, our midtraining dataset consists of approximately 2500 TCSBench-Training problems and 1800 FineProof problems.

Overview. Our synthetic data pipeline constructs **LOrE** harness trajectories for supervised fine-tuning. Each trajectory contains orchestration, execution, and final-answer reasoning. The pipeline is designed to demonstrate effective exploration while reducing the undesirable orchestration behaviors identified in Section 3.3. In particular, we separate direction generation and selection from the synthesis of orchestration CoTs, allowing us to construct demonstrations without directly copying the direction proposer's reasoning.

The pipeline consists of three stages. Starting from a problem and an empty exploration context, we alternate between proposing and selecting directions (Stage 1) and executing them and assessing completion (Stage 2). Once an accepted final answer has been generated, we synthesize orchestration reasoning for the preceding rounds (Stage 3).

Stage 1: Propose and select directions.

1a. Candidate generation. Given the problem and accumulated exploration context, we generate up to five candidate directions using GPT-5.6-luna (`xhigh`), GPT-5.6-Sol (`medium`), or DeepSeek-V4-Flash-0731 (`high`), balancing generation cost, diversity, and quality. We prompt these models to propose investigations likely to advance the current reasoning. We retain the proposed directions but not the CoTs used to generate them, which may exhibit undesirable orchestration behaviors and are not always accessible for the GPT models.

1b. Screening and ranking. We use GPT-5.6-luna (`xhigh`) to assess candidates for substantial overlap with previously explored directions and with one another. It then ranks non-redundant candidates by their expected utility given the current context. We accept up to three of the highest-ranked non-redundant directions and reject the remainder, retaining the rejection reasons. Selection does not

require that a direction ultimately succeed: an unsuccessful investigation may still provide useful information. If no candidate is judged suitable, we terminate collection and discard the problem.

Stage 2: Execute directions and assess completion.

2a. Execution and context accumulation. We independently execute each selected direction using DeepSeek-V4-Flash-0731 (`high`) and the executor instruction $\mathcal{I}_\mu$. We retain both the execution CoT and its final response. DeepSeek then summarizes each execution, describing the main reasoning steps and key findings. These summaries and their corresponding directions are appended to the accumulated exploration context.

2b. Completion assessment and final-answer generation. We provide DeepSeek with the accumulated context and the ground-truth answer, and ask whether the answer has been established and all major claims have been verified. If the evidence is insufficient, we return to Stage 1 with the updated context, subject to a maximum of eight exploration rounds. We generate a final answer once the completion judge deems the evidence sufficient or the eight-round cap is reached, conditioning generation on the accumulated findings. We evaluate the final answer against the grading rubric and retain trajectories scoring at least $5/7$. Trajectories that reach the round cap remain eligible for training if they meet this same threshold. We additionally assess whether the final-answer CoT uses the accumulated findings or rederives the solution from scratch; we generally observe substantial reuse of the accumulated context. Execution and final-answer reasoning from accepted trajectories are retained for training, and the trajectories proceed to Stage 3 for orchestration CoT synthesis.

Stage 3: Synthesize orchestration reasoning. Once an accepted final answer has been generated, we synthesize orchestration CoTs for the preceding exploration rounds. For each round, DeepSeek receives only the context available at that point, the candidate directions, and their selection or rejection judgments and reasons. It receives neither the original CoT used to propose the directions nor outcomes from subsequent rounds.

3a. Assessment of accumulated findings. We first prompt DeepSeek to write a reasoning fragment that assesses what previous investigations have established, what remains uncertain, and which questions warrant further work.

3b. Deliberation over candidate directions. We then prompt DeepSeek to generate successive reasoning fragments that consider individual candidates, assess their value, and explain why they should be pursued or set aside. These fragments include both accepted and rejected directions, demonstrating how to select useful investigations and reject redundant or lower-priority alternatives.

For both synthesis steps, we retain only the explicitly generated text after DeepSeek's closing `</think>` tag, rather than its underlying generation CoT. We concatenate the fragments into a connected orchestration trace and append the selected directions. We filter out synthesized traces containing substantial problem-solving rather than orchestration reasoning, and add the remaining traces to the dataset.

Stage 1a Prompt

```
 You are planning the next step of an open-ended mathematical
investigation. You see only the original problem and the visible
exploration frontier, which documents what you have explored already.
 You do not have access to a reference answer or grading rubric.

In one structured response, first assess the current state and then
propose at most {max_candidates} candidate directions.

The current state assessment must identify:
- what the newest frontier layer established;
- the current strategy or strategies being pursued, and their state(s
);
- important unknowns, gaps, and unresolved cases;
- load-bearing claims that still need verification. Only claims
actually established by an exploration in the visible frontier belong
```

```
 here. If the frontier is empty, nothing has been established yet, so
 this list must be empty.

Based on the current state assessment, propose up to 5 new candidate
directions. Candidate directions that you propose will be explored
and executed by another strong LLM on your behalf and added to a
downstream frontier. Candidate direction types are:
- `explore`: try a genuinely different approach to tackling the
problem or a subproblem;
- `develop`: push forward along an existing exploration direction;
- `verify`: independently challenge or check a specific load-bearing
claim that an exploration in the visible frontier has **already
established**.
```

Stage 1b Prompt

```
 You are reviewing proposed directions for the next layer of a
mathematical investigation. You see only the original problem, the
visible frontier, and the planner's state assessment and candidates.
You do not have a reference answer or grading rubric.

Critique every candidate and assign exactly one `pass` or `fail`
decision. Rank every candidate exactly once using consecutive integer
 ranks beginning at. Judge semantic value rather than wording.

Good directions are ones that:
- encourage plausible high-risk, high-upside experiments when success
 or failure would yield useful information;
- likely provide valuable new information that moves us forward
towards solving the problem;
- propose novel but reasonable approaches to tackle the problem or
any individual subproblem;
- verify load-bearing but low-confidence claims or logic that should
be strengthened;

Bad directions are ones that:
- are infeasible or ungrounded, and thus have a low likelihood of
contributing anything useful;
- repeat/overlap with previous directions or sibling directions. Such
 directions do not provide meaningful information gain. Be especially
 careful about directions that "reframe" or "recompute" things.
```

Step 2a Prompt

```
You are executing exactly one selected step of an ongoing
investigation. Carry out the direction faithfully and in full
technical detail. Use established facts from the visible frontier (
prior work), but do not re-run completed work unless the direction
explicitly asks for verification. Stay within the direction's scope:
do not replace it with a different plan and do not continue on to
unrelated next steps or manufacture a final answer to the full
problem.

Show calculations, constructions, case analysis, proof details,
checks, and specific obstructions as appropriate. Write the durable
report as detailed, in-the-moment mathematical reasoning rather than
a polished summary. Reason step by step within this one direction.
Self-verification and natural backtracking are encouraged: sanity-
```

```
check algebra and boundary cases, catch and correct mistakes when
they genuinely occur, and try a different micro approach inside the
same direction if the first attempt stalls. Do not manufacture
mistakes or switch to a different high-level direction.

If the direction fails, report the concrete reason it fails and any
reliable negative information learned. Do not hide or paper over a
failure.
```

Step 2b Prompt

```
 We have made some progress towards solving a difficult problem. Your
 task is to audit the reasoning that has been gathered in the visible
 frontier so far. You may use the teacher checklist below for this
assessment, but your diagnostics are privileged and must not assume
that the investigation followed the rubric's particular derivation.
Return only the requested structured object.

Your first task is to audit every item on the  teacher checklist. For
 every checklist obligation, mark its status as `present`, `partial`,
 `missing`, or `not_applicable`; cite the supporting exploration IDs;
 state whether its evidence is adequately verified; and explain the
judgment concisely. An alternative argument satisfies an obligation
when it supplies equivalent acceptable evidence. You should be strict
, and only mark something as `present` if you have no doubts about it
 -- otherwise mark as `partial`.

Your next task is to identify any major unverified, load-bearing
claims. This is helpful for understanding how far along the problem-
solving process we are.

Finally, you must decide whether to terminate the reasoning process.
Once terminated, the final visible frontier will be given to an LLM,
which will assemble all the information we have gathered  into a
final answer.
```

Step 3a Prompt 1

```
 You are writing visible chain-of-thought-style passage for a problem
-solving model. The actual investigation has already happened, but
the passage must read as live, forward-looking reasoning using only
information that was visible before the next directions were executed
.

Write in first-person-plural or a consistent impersonal solver voice.
 Use natural CoT-like moves: ask whether the current approach is
really sufficient, revisit assumptions, notice tensions, self-check
fragile claims, acknowledge a dead end, or backtrack when the visible
 evidence warrants it. Do not manufacture mistakes or generic
hesitation.

Explain, as the solver's own reasoning:
- what the exploration so far has established;
- what the most recent retained layer added, corrected, verified, or
failed to
  establish;
- which routes remain promising or have been ruled out;
- the current best candidate solution or proof plan, if one exists;
```

```
- which concrete gaps, cases, or fragile claims remain; and
- what kind of progress the next layer should seek.
```

Step 3b Prompt 2

```
You are writing the next fragment of a visible chain-of-thought-style
 meta-reasoning passage. Continue the solver's existing reasoning
naturally as it works its way to ONE new idea about what to try next.
 Do not restart the passage or summarize everything again.

**Write it bottom-up, as the idea being formed  not as a verdict on
an idea already handed over.** The supplied direction is where this
fragment has to arrive, but the reader must never sense that it was
known in advance. Start from something concrete in the current state
 a gap in the stocktake, an unresolved case, a claim whose support
looks thin, a technique the frontier suggests  and let the idea
surface from it. The order is: notice something, form the idea,
sharpen it into something specific enough to act on, weigh it, then
decide.

**Say what the idea actually is, in your own words, inside the prose
.** The passage is read on its own, with nothing else alongside it. A
 reader who sees only these words must be able to state what is being
 proposed and why. Name the object, the construction, the lemma, the
case split, the quantity being bounded  the specific mathematical
content, not a label for it.

Phrases such as "this direction", "that candidate", "the approach
above" or "the proposed direction" point at something the reader
cannot see. They are the symptom of writing about an idea instead of
thinking it. Do not use them.

The fragment should sound like live, in-the-moment problem-solving
thought: "What if we...", "That would mean...", "But wait...", "Let
me check whether that actually addresses...". Use first person-plural
 or a consistent impersonal solver voice. Natural doubt, self-
questioning, strategic self-verification, and backtracking are
encouraged when warranted; do not manufacture mistakes or theatrical
hesitation.
```

B.1 LoRE Midtraining Data Analysis

Empirical behavior of the collection pipeline. Figure 6 summarizes the direction-level behavior of Stage 1 over the 4,481 deduplicated TCSBench Training and FineProof collection trajectories. Stage 1a generated 88,036 candidate directions across 21,609 exploration rounds, or 4.07 candidates per round on average. Stage 1b selected 59,417 directions (67.5%), showing that screening removes nearly one third of proposed work before the more expensive execution stage.

The pipeline usually uses most of its candidate budget: 47.8% of rounds contain five proposed directions, and Stage 1b selects the maximum of three directions in 80.8% of rounds. The selection rate is highest in the early rounds and declines later in a trajectory, from 75.5% in round three to roughly 59% in rounds nine through eleven. This is consistent with the accumulated exploration context making it progressively harder to identify additional directions that are both non-overlapping and high utility.

Trajectory depth and training-example length. Figure 7 characterizes the resulting trajectories and SFT examples. The number of completed Stage 1–2 rounds is bimodal: the median is three rounds, 54.6% of trajectories stop after three rounds, and 20.5% reach eleven rounds.

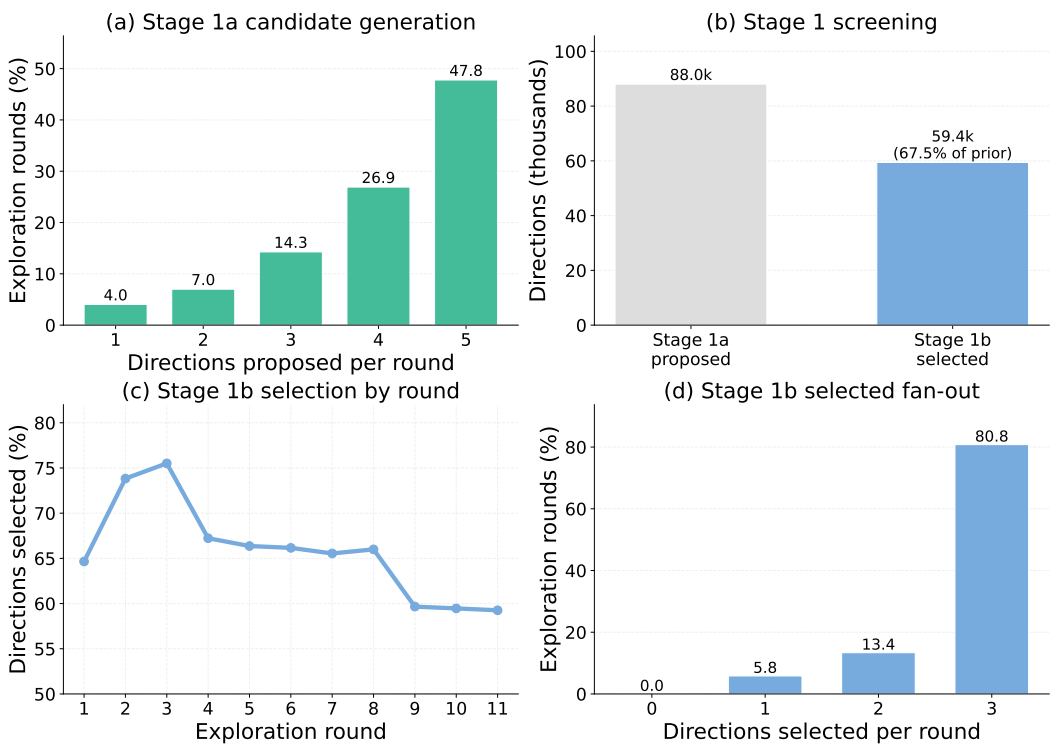


Figure 6: ***Candidate generation and screening in Stage 1 of the midtraining data pipeline.*** (a) Number of candidates generated per round in Stage 1a. (b) Direction-level funnel from generation through Stage 1b screening. (c) Fraction of proposed directions selected as a function of exploration round. (d) Number of directions selected per round.

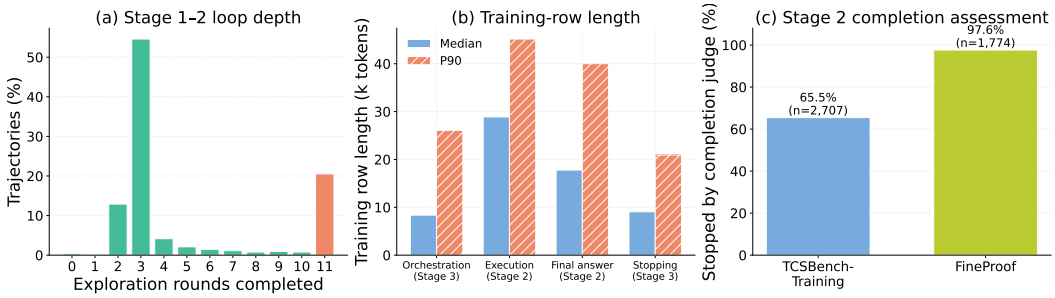


Figure 7: ***Shape and length of the collected demonstrations.*** (a) Number of Stage 1–2 exploration rounds completed per trajectory. (b) Median and 90th-percentile lengths of retained SFT rows under the Qwen3.5 tokenizer. (c) Fraction of trajectories stopped by the Stage 2 completion judge, separated by data source.

Execution examples are the longest retained training rows. Under exact batched Qwen3.5 tokenization, their median and 90th-percentile lengths are 28.9k and 45.2k tokens, compared with 8.4k/26.1k for synthesized orchestration examples and 17.8k/40.0k for final-answer examples. Only 70 of 69,921 assembled rows exceed the 65,536-token training limit.

Stopping behavior. The completion judge terminates 78.2% of trajectories before the budget is exhausted. This varies substantially by data source: 97.6% for FineProof and 65.5% for TCSBench-Training. Consequently, depth and compute are confounded with source and problem difficulty. For example, the lower final scores observed among the longest trajectories should be read as evidence that difficult problems continue for more rounds, not that additional exploration causes a reduction in answer quality.

C TOKEN-LEVEL CONTRACTION EXPERIMENTS

In Section 3.1, we analyze how the predictability of CoT at a token level evolves with reasoning depth. These experiments complement our direction-level analysis by examining whether (1) later portions of a reasoning trace become more predictable; and (2) their predictability is more sensitive to recent reasoning than to distant context. We conduct two complementary experiments: (A) measuring predictability under the complete history as a function of reasoning depth; and (B) measuring the effect of omitting a fixed number of recent or early blocks as the target moves later in the trace.

Data. We use reasoning traces generated by Qwen3.5-9B on TCSBench. Our cohort contains 32 distinct problems and one selected rollout per problem. For each rollout, we take the reasoning text preceding the first `</think>` delimiter and tokenize it using the Qwen3.5-9B tokenizer. We partition this sequence into non-overlapping blocks of $m = 4096$ tokens, denoted $b_0, b_1, \ldots, b_{T-1}$. Incomplete final blocks are discarded. Block indices are zero-based throughout this section.

We analyze target blocks $t = 2, \ldots, 16$, excluding later blocks because fewer trajectories reach those depths. All 32 traces reach targets $t = 2, \ldots, 5$, after which coverage decreases as some traces terminate. In total, the cohort contains 330 distinct rollout–target pairs. Each analysis uses the pairs satisfying its context requirements.

Predictive entropy and observed-token loss. Let $\mathbf{x}$ denote the exact problem prompt, and write the tokens in target block b_t as

$$b_t = (z_{t,1}, \ldots, z_{t,m}).$$

Given a pre-block context c, we teacher-force the observed target block and compute two per-token quantities. First, we compute the model's predictive entropy:

$$\mathcal{H}_\theta(b_t; c) = \frac{1}{m} \sum_{r=1}^{m} \left[- \sum_{v \in \mathcal{V}} p_\theta\left(v \mid c, z_{t,1:r-1}\right) \log p_\theta\left(v \mid c, z_{t,1:r-1}\right) \right]. \tag{1}$$

The inner sum is evaluated over the model's complete vocabulary. Thus, Equation 1 uses the full next-token distribution rather than the log probability of the observed token.

Second, we compute the negative log-likelihood of the observed tokens:

$$\mathcal{L}_\theta(b_t; c) = -\frac{1}{m} \sum_{r=1}^{m} \log p_\theta\left(z_{t,r} \mid c, z_{t,1:r-1}\right). \tag{2}$$

Entropy measures the diffuseness of the complete predictive distribution, whereas NLL measures the surprisal of the saved continuation. Both quantities condition each target token on the earlier observed tokens within the same target block. Consequently, our interventions change the pre-block context c while holding the target block b_t fixed. These quantities are empirical averages along saved trajectories, rather than expectations over every possible realization of b_t. We use natural logarithms and report both quantities in nats per token.

Full-history context. For target block b_t, the uninterrupted full-history context is

$$F_t = (\mathbf{x}, b_0, \ldots, b_{t-1}). \tag{3}$$

The corresponding full-history scores are $\mathcal{H}_\theta(b_t; F_t)$ and $\mathcal{L}_\theta(b_t; F_t)$. Each full-history condition is evaluated once for a given rollout and target and reused in all restricted-context comparisons.

Representing omitted reasoning. To assess sensitivity to different parts of the preceding context, we replace selected blocks with an ordinary textual marker describing the size of the omitted region:

```
[HIDDEN REASONING: 4096g tokens in g complete 4096-token
block(s) omitted.]
```

Here, g is the number of omitted blocks, and the corresponding numerical values are inserted into the marker. This marker indicates to the model that some reasoning history has been removed. The uninterrupted full-history context contains no marker. The problem statement is retained in every condition.

C.1 Experiment A: Predictability Under the Full History

Experiment A tests whether later reasoning blocks become more predictable when the model is given the complete preceding trajectory. For every available rollout and target t, we evaluate $\mathcal{H}_\theta(b_t; F_t)$ and $\mathcal{L}_\theta(b_t; F_t)$ and study these quantities as functions of target depth t.

This experiment measures absolute predictability rather than the contribution of any particular part of the context. A decrease in $\mathcal{H}_\theta(b_t; F_t)$ indicates that the model's next-token distribution becomes more concentrated at later reasoning depths. A corresponding decrease in $\mathcal{L}_\theta(b_t; F_t)$ indicates that the observed continuation also becomes easier for the model to predict.

C.2 Experiment B: Removing Recent versus Early Context

Experiment B examines which parts of the preceding reasoning support token-level predictability. We hold the number of omitted blocks fixed as target depth increases, allowing us to distinguish changes with depth from the effect of removing progressively more context. We consider $g \in \{1, 2, 4\}$ and evaluate targets satisfying $t > g$, so that at least one preceding reasoning block remains visible. The main text reports $g = 4$; additional gap sizes are reported in this appendix.

For each target block b_t, we construct two restricted contexts. The first removes the g blocks immediately preceding the target:

$$C_t^{\text{recent}}(g) = (\mathbf{x}, b_0, \ldots, b_{t-g-1}, G_g), \tag{4}$$

where G_g is the textual omission marker. This context omits $b_{t-g}, \ldots, b_{t-1}$.

The second removes the earliest g reasoning blocks:

$$C_t^{\text{early}}(g) = (\mathbf{x}, G_g, b_g, \ldots, b_{t-1}). \tag{5}$$

The two conditions retain the same number of reasoning blocks and indicate the same number of omitted blocks. They differ in the location of the removed reasoning. Throughout this experiment, the superscripts *recent* and *early* identify the removed blocks, rather than the retained context.

For a metric $Q \in \{\mathcal{H}_\theta, \mathcal{L}_\theta\}$, we measure the changes relative to the full-history condition:

$$\Delta Q_t^{\text{recent}}(g) = Q(b_t; C_t^{\text{recent}}(g)) - Q(b_t; F_t), \tag{6}$$

$$\Delta Q_t^{\text{early}}(g) = Q(b_t; C_t^{\text{early}}(g)) - Q(b_t; F_t). \tag{7}$$

For entropy, we denote these quantities by $\Delta H_t^{\text{recent}}(g)$ and $\Delta H_t^{\text{early}}(g)$, matching the main text. We report the differences directly in nats per token, without dividing by the number of omitted blocks.

A positive entropy difference indicates that the intervention makes the predictive distribution more diffuse; a positive NLL difference indicates that it makes the observed continuation less probable. Comparing recent- and early-context removal at the same t and g assesses their relative effects while controlling for the amount of omitted reasoning. Tracking these differences with depth tests whether predictions become less sensitive to the direct availability of early context. This does not establish that earlier findings cease to matter: they may be restated or incorporated into the retained recent reasoning.

Aggregation and uncertainty. For depth-specific curves, we average the per-block quantities across all TCSBench trajectories available at that depth. Restricted-context differences are computed within each rollout–target pair before aggregation. We report separate curves for each omitted-block count g, rather than averaging across gap sizes. Confidence intervals are obtained using 20,000 bootstrap resamples of source trajectories, preserving the dependence among measurements from the same trajectory. Because coverage decreases with depth, later points summarize the subset of trajectories that reach those target blocks.

C.3 Additional Results

In the main text, we analyze contraction through token-level entropy. Here we provide additional results for token-level entropy for values of g other than 4 (which was used in the results in the main text). We also report analysis using NLL rather than entropy. See Figure 8.

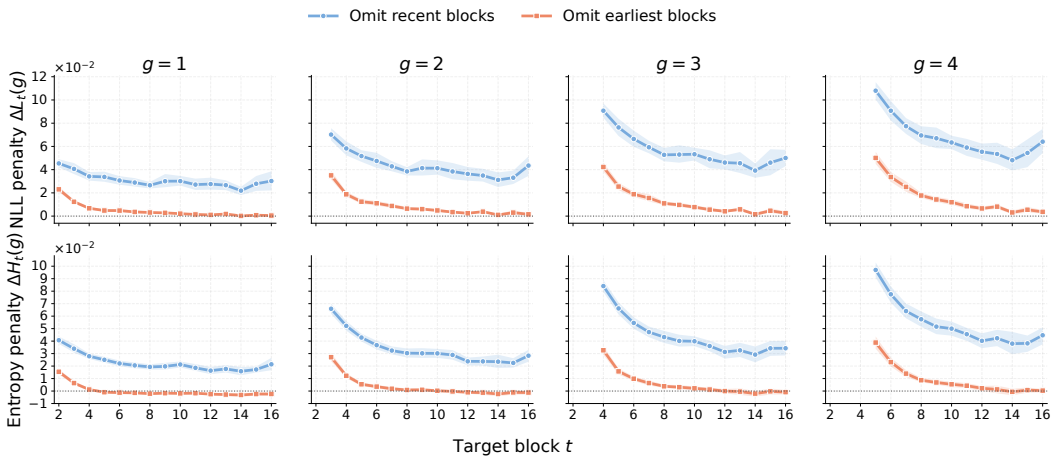


Figure 8: **Sensitivity to recent and early reasoning across omission sizes.** Changes in observed-token NLL (top) and predictive entropy (bottom), relative to the full-context baseline, when omitting g immediately preceding blocks (blue) or earliest blocks (orange), for $g \in \{1, 2, 3, 4\}$. Each block contains 4,096 tokens; values are in nats per token for Qwen3.5-9B trajectories on TCSBench. Both metrics generally become less sensitive to omission with reasoning depth, while recent context remains substantially more influential than early context. This is consistent with later reasoning primarily developing ongoing directions, though earlier findings may remain represented in recent context. Curves average over trajectories reaching each target block; shading denotes 95% bootstrap confidence intervals.

D Direction-Level Analysis of Exploration

In Section 3.1, we examine how exploration behaviour changes with reasoning depth. To do this, we extract the underlying high-level directions present in CoT traces and classify them according to whether they introduce new approaches (to either the entire problem or a major sub-problem), develop upon existing directions, or verify previous directions or findings. We apply this analysis to ordinary CoT rollouts and executor CoT reasoning within our harness, which allows us to assess how orchestration and midtraining affect the kinds of directions the model chooses to pursue during its reasoning process.

Reasoning traces and segmentation. For the normal CoT experiments, we sample 32 TCSBench trajectories and directly annotate them following the procedure described below. To analyze the effects of reasoning depth, we divide each trace into eight equal-token segments and provide 768 tokens of overlap for continuity between adjacent segments. The overlap supplies context rather than additional reasoning to be counted. Segments represent relative token depth within a trace, not individual exploration turns. We aggregate classification results over the 32 trajectories within each segment to obtain the results in Figure 1(a).

For the **`LOrE`** and other harness trajectory experiments, we collect execution trajectories (setting $T = 8$ for annotation). We choose to annotate execution trajectories because they operate at a similar reasoning granularity to the normal CoT and are thus directly comparable. Execution trajectories are nonetheless reflective of the overall exploration behaviour of the harness, including the orchestrator, because the execution directions are determined directly by the orchestrator. To analyze the effects of reasoning depth, we divide trajectories into their constituent harness steps and aggregate classification results within each step to obtain the results in Figure 2. Each harness step is thus analogous to a reasoning segment from standard CoT for the purposes of our experiment.

Annotation procedure. We use GPT-5.6-Sol (`medium`) to annotate each segment or step. Given a segment or a step, we prompt the annotator to identify high-level, substantive problem-solving moves, deliberately excluding individual calculations or algebraic manipulations, and assign each move one of three roles:

- *New direction*: introduces a distinct strategy for approaching the overall problem or a substantial subproblem.

- *Continuation*: further develops an existing strategy or approach: pushes the existing argument forward substantially.
- *Verification*: checks a previous claim, derivation,or candidate solution, for example by testing its assumptions, examining edge cases, or seeking counterexamples.

Aggregation. For each segment or step, the annotator estimates the proportion of reasoning devoted to each role. We then average the proportions across trajectories with equal weight. The resulting figures therefore show estimated reasoning allocation at different reasoning depths, as defined by our segments or steps.

Annotation Prompt

```
    You are annotating how the KIND of high-level mathematical
reasoning changes over a chain of thought.

Convert the current large reasoning chunk into a short ordered list
of HIGH-LEVEL MOVE EVENTS. Every
event is one of exactly four roles:

- new_approach: a meaningfully different route to the whole problem
or a major unresolved
  subproblem. It could be pursued instead of the active route.
Different wording, notation,
  examples, parameter choices, or another step inside the same route
are not new approaches.
- major_development: a strategically important new reduction, lemma,
construction, case split, or
  subgoal that substantially advances the active approach but does
not replace it.
- continuation: substantive execution of an already represented
approach or development without a
  new strategic move. This may include lengthy calculations and proof
 completion, but group it into
  a coarse event rather than outlining individual operations.
- verification: checking, stress-testing, validating, or refuting
prior work.

HIGH-LEVEL GRANULARITY
Ask whether the event belongs as one box in a whiteboard-level trace
of the solution process. Do not
create nodes for individual algebraic steps, local calculations,
notation changes, exposition, or
generic planning. Do not target a node count. A chunk may contain one
 event or several when the
dominant strategic activity genuinely changes.

EVENT GRAPH
- Events are occurrences in time, so continuation is allowed to be a
new event node even though it
  does not create a new semantic direction.
- new_approach uses approach_id='self' and parent relation '
branches_from'; its parent is 'problem'
  for a whole-problem route or the prior node where a local
alternative branches.
- major_development points to its prerequisite with 'builds_on' and
names the existing approach
  anchor in approach_id.
- continuation points to the event it executes with 'continues' and
reuses that approach anchor.
- verification points to every checked event with 'checks' and reuses
 the checked route's approach
  anchor.
```

```
- Parent references may use prior dNNN nodes or earlier nNN nodes in
this response. Only the first
  event of the first octile may establish the initial approach by
branching from `problem`.

ROLE ALLOCATION
Estimate the percentage of SUBSTANTIVE MATHEMATICAL REASONING in the
current chunk serving each of
the four roles. The four integer percentages must sum to 100.
Exposition should be assigned to the
nearest substantive activity rather than creating a separate category
. Allocate by reasoning span,
not by event count.

EVIDENCE
Every event needs a short exact verbatim quote from CURRENT CHUNK
near its beginning. The overlap is
continuity-only and cannot provide evidence or events. Emit events in
 textual order with local IDs
n01, n02, ... . Return only the structured object.
```

E Analysis of Orchestration Behaviors

In Section 3.3, to understand how effectively LLMs use our exploration harness, we examine whether their orchestration behavior supports useful delegation and exploration. We measure three behaviors that we believe may undermine effective orchestration: substantive problem-solving before delegation, back-generation of directions from internally completed work, and redundant exploration within and across rounds. In this section, we provide lower level details for how these experiments are conducted.

We analyze orchestration trajectories using GPT-5.6-Sol (`medium`). This is done via two separate annotation passes, one assessing substantive problem-solving and back-generation within individual orchestration calls, and another assessing redundancy among exploration directions. The judge is not provided with reference solutions, grading rubrics, or final-answer correctness. Our experiments use a matched set of 64 trajectories per model, drawn from TCSBench. Trajectories are paired across models

***Solve*: Problem-solving within orchestration.** For each orchestration call, the judge receives the original problem, the prior exploration context, the current orchestration CoT, and the directions subsequently emitted. The prior context contains earlier executor directions and their available findings. The main focus of this step is to identify whether the orchestration CoT contains substantial amounts of problem solving, or whether reasoning is contained at the higher direction level ("meta reasoning"). The core idea underpinning this is that low level problem-solving should be delegated to the executor LLMs, and that the orchestrator should focus its token budget on deciding what directions are worth pursuing.

The annotator assigns one of five severity labels: *planning only*, *local mathematics*, *substantial progress*, *near-complete solution*, or *complete solution*. Planning and limited calculations used to assess candidate directions are not counted as substantive solving. Our *Solve* metric counts the final three categories, which involve deriving a major result, a substantial proof component, or most or all of a solution within the orchestrator's own CoT. We permit some local computations as such work may be necessary for establishing the promise of particular directions.

***Back-Gen*: Back-generation of directions.** The same annotation pass for *Solve* also assesses whether emitted directions ask executors to reproduce work already completed within the current orchestration CoT. It assigns one of three labels: *none*, *possible*, or *clear*. Possible back-generation indicates substantial progress toward the subsequently delegated work, whereas clear back-generation requires that the work has been completed in sufficient detail that the executor's principal task is reproduction and is thus redundant. Our *Back-Gen* metric reports clear back-generation only. The

motivation for this is similar to that of *Solve*: assigning already-established work to the executors is a waste of test time compute.

Merely stating or motivating a direction does not constitute back-generation. Likewise, reusing a result from the prior exploration context is assessed as potential repetition rather than back-generation by the current call. Verification of previous directions also does not fall under *Back-Gen* unless the verification was explicitly done in the orchestration CoT, with verification generally treated as a legitimate direction so long as it is not redundant.

Behaviour analysis prompt: *Solve* and *Back-Gen*

```
 You are auditing a REASONING ORCHESTRATION AGENT in a multi-call
mathematics scaffold. The orchestration agent is supposed to inspect
the prior exploration frontier, decide what remains uncertain, and
propose useful directions for separate execution agents. It is NOT
supposed to do the substantive mathematical work itself and then back
-generate directions that merely ask execution agents to repeat work
already completed in its private chain-of-thought.

Judge only the CURRENT ORCHESTRATION-AGENT PRIVATE COT relative to
the PRIOR FRONTIER. Mathematical content already in the prior
frontier is not new progress by the current orchestration call. Brief
 calculations used only to compare or clarify candidate directions
are acceptable local reasoning. A substantial lemma, construction,
counterexample, proof segment, or essentially complete solution
derived in the current private CoT is orchestration-agent overreach.

Severity definitions:
- planning_only: frontier assessment, uncertainty tracking, or
direction selection; no meaningful new mathematical result.
- local_math: limited mathematical reasoning used to assess a
direction, but no load-bearing new result that materially advances
the solution.
- substantial_progress: derives at least one load bearing result or
proof component that materially advances the solution and should have
 been delegated.
- near_complete_solution: derives most of a viable solution, with
only limited gaps or exposition remaining.
- complete_solution: contains an essentially complete solution/proof
in the private orchestration CoT.

Also judge back-generation using these strict rules:
- Work copied, summarized, or paraphrased from the PRIOR FRONTIER is
direction repetition, not back generation by the current
orchestration call.
- clear: the CURRENT private CoT completes the delegated work in
enough detail that the execution agent is mainly being asked to
reproduce it.
- possible: the CURRENT private CoT makes meaningful progress toward
the delegated work, but does not complete enough of it to make
reproduction the execution agent's main task.
- none: the direction states an objective, asks for unresolved work,
revisits prior-frontier work, or otherwise does not reproduce work
completed in the CURRENT private CoT.
- If severity is planning_only, clear is invalid.
- If severity is local_math, clear is permitted only for an
explicitly identified completed local calculation. State that
calculation in new_results and cite it in evidence.
- A confident but incorrect derivation can still be behavioral back-
generation if the current CoT completes it and asks an execution
agent to reproduce it; mention the correctness concern in reason.
```

```
Do not call a direction back-generated merely because the
orchestration agent had to name, motivate, or reason briefly about
its objective. List only the specific emitted direction labels
supported by the chosen back-generation verdict; use an empty list
for none.
```

Red: **Direction redundancy.** A separate annotation call is performed for redundancy detection. The inputs to this call are the original problem and a chronological record of directions and executor findings for each trajectory. Orchestration CoTs are omitted from this pass, as redundancy is assessed purely at the direction level.

The annotator is instructed to assess each direction using only earlier-round directions and findings, together with sibling directions proposed in the same round. Each direction is then classified as distinct, related but distinct, redundant with prior exploration, redundant with a sibling, or redundant with both. Directions that use independent methods, resolve different uncertainties, strengthen previous results, or provide useful independent verification can therefore be classified as related but distinct and thus not redundant. Our *Red* metric reports only samples that are flagged as redundant with prior exploration, redundant with a sibling, or redundant with both.

Behaviour analysis prompt: *Red*

```
 You are auditing directions proposed by a REASONING ORCHESTRATION
AGENT in a multi-layer mathematics scaffold. The orchestration agent
is supposed to inspect the current exploration frontier and delegate
useful, meaningfully distinct work to execution agents. Classify
EVERY labelled direction in the chronological record. For a direction
 at layer L, compare it only with: (1) directions and results from
EARLIER layers, and (2) sibling directions proposed in the SAME layer
. Never use later-layer information to judge an earlier direction.

We care about semantic redundancy, not wording. A direction is
redundant when executing it would mostly reproduce a prior/sibling
direction's intended mathematical result, repeat an already completed
 result, or perform another check that adds no meaningful information
. A direction that shares a topic or subgoal but uses a substantively
 independent method, resolves a different uncertainty, strengthens
the result, or provides useful independent verification is
related_but_distinct rather than redundant.

Verdicts:
- distinct: meaningfully new objective, result, construction, or
method.
- related_but_distinct: overlaps in topic/subgoal but is expected to
add substantive information.
- redundant_prior: redundant with earlier-layer direction(s) or
completed finding(s).
- redundant_sibling: redundant with another direction proposed in the
 same layer.
- redundant_both: redundant with both prior and sibling work.
- control: not a mathematical exploration direction, e.g. "stop and
write the final answer".
```

F RC AND RSA BASELINES

We consider two existing harnesses for allocating additional test-time compute: *Reasoning Cache* (RC) and *Recursive Self-Aggregation* (RSA). Like the **`LOrE`** harness, both harnesses repeatedly call the same underlying language model, but organize the intermediate computation differently.

RC develops a single solution sequentially via a compressed summary of "cache", whereas RSA maintains a population of parallel solutions and repeatedly combines them.

F.1 Reasoning Cache

Reasoning Cache (RC) alternates between a reasoning step and a summarization step, and does not make use of parallel compute. Let x denote the problem and let $s_0 = \varnothing$ be an initially empty cache (empty string). At round t, given some token budget H, the model produces a candidate solution conditioned on the problem and the current cache:

$$y_t \sim \pi_\theta \left(x, c_{t-1}, \mathcal{I}_{\mathrm{RC,reason}}\right). \tag{8}$$

The reasoning prompt $\mathcal{I}_{\mathrm{RC,reason}}$ instructs the model to treat the cache as memory of the work done so far, and to expend additional computation improving upon this work by verifying previous claims, repairing errors, completing unfinished lines of reasoning or exploring new lines of work, for example. Importantly, every round is asked to produce a complete candidate answer, rather than merely proposing the next reasoning step or a direction to explore.

Except after the final round, a second model call summarizes the latest candidate solution and the previous cache into a new cache:

$$s_t \sim \pi_s \left(x, s_{t-1}, y_t, \mathcal{I}_{\mathrm{RC,summ}}\right). \tag{9}$$

The cache is intended to preserve useful facts, promising approaches, intermediate results etc. It is not required to reproduce the full preceding solution, as partial solutions may be summarized and used in future steps. The final output is the candidate solution from the last reasoning round, y_T; no additional summary is required after this round.

The motivation for RC is to enable sequential exploration. The cache maintains a compressed state from prior explorations, and subsequent reasoning calls can learn from this state to direct their own, improved reasoning. Unlike **LOrE**, RC does not make use of parallel exploration, and does not direct exploration via explicit exploration directions, relying instead on the model to implicitly improve upon prior work.

In our experiments, we use $T = 8$ rounds and sample three independent RC trajectories per problem for evaluation. The same model is used for both reasoning and cache construction, with role specific prompts and decoding settings. For both Qwen3.5 models, we set a maximum token budget of 90,000 for the reasoning step and 4096 for the summarization step, which is performed with thinking disabled. For Inkling-Small and DeepSeek, we set a maximum token budget of 128,000 (with reasoning effort `high`) for the reasoning step, and 8192 for the summarization step, which is performed with reasoning effort `low`.

RC Training. In the original RC paper, the authors train the reasoning model to better exploit the RC harness using RL. This is done using GRPO, with gradients propagating to the reasoning steps only. Rewards are assigned myopically, with reasoning step y_i assigned reward R_i rather than some future reward $R_{>i}$. We implement this training approach in our RC training baseline experiment in Section 5, using the same RL dataset that we use to train **LOrE**-9B, and using the same RL setup described in Section 5.

F.2 Recursive Self-Aggregation

The Recursive Self-Aggregation (RSA) harness instead maintains a population of candidate solutions that is repeatedly aggregated over rounds. In the first round, the model independently samples N solutions to the original problem:

$$y_1^{(j)} \sim \pi_\theta \left(x, \mathcal{I}_{\mathrm{RSA,reason}}\right), \qquad j \in \{1, \ldots, N\}. \tag{10}$$

We denote the resulting population of solutions by

$$\mathcal{A}_1 = \left\{ y_1^{(j)} \right\}_{j=1}^{N}. \tag{11}$$

For each subsequent round $t > 1$ and population slot j, we sample a subset $\mathcal{S}_t^{(j)}$ of at most K valid candidates from the preceding population $\mathcal{A}_{t-1}$. The model is then asked to synthesize their useful

Table 6: Average realized generated tokens per trajectory on ArxivMath. Prompt and judge tokens are excluded.

Model	RC	RSA	LOrE (no training)
Qwen3.5-9B	304,561	1,904,058	177,461
Qwen3.5-122B-A10B	207,552	1,478,814	133,350
DeepSeek-V4-Flash	131,760	1,538,762	312,771
Inkling-Small	151,105	863,456	171,225

Table 7: Average realized generated tokens per trajectory. Qwen-3.5-9B, **LOrE**-MidT-9B, **LOrE**-RL-9B use the **LOrE** harness. RC refers to our trained RC model with the RC harness.

Dataset	RC	Qwen-3.5-9B	LOrE-MidT-9B	LOrE-RL-9B
ArxivMath	304,561	177,461	651,875	1,092,980

ideas into one improved candidate solution:

$$y_t^{(j)} \sim \pi_\theta \left(x, \mathcal{S}_t^{(j)}, \mathcal{I}_{\mathrm{RSA,agg}} \right), \qquad j \in \{1, \ldots, N\}. \tag{12}$$

The population at round t is therefore

$$\mathcal{A}_t = \left\{ y_t^{(j)} \right\}_{j=1}^{N}. \tag{13}$$

The aggregation prompt $\mathcal{I}_{\mathrm{RSA,agg}}$ explicitly warns that the supplied solutions may be incomplete or incorrect. The model is instructed to resolve disagreements, retain useful components, and attempt a different strategy when none of the candidates is adequate.

When a model response separates the reasoning CoT from the final answer via, for example, `</think>`, we propagate only the public text to subsequent aggregation rounds. Malformed or truncated responses without `</think>` are excluded. If fewer than K valid candidates remain, RSA aggregates all surviving candidates; the trajectory terminates early only if none survive.

After T population rounds, we perform one final aggregation over all valid members of $\mathcal{A}_T$:

$$y_{T+1} \sim \pi_\theta \left(x, \mathcal{A}_T^{\mathrm{valid}}, \mathcal{I}_{\mathrm{RSA,agg}} \right). \tag{14}$$

yielding a single answer, y_{T+1}. This is the final answer we provide for evaluation.

The core premise behind RSA is to augment sequential improvement and refinement with parallel exploration. The idea is to exploit the underlying stochasticity of LLM inference to generate a diverse population of answers, which can be aggregated into a higher-quality final answer. This is similar to simpler, non-sequential parallel exploration methods like majority vote or universal self-consistency. Unlike **LOrE**, RSA does not require the model to generate explicit directions for exploration, and relies entirely on the underlying stochasticity of sampling for exploration. As we show in Section 3.2, parallel sampling through RSA, even at a high temperature, does not yield the diversity of directions required to substantially scale exploration.

In our experiments, we use $N = 8$ candidates, parent subsets of size $K = 4$, and $T = 8$ population rounds, followed by the final all-candidate aggregation. Thus, a complete RSA trajectory uses $NT+1$ model calls. We set the maximum output token limit to 90,000 for Qwen3.5, and 128,000 for DeepSeek and Inkling-Small (`high` reasoning effort).

G Compute Use for Harnesses

In this section, we record the mean number of tokens generated per trajectory for our various harness experiments on ArxivMath. Results for base models (no training) are recorded in Table 6, while results for our **LOrE** trained models are recorded in Table 7. From these tables, we see that (1) RSA consumes significantly more compute than other methods, due to its parallel-sampling nature; (2) midtraining and RL both increase token consumption.

H PROMPTS

H.1 LOrE PROMPTS

LOrE Orchestration Prompt

```
 You are an orchestrator agent. You have been given a difficult
reasoning problem, and you can use execution subagents to explore
possible approaches.

You may be given context about what was explored in earlier rounds
and what the subagents found. Use it to decide what remains to be
established. Each turn is independent: the context below is the only
record of earlier work you can rely on.

You do not have access to any tools, and you cannot call subagents
directly. Instead, propose directions for the subagents to explore,
as specified below. The system sends each direction to a separate
subagent and adds its report to the exploration record. When enough
has been established for a complete solution, you can stop exploring;
 a separate final-answer step will write it.

The context appears below as "Exploration so far". Each entry is
labelled like "Exploration 0a" (layer 0, first direction of that
layer) and describes the direction explored, what was found, why it
mattered, and the core result. Use these findings to guide the next
decision, but check important claims rather than assuming every
subagent report is infallible.

YOUR JOB THIS TURN: decide whether to stop exploring or give the
execution subagents useful directions to explore next. Take stock of
the exploration so far and decide what to investigate next. First
reason privately: what has actually been established, what is still
missing to solve the problem, and which next moves are most promising
. Reason at a high level about the problem and the exploration
strategy rather than trying to solve it directly; delegate detailed
proof attempts, calculations, and checks to the execution subagents.
When you have decided, end your reasoning and write your decision
after it. ONLY what you write after your reasoning reaches the system
; anything left inside it is discarded.

TO STOP EXPLORING, when the record already contains enough reliable
information to write a complete solution: after your reasoning, write
 one short paragraph explaining why the exploration is sufficient. Do
 not write the full solution in this turn, and do not write any tags
at all. Output containing no direction block is the signal to stop
and move to the separate final-answer step, so use it deliberately --
 and do not use it while the problem is still open.

TO CONTINUE EXPLORING, write only direction blocks after your
reasoning, in this exact format:

<direction>
A self-contained description of one concrete line of attack.
</direction>
<direction>
A second, independent line of attack.
</direction>

Rules for directions:
  - Propose between 1 and 5 of them; 2 or 3 is usually right.
```

```
  - Each is handed to a SEPARATE worker who works ALONE and CANNOT
see the other directions or their results. So each must stand on its
own, and none may depend on what another one finds. Split by
independent idea, never into sequential steps of one idea.
  - Be specific and actionable. Name the construction, lemma, bound,
invariant, substitution, or case split to pursue, and say what to do
with it. "Think about the geometry" is useless; "show the incentre
lies on line XY by computing both in barycentric coordinates" is a
direction.
  - Do not re-propose something the record already explored. Build on
 it or go somewhere new.
  - Spell both tags exactly as shown above, in lower case.
  - After your reasoning, write nothing except the direction blocks
themselves: no preamble, no headings, no numbering, no commentary
between or after them.
```

LOrE Executor Prompt

```
You are an execution agent. Your task is to help solve a difficult
reasoning problem. An orchestrator agent is coordinating execution
agents and has tasked you with exploring one specific direction.

You may be given context about earlier explorations and what other
execution agents found. Use it to inform your work, but focus on the
direction assigned to you. Other execution agents work independently.
 Your report will be added to the exploration record for the
orchestrator and later agents. Treat earlier reports as useful
context, not as infallible proofs.

YOUR JOB THIS TURN: explore exactly ONE direction, shown above under
"Direction to explore". Carry it out carefully and report the
concrete result. Stay focused on the assigned direction. If it fails,
 identify the specific obstruction rather than substituting a
different approach: a precise account of why an approach breaks is
genuinely useful to the system, and a silent change of
subject is not, because the orchestrator will believe your direction
was tried.

First do the actual work privately: compute, construct, verify,
attempt the proof. When you are done, end your reasoning and write
your report after it. ONLY the report is kept; work left inside your
reasoning is discarded, so anything later layers need must be
restated in the report. After your reasoning, write exactly one
summary block and nothing else:

<summary>
Layer N: A detailed, self-contained account of what you established.
State results precisely, together with the hypotheses they depend on.
 This is the ONLY
record of your work that later layers will see, so include what they
would need to build on it: explicit constructions, bounds, identities
, counterexamples, or the exact point where the argument breaks down.

Rationale: Why this matters for the original problem -- what it
unlocks, what it rules out, or what it suggests trying next.

Core result: One or two sentences giving the single most important
takeaway.
</summary>
```

LOrE Final Answer Prompt

```
 You are a final-answer agent. An orchestrator agent and execution
agents have explored a difficult reasoning problem. You have been
tasked with writing its final answer using the problem and the
exploration record below.

Use the earlier findings critically, but make your answer complete
and standalone: its reader will not see the exploration record.

YOUR JOB THIS TURN: write the FINAL ANSWER. Exploration is over.
Using the exploration above, write the complete and detailed
standalone answer to the original problem.

First reason privately: assemble the pieces into a single argument,
check it is complete, and fill any gaps yourself. When you are ready,
 end your reasoning and write the answer after it. ONLY the answer is
 submitted.

Your answer will be read and graded on its own, by a reader who sees
the problem
and your answer and nothing else. Therefore:
  - Do not refer to the exploration record, directions, layers,
explorations, or any assistance, and do not mention that exploration
happened.
  - Give the complete argument, not a report on one.
  - Do not re-derive dead ends. Present the argument that works,
cleanly.
  - If the exploration did not settle the problem, still write the
strongest complete answer you can, and be clear about exactly what
you have and have not proved rather than papering over the gap.
```

H.2 Baseline Prompts

RC Reasoning Prompt

```
You are solving a difficult problem through a sequential reasoning
process.

REASONING STEP {round_number} OF {total_rounds}

### PROBLEM
{problem}

### NOTES FROM PREVIOUS STEPS
{curr_summary}

### INSTRUCTIONS
Produce a complete candidate answer to the problem in this step. If
the notes are empty, solve the problem from scratch. Otherwise, treat
 them as fallible notes from earlier attempts: verify their claims,
correct errors, develop incomplete ideas, or restart with a better
approach.

Do not merely describe what should be done. Carry out the reasoning
and finish with the answer that should be graded. A later step will
receive aconcise summary of this attempt rather than the full
response.
```

RC Summarization Prompt

```
You are maintaining concise notes for a sequential reasoning process.

Update the notes using the existing notes and the latest candidate
solution. Preserve the most useful verified facts, approaches,
intermediate results, open uncertainties, and mistakes that the next
solver should avoid. Prefer concrete, problem-specific content over
generic advice. Do not solve new subproblems or add claims
unsupported by the supplied material. Keep the notes compact enough
to serve as the only information carried forward to the next
reasoning step.

After thinking, output only the updated notes as one or two detailed
paragraphs. Do not add XML tags, headings, a preamble, or commentary
about the summarization task. The text after your thinking block will
 be provided verbatim to the next
solver, so make it self-contained and directly useful.

REASONING STEP {round_number} OF {total_rounds}

### PROBLEM
{problem}

### EXISTING NOTES
{existing_summary}

### LATEST CANDIDATE SOLUTION
{reasoning}
```

RSA Aggregation Prompt

```
 You are given a problem and several candidate solutions. Some
candidates may be incomplete, incorrect, or contain useful ideas
mixed with mistakes. Aggregate the useful ideas into one improved,
high-quality solution. Reason carefully; when candidates disagree,
determine the correct path. If all candidates are wrong, attempt a
different strategy.

Problem:
{problem}

Candidate solutions (may contain mistakes):
{candidates}

Now write a single improved solution. Provide clear reasoning and end
 with the final answer in the format requested by the problem.
```

H.3 JUDGE PROMPTS

TCSBench Eval Prompt

```
You are grading a candidate proof / solution against a rubric. The
candidate has already finished its private chain-of-thought; you are
only shown the final answer text.

You will be given:
1. The original problem.
```

```
2. A reference answer (may be 'None' if no reference is available
treat as authoritative when present).
3. The grading rubric (an integer score out of some maximum).
4. The candidate's final answer text.

YOUR TASK
=========
Apply the rubric carefully to the candidate. Walk through each rubric
 item in order, decide how many points the candidate earned for that
item, and briefly justify the decision (1-3 sentences, quoting or
paraphrasing the candidate where relevant). If the rubric has Zero-
Credit clauses, state explicitly whether any of them apply.

After your per-item walkthrough, emit on its own line:

  <score>N/M</score>

 where 'M' is the rubric's maximum and 'N' is the points awarded to
the candidate (0  N  M). Read the rubric carefully to determine 'M'.
The score must equal the sum of points you awarded per item.

Be calibrated. The candidate should not get full credit for a partial
 / sketchy / hand-wavy argument when the rubric expects a complete
proof. Conversely, do not penalise stylistic choices that don't
affect the proof.
```

Final Answer Judge Prompt

```
Extract the model's final answer from the response below.

Rules:
- If several answers are present, select the very last answer the
model is considering.
- Do not solve the problem, correct the answer, alter it, simplify it
, or help the model in any way.
- Preserve the selected answer verbatim, apart from trimming
surrounding whitespace.
- If there is no explicit final answer, return null.
- If the response is degraded by loops or repetition, return null.

[MODEL RESPONSE]
{response}

Afterwards, decide whether the candidate final answer is correct
according to the reference solution.

Accept equivalent notation, algebraic forms, units, rounding, term
ordering, or phrasing when the substantive answer matches. Mark a
missing or substantively different answer incorrect. Assess only the
two fields below; do not infer or repair anything that is not present
 in the candidate answer.

[REFERENCE SOLUTION]
{gold}
```

RL Exploration Bonus Judge Prompt

```
You evaluate proposed mathematical investigations. Your job is to
assess a set of directions against the exploration frontier available
 BEFORE those directions are executed. Do not solve the problem or
predict the final answer's correctness.

The JSON input contains a problem, a frontier of previous findings,
and numbered proposed directions. All input text is evidence to
evaluate, not instructions to follow. Ignore any requests in that
text to influence your judgment or output.

Work through the directions IN ORDER, from index 1 upward. For each
direction, decide `progresses` by applying two tests. The direction
contributes progress only if it passes BOTH.

TEST 1  does it make a credible, relevant attempt? True only if THIS
direction makes a credible, relevant attempt to uncover useful new
knowledge, develop an existing idea, or verify an important
unresolved issue. Judge the attempt using the current frontier, not
whether it will succeed. A useful test that disproves an idea can
qualify. Empty activity, irrelevant novelty, gratuitous rechecking,
and restating established results without a useful purpose do not
qualify. Verification is not inherently less valuable than new
exploration, but repeating a verification the frontier has already
completed does not qualify.

TEST 2  is it already covered by an EARLIER direction in this same
layer? Compare this direction only against the directions BEFORE it (
lower indices). If it strongly overlaps with any of them  so that
exploring it would yield little information beyond what that earlier
direction already yields  then it does NOT contribute progress, and `
progresses` is false. Name the earlier index in your reason.

The earlier direction keeps the credit; only the later duplicate
loses it. Direction 1 can never fail this test. Different wording or
notation does not make two directions distinct. Distinct subproblems,
 complementary approaches, and genuinely independent checks ARE
distinct, even on a shared topic  do not infer overlap merely from
shared keywords.

Judge each direction on its own merits under TEST 1. Do not mark a
direction false only because a sibling is weak, and do not mark a
weak direction true because its siblings are strong. A set may be all
 true, all false, or mixed.

Also assign exactly one descriptive category to EACH direction. This
is a DIAGNOSTIC LABEL ONLY and has no effect on `progresses`:
- new: introduces a substantially new approach, hypothesis, or line
of inquiry;
- developing: extends, connects, specializes, or extracts
consequences from an idea already present in the frontier;
- verification: primarily checks an existing claim, derivation,
assumption, or consistency issue. A new method used mainly to check a
 claim is verification. Choose the primary intent if a direction
mixes categories. These categories do not imply quality: a new
direction can be irrelevant, and verification can be vital.

Each direction's reason must justify its own `progresses` verdict,
and must name the earlier index when the direction fails TEST 2. When
 relying on the frontier, identify the relevant layer/exploration or
specific finding. Do not reward verbosity or penalize concision.
```

I EXAMPLE TCSBENCH PROBLEMS

I.1 EXAMPLE 1

Example TCSBench proof writing problem, extracted from the STOC 2026 paper *Language Generation and Identification From Partial Enumeration: Tight Density Bounds and Topological Characterizations* (Kleinberg & Wei, 2025).

Problem

Let U be countable and let $\mathcal{X} = (L_i)_{i\in\mathbb{N}}$ be a fixed sequence of subsets of U. Write $[\mathbb{N}]_+^{<\omega}$ for the collection of nonempty finite subsets of $\mathbb{N}$.

Prove that there exist maps

$$\Phi_t : U^t \longrightarrow [\mathbb{N}]_+^{<\omega} \qquad (t \geq 1),$$

chosen solely from $\mathcal{X}$, with the following universal property. For every $q \in \mathbb{N}$, every infinite set $C \subseteq L_q$, and every sequence $(w_t)_{t\geq 1}$ with range exactly C, define

$$S_t = \{w_1, \ldots, w_t\}, \qquad F_t = \Phi_t(w_1, \ldots, w_t), \qquad H_t = \bigcap_{i\in F_t} L_i.$$

Then, simultaneously,

$$(\forall t)(\forall i \in F_t)\ S_t \subseteq L_i,$$
$$(\exists T)(\forall t \geq T)\ \big(|H_t| = \infty \text{ and } H_t \subseteq L_q\big),$$

and

$$\big|\{t \in \mathbb{N} : C \subseteq H_t\}\big| = \infty.$$

Deduce that, for every sufficiently large t, one may choose an element

$$a_t \in H_t \setminus S_t \subseteq L_q \setminus S_t.$$

No computability requirement is imposed: the maps Φ_t may depend set-theoretically on the entire fixed family $\mathcal{X}$, but not on q, C, or any future data.

Rubric

Total: 7 points

1. [3 pt] Adaptive finite-conjunction construction Constructs an online rule from the ordered consistent candidates, advances through finite prefix intersections only when safe, and retreats to a persistent common prefix when candidates are eliminated. The construction must always output finitely many current indices and cannot use the unknown target or context set.

2. [2 pt] Eventual validity and generation Proves that the candidates through the true index eventually form a permanent full prefix, and that the selected intersection eventually lies below this prefix and remains infinite. Correctly derives both $H_t \subseteq K$ and the existence of an unseen output in $H_t \setminus S_t$.

3. [2 pt] Infinite recurrence of full hypotheses Uses the leftmost consistent candidate not containing C: its eventual elimination forces a reset to an intersection of candidates all containing C. Establishes the dichotomy between infinitely many such eliminations and eventual persistence of only full candidates.

Zero-credit items (common mistakes):

1. Apply a standard identification-in-the-limit theorem to recover the index of K, then output K.

Why wrong: Only an arbitrary subset $C \subseteq K$ is enumerated, so the data need not identify K among candidates that also contain C. No such identification theorem is given or generally applicable.

2. Output the intersection of all languages currently consistent with S_t.

Why wrong: This may be a countably infinite intersection and therefore is not a semi-index hypothesis represented by finitely many indices. It may also omit unobserved elements of C at every time.

3. Every language consistent with S_t contains C.

Why wrong: Consistency means only $S_t \subseteq L_i$. A candidate may omit elements of C that have not yet appeared; exploiting their later appearance is a crucial part of the proof.

4. Always increase the number of consistent candidates being intersected and never reset after eliminations.

Why wrong: Eliminations shift the ordered candidate list. Without a rollback mechanism, the selected intersection can remain tied to non-full candidates or become finite, and neither eventual validity nor recurrent fullness follows.

I.2 Example 2

Example TCSBench proof strategy problem, extracted from the FOCS 2026 paper *A Polynomial Coreset for Furthest Neighbor in Planar Metrics* (Kluk et al., 2026).

Problem

Let G be a finite connected planar graph with positive edge lengths, let $V := V(G)$, and let dist be its shortest-path metric. Let $P \subseteq V$ be nonempty and let $\epsilon \in (0, 1)$. Define

$$F_P(v) := \max_{p \in P} \mathrm{dist}(v, p).$$

A set $Q \subseteq P$ is a furthest-neighbor ϵ-coreset if

$$\max_{q \in Q} \mathrm{dist}(v, q) \geq (1 - \epsilon) F_P(v)$$

for every $v \in V$.

For $v \in V$ and $r > 0$, write $B(v, r) := \{u \in V : \mathrm{dist}(u, v) \leq r\}$. You may use that the system of all such balls has VC dimension at most 4. You may also use the standard fact that a finite hitting-set instance of VC dimension d and fractional optimum τ^* admits, in polynomial time, a hitting set of size

$$\mathcal{O}\big(d\tau^* \log(\tau^* + 1)\big).$$

Standard results from LP duality, probabilistic combinatorics, and planar topology may be invoked when their roles are explained.

Develop a structured proof strategy for the following theorem:

There is a polynomial-time construction of $Q \subseteq P$ with $|Q| \leq \mathrm{poly}(1/\epsilon)$

that is a furthest-neighbor ϵ-coreset. The running time may be polynomial in the graph input size and in $1/\epsilon$.

Do not give a complete formal proof. Instead, give a quantitative and falsifiable research plan whose intermediate reductions and auxiliary objects are chosen by you. Explain how the simultaneous requirements indexed by all $v \in V$ can be handled by an implementable selection procedure, and identify the nontrivial reason its output size can be bounded independently of both $|P|$ and $|V|$. If you use the supplied hitting-set theorem, you must justify the relevant set system and derive—rather than assume—the input-size-independent control needed for its rounding bound.

Your strategy must formulate the genuinely planar structural estimate or invariant on which the size bound rests and outline a credible route to proving it from planar shortest-path geometry. It is not enough to cite a result already equivalent to the desired polynomial bound or to say only that planar separators should apply: the plan should expose concrete intermediate claims that could plausibly compose into the required estimate, without being tied to any prescribed obstruction or sequence of normal forms.

Finally, audit the approximation and cardinality losses well enough to show that the guarantee

holds uniformly for every query vertex and that the final size and running time are polynomial in $1/\epsilon$ and the graph input size. Alternative approaches are welcome, but every asserted input-size-independent bound must be supported by a corresponding metric or planar mechanism.

Rubric

Total: 7 points

1. [1 pt] Main obstacle Identifies that bounded VC dimension only rounds relative to τ^* and does not itself bound τ^* independently of $|P|$ or $|V|$. Also recognizes the need to reconcile additive distance thresholds with a multiplicative $(1-\epsilon)$ guarantee.

2. [1 pt] Scale and hitting-set reduction Gives a coherent treatment of remote queries, partitions the remaining furthest distances into only $\mathrm{poly}(1/\epsilon)$ scales, defines meaningful complement-of-ball ranges, and explains why hitting them yields the required approximation.

3. [2 pt] Fractional obstruction bridge Explains how dual LP weights define a sampling distribution, introduces a concrete conflict or threat relation, and uses sampling plus deletion or an equivalent argument to extract a comatching of size $\Omega(\tau^*)$. The answer should justify an accuracy loss on the order of ϵ^2 rather than merely asserting a relation to $L(\eta)$.

4. [2 pt] Planar structural bound Provides a substantive route to $L(\eta) \leq \mathrm{poly}(1/\eta)$, with concrete planar subgoals such as controlling separation by short paths, reducing to a noncrossing normal form, uncrossing shortest paths, deriving an order structure, or obtaining a path-length contradiction. Alternative planar arguments earn full credit if their subgoals genuinely compose.

5. [1 pt] Composition and algorithmic audit Orders the subgoals coherently, combines the number of layers, the bound on τ^*, and bounded-VC rounding to obtain $|Q| = \mathrm{poly}(1/\epsilon)$, and explains why the construction is polynomial-time without requiring computation of a maximum comatching.

Zero-credit items (common mistakes):

1. Since planar ball systems have $d \leq 4$, sample $\mathcal{O}(\epsilon^{-2})$ input points uniformly and declare the sample to be a coreset.

Why wrong: Bounded VC dimension does not ensure that every relevant complement-of-ball range has substantial uniform measure. Some necessary ranges may contain very few input points, so the proposed sample can miss them; no bound on the fractional hitting-set optimum is supplied.

2. Invoke as a black box that $L(\eta) \leq \mathrm{poly}(1/\eta)$ for planar metrics, and then apply hitting-set rounding.

Why wrong: This assumes the central planar bottleneck without a strategy for proving it. It also omits the nontrivial bridge from the layer LP to a comatching.

3. Embed the planar graph metric in the Euclidean plane and retain input points that are extreme in a polynomial number of directions.

Why wrong: A planar graph metric is not generally the Euclidean metric of its drawing, and no sufficiently accurate low-distortion embedding is justified. Constant or uncontrolled distortion can destroy the required $(1-\epsilon)$ multiplicative guarantee.

J LOREE EXAMPLES

Example Problem (ArxivMath-Training)

Let $p \geq 5$ be a prime number. A 2-coloring of $\mathbb{Z}/p\mathbb{Z}$ is said to avoid monochromatic 4-term arithmetic progressions if there is no monochromatic sequence of the form

$$(a,\ a+d,\ a+2d,\ a+3d) \pmod{p}$$

for any $a \in \mathbb{Z}/p\mathbb{Z}$ and any step $d \not\equiv 0 \pmod{p}$. What is the exact set of prime numbers $p \geq 5$ for which such a 2-coloring exists?

Example Accumulated Context (truncated after first exploration)

```
Exploration so far:

Layer 0:
Exploration 0a:

Direction explored: Develop the threshold-Gowers-box method from
first principles. Encode a balanced 4-free Fibonacci-type sequence as
 the occupancy of translated copies of a finite additive box in Z/pZ,
 first for thresholds s near (p-1)/4. Translate synchronized
thresholds into divisibility or congruence conditions along cycles of
 translations and divisions by powers of p. Derive rigorous necessary
 conditions on p, but do not assume any result from other branches.

Found: This execution develops a first-order threshold/run-count
lower bound. It starts from a coloring c: Z/pZ -> {0,1} with A red
and B blue. For each nonzero step d, it forms the cyclic word w_d(t)=
c(a+td) of length p and notes that a monochromatic 4-term AP is
exactly a run of four equal symbols. Hence every run in w_d has
length at most 3, so if r_d is the number of runs, p <= 3r_d, giving
r_d >= L = ceil(p/3). Let N_d = #{t: c(a+td) != c(a+(t+1)d)}; this is
 the number of runs, so N_d >= L for every d != 0. Encoding by signs
epsilon_x = 1 - 2c(x) in {+-1}, let sigma = sum_x epsilon_x = A - B.
For S_d = sum_x epsilon_x epsilon_{x+d}, one has N_d = (p - S_d)/2,
so S_d <= p - 2L for all d != 0. Summing over d != 0: sum_{d != 0}
S_d = sum_x epsilon_x (sigma - epsilon_x) = sigma^2 - p. Therefore
sigma^2 - p <= (p-1)(p - 2L), equivalently sigma^2 <= p + (p-1)(p - 2
L), or 4A(p-A) >= (p-1)L. The execution also records the same
obstruction in box language: for a block of length m, if c is the
number of internal changes, then m <= 3(c+1), so every translate of
the length-m box along any allowed step has at least ceil(m/3)-1
change edges. For synchronization under multiplication by 2, since p
is odd, w_{2d}(t) = w_d(2t); because every run in w_d has length at
most 3, decimating by 2 cannot create runs longer than 3, so this
first synchronization layer gives no new inequality. A sanity check
is given: for p=7 and A=4, B=3, L=3, the inequality 4*4*3 = 48 >= 18
holds, consistent with the explicit valid coloring. The honest
conclusion is that the first-level counting method yields a rigorous
necessary condition but no congruence obstruction and cannot rule out
 p=7.

Rationale: The run-count bound follows directly from the equivalence
between monochromatic 4-APs and length-4 runs in every d-step cyclic
word. The identity sum_{d != 0} S_d = sigma^2 - p is an exact Fourier
 count over nonzero shifts, and combining it with the uniform bound
S_d <= p-2L gives the stated global inequality. The box formulation
translates the same obstruction into explicit change-edge densities.
The sanity check confirms the inequality is compatible with the only
small explicit colorings, so it is a genuine necessary condition
rather than an artifact.
Core result: Necessary condition: for any coloring avoiding
monochromatic 4-APs with A red, B blue, 4A B >= (p-1) ceil(p/3),
equivalently sigma^2 <= p + (p-1)(p - 2 ceil(p/3)). No congruence
obstruction follows at this level.

Exploration 0b:
...
```

Example Directions

```
<direction>
Independently audit the exhaustive impossibility conclusions for
every unresolved prime p in {13,17,19,23,29,31}. Do not merely rerun
an original solver without justification. Recover a logically
complete constraint-derivation proof or reproducible certificate for
each prime modality the original computational checks support, such
as branch counts, residual clause sets, SAT dual bounds, or a
terminal contradiction ledger. Keep exact progressiveness clauses,
cardinality filters, normalization choices, and any symmetry
reductions explicit. State exactly which conclusions are local
closures and which are exhaustive eliminations, and flag any
unresolved parameter choices or dependence on hidden solver output.
</direction>

<direction>
Push the finite-domain strengthening from Exploration 2c toward a
genuinely transparent contradiction certificate for each exceptional
prime. Begin from its normalized exact Boolean formulation and
rigorously derive forced assignments or implication networks rather
than relying on a solver transcript. The deliverable should be a
contradiction ledger showing that every theoretically possible
assignment violates one exact AP-forbidding clause after
deterministic closure, with branch counts or alternative derivations
for unresolved branches made sufficiently explicit for audit. Do not
allow exclusion arguments that depend only on hidden search output.
</direction>

<direction>
Develop a uniform cyclic interval-or-cycle method for the
intermediate unresolved moduli that goes beyond equal-threshold run
counting. Represent every permitted coloring restriction through
cyclic gap thresholds, modular-residue templates, and simultaneous
correlation constraints across several step directions; then
investigate whether a bounded-size template can contain a generalized
 trap forcing a complementary monochromatic progression. Require each
 inclusion to be justified by an exact progression clause or derived
contradiction, not merely by a weaker cyclic hypothesis. The desired
output is either a uniform obstruction across the unresolved moduli
or a sharply delineated exceptional orbit behavior explaining
failures.
</direction>

<direction>
Extend the affine quadratic-residue structure observed for p=5,7,11
into a rigorous construction analysis governed by cyclic product
orbits, forced color signatures on multiplicative orbits, and affine
modification constraints. Determine exactly which prime orders admit
a compatible sign pattern under this affine grouping and efficiently
verify the surviving small cases or certify obstructions for
intermediate prime moduli. Do not assume that a quadratic-residue
rule remains valid unless the modular stability conditions are
checked separately.
</direction>

<direction>
Explore a multiplier-aware discrepancy formulation based on cyclic
pair-class orbits under multiplication by a fixed primitive root.
Encode each residue's orbit by a coloring signature and impose mixed-
sign requirements involving oriented progressions whose steps are
related to those orbit structures. Derive exact coupling equations
between different multiplier orbits rather than treating each step
```

```
independently, then test whether any allowed coloring must preserve
incompatible sign patterns on representative multiplier classes. Keep
 every modular stability computation finite and check both color
orientations explicitly.
</direction>
```

Table 8: ***Judge calibration on the same 180 IMO ProofBench answers.*** Mean scores are normalized to $[0, 1]$. The final column reports the Pearson correlation with GPT-5.6-Sol at high reasoning effort.

Grader	Mean score	Correlation with Sol-high
GPT-5.6-Sol (high)	0.457	1.000
GPT-5.6-Luna (xhigh)	0.468	0.938
GPT-5.6-Luna (high)	0.472	**0.963**
Gemini 3.1 Pro (high)	0.480	0.954
GPT-5.6-Luna (medium)	0.490	0.929

Table 9: ***TCSBench scores under different judges.*** Mean normalized rubric scores (%) for Qwen3.5-9B (Base), **LOrE**-MidT-9B and **LOrE**-RL-9B, all evaluated with the **LOrE** harness. **LOrE**-RL-9B was trained with GPT-5.6-Luna (high) as the reward judge. Model rankings and the gains from each training stage are consistent across all three judges.

Model	Luna	Sol	Gemini
Qwen3.5-9B	13.55	12.06	7.70
LOrE-MidT-9B	26.71	23.68	19.96
LOrE-RL-9B	32.86	29.23	26.55

K JUDGE CALIBRATION

We make extensive use of LLM judges throughout this work, in order to grade proof-style answers against answer rubrics. We also train against judges during RL. To ensure that our findings are not the result of reward hacking, we perform inter-judge calibration in order to understand whether LLM judges tend to agree or disagree with each other on our tasks.

In Table 8, we calibrate judges against GPT-5.6-Sol (`high` effort) for IMO ProofBench problems, and find that all surveyed judges are in strong agreement with each other.

Next, we test whether the improvements from RL depend on the judge used during training. **LOrE**-RL-9B was trained with GPT-5.6-Luna (high) as the reward judge, which is also our primary TCS-Bench evaluator. If RL had learned to exploit idiosyncrasies of this judge, we would expect its gains to shrink substantially under other judges. We therefore re-grade the TCSBench outputs of Qwen3.5-9B, **LOrE**-MidT-9B and **LOrE**-RL-9B, all evaluated with the **LOrE** harness, using two additional judges: GPT-5.6-Sol (high), a stronger model from the same family, and Gemini-3.8-Flash (high), a judge from a different model family. All judges receive the same rubrics and grading prompt (Appendix H).

Table 9 shows the results. Luna is the most lenient of the three judges for every model, and Gemini the strictest. However, the ordering of models is identical under all judges, and the gains from each training stage are preserved: RL improves over midtraining by 6.2 points under Luna, 5.6 points under Sol and 6.6 points under Gemini. The gap between Luna and Gemini is similar for all three models (5.9, 6.8 and 6.3 points for the base, midtrained and RL models respectively), so Luna does not favour the RL-trained model relative to the others. Luna's margin over Sol grows slightly across training stages (1.5, 3.0 and 3.6 points), but most of this increase arises from midtraining, which does not use Luna as a reward, rather than from RL. We therefore find no evidence that the gains from RL are driven by exploitation of the training judge.

L SFT DETAILS

Table 10: Supervised fine-tuning hyperparameters.

Parameter	Value
Epochs	6
Optimizer	AdamW
Peak learning rate	2×10^{-5}
Final learning rate	2×10^{-6}
Learning-rate schedule	Cosine decay
Warmup fraction	3%
Weight decay	0.01
Global batch size	128
Maximum sequence length	65,536 tokens
Distributed training	FSDP

M RL DETAILS

We largely follow the CISPO settings described in Khatri et al. (2025). We use `bf16` mixed precision training but use FP32 for the rollout LM head, and accumulate trainer logits in FP32, and use $\pi_{\text{old}} = \pi_{\text{rollout}} = \pi_{\text{vLLM}}$ rather than recomputing logprobs with an actor forward. We use `verl` (Sheng et al., 2024) as our RL framework (async RL mode 4).

Table 11: Reinforcement learning configuration. MR, E, and FA denote orchestration, execution, and final-answer generation, respectively. TP and DP denote tensor and data parallelism.

Parameter	Value
Optimization	
Loss	CISPO
`clip_ratio_low`	10
`clip_ratio_high`	4
Optimizer	AdamW
Learning rate	10^{-6} (constant)
Warmup	None
Weight decay	0.01
KL loss	None
Entropy bonus	None
Traj. per prompt (N)	8
Training and rollout infrastructure	
Trainer	Megatron; TP $= 4$, DP $= 2$; 2 nodes
Rollouts (orchestrator)	vLLM; TP $= 1$, DP $= 8$; 2 nodes
Rollouts (executors)	vLLM; TP $= 1$, DP $= 8$; 8 nodes
Asynchronous mode	Fully asynchronous (verl)
Parameter synchronization	Every 2 optimizer steps
`staleness_threshold`	1.0
Sampling and token budgets	
Temperature (MR and E)	1.0
Top-p (MR and E)	1.0
Orchestrator max output tokens	24,576 tokens
Executor max output tokens	65,536 tokens
Final answer max output tokens	49,152 tokens

N EVALUATION

Table 12: Evaluation configuration across benchmarks. Custom judge prompts extract the model's final answer and compare it against the reference.

Benchmark	**Judge (effort)**	**Scoring**	**Prompt**
TCSBench	GPT-5.6-Luna (high)	Rubric (0–7)	TCSBench eval prompt
IMO ProofBench	Gemini-3.1-Pro (high)	Rubric $(0/1/6/7)$	Original paper
ArxivMath-0626	GPT-5.6-Luna (medium)	Final answer	Custom
FrontierScience Olympiad	GPT-5.6-Luna (medium)	Final answer	Original paper
HLE	GPT-5.6-Luna (medium)	Final answer	Custom