SOLVER.EXE: LEARNING TO SOLVE EXPERT-LEVEL PROBLEMS VIA EXPLORATION AND EXPLOITATION AT TEST-TIME

Anonymous authors
Paper under double-blind review

ABSTRACT

An expert develops mastery not only from textbooks, but also through practice and refinement. We show that LLMs can do the same at test time: with the right self-evolution procedure, small- to medium-sized models can more fully realize their reasoning potential and grow into expert-level solvers. Motivated by the classical exploration-exploitation tradeoff, we introduce **Solver.EXE**, a multi-agent test-time learning framework built around a single frozen backbone and an online-evolving memory. Solver.EXE turns each reasoning problem into a miniature learning environment: a Reasoner exploits the current memory to produce rollouts, an Explorer expands the strategy space by surfacing new solutions, an Extractor distills insights and warnings from both kinds of rollouts, and a Curator consolidates them into a compact, evolving memory. Over several expert-level reasoning benchmarks spanning research-level mathematics and broader STEM, Solver.EXE consistently outperforms exploitation-only self-learning baselines and enables 30B-level open-source backbones (Gemma4-26B-A4B-it and Qwen3.5-35B-A3B) to match or surpass proprietary models, including Claude Opus-4.6 and GPT-5.2. Our analysis traces these gains to two latent capabilities of the backbone: bold strategy exploration and careful insight exploitation. These findings suggest that inference-time orchestration is a powerful and still largely untapped axis for advancing reasoning abilities.[1]

1 INTRODUCTION

Expert-level reasoning is not easy. For a human mathematical olympian, it demands years of dedicated learning and practice; for a language model, the prevailing recipe (Kaplan et al., 2020; DeepSeek-AI, 2025) has been to scale to trillion-parameter systems and post-train on large amounts of expert-annotated reasoning data (Anthropic, 2026). As a result, expert-level reasoning (Phan et al., 2025) has become a natural capability moat between frontier closed-source systems and smaller open-source models. However, in this work, we ask: *Can a medium-sized model learn and practice, purely at test time, to solve expert-level reasoning tasks and reach the performance of trillion-parameter models?* We find that the answer is *yes*.

We first localize the unit of evolution to the problem level: each reasoning problem becomes a miniature learning environment. Post-training reinforcement learning (DeepSeek-AI, 2025; Shao et al., 2024) builds reasoning expertise through aggressive *exploration*: sampling diverse trajectories and extracting reward signals from environmental feedback. However, this recipe is not applicable to our setting: environmental rewards are hard to obtain for general reasoning, especially at test time. By contrast, test-time learning (TTL) methods (Hu et al., 2025; Venkatraman et al., 2025) adapt a frozen model online through *exploitation*: sampling from the model's own distribution, learning from the resulting rollouts, and refining subsequent attempts. This reward-free recipe is more general, but its performance ceiling is often bounded by a local optimum (Sutton & Barto, 2018; Maheswaran et al., 2026): because all update information comes from the model's default output distribution, the policy receives little signal to move beyond it.

[1] Our code will be released upon acceptance.

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

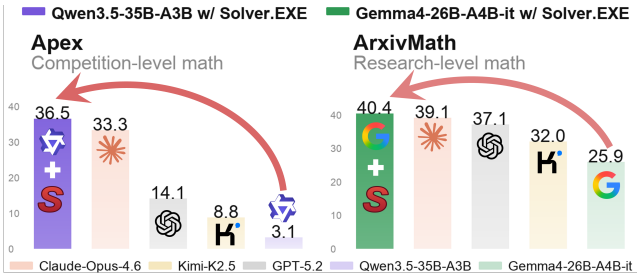

Figure 1: Model performance on expert-level reasoning benchmarks. Our proposed TTL method, Solver.EXE, significantly improves the base model's performance.

Inspired by these two complementary paradigms, we reframe test-time learning as an exploration-exploitation process: *exploration* boldly pushes the model beyond its default solution distribution to discover new insights and strategies, while *exploitation* carefully distills reusable learning signals from the expanded solution space and reuses them to drive self-evolution. We propose **Solver.EXE**, a multi-agent system that leverages these two capabilities of a single backbone model. Given a test problem, four agents powered by the same backbone form an evolution loop: a **Reasoner** samples rollouts from the model's current solution distribution, an **Explorer** expands this distribution by discovering new strategies, an **Extractor** distills insights from the resulting trajectories, and a **Curator** organizes them into an online-evolving memory of problem-specific expertise. As shown in Figure 1, over several rounds of self-evolution, 30B-scale open-source backbones, equipped with the resulting memory, achieve substantial performance gains on expert-level reasoning tasks, ultimately matching or even surpassing contemporary trillion-parameter frontier models.

In summary, our contributions are three-fold:

- We introduce an exploration-exploitation framework for test-time learning and instantiate it as Solver.EXE, a multi-agent system that improves the model's reasoning performance through self-learning.
- Across several representative reasoning benchmarks spanning research-level mathematics and broader STEM expertise, Solver.EXE consistently and significantly improves the performance of its underlying backbones, outperforming prior test-time learning methods.
- Our results and analyses reveal two important latent capabilities in the backbone model: *bold strategy exploration* and *careful insight exploitation*. This finding suggests a promising direction for future work on test-time learning and self-improving agents.

2 Related Works

Test-time learning. Unlike pre-training or post-training (Team, 2024; Abdin et al., 2025; DeepSeek-AI, 2025) that relies on static, human-curated data, test-time learning (TTL) adapts a model to dynamic online queries through a self-supervised closed loop: within the target problem space, the model derives learning signals from its own trajectories and iteratively refines its policy. Existing TTL methods can be broadly divided into *training-based* and *inference-based* paradigms. Recent training-based TTL methods propose various forms of self-generated reward proxies: Zuo et al. (2025) treat majority-voted predictions as pseudo-labels, Zhao et al. (2025) uses the model's self-certainty as a reward signal, and Wang et al. (2026b) rewards the topological structure of reasoning trajectories. In practice, these methods incur substantial training cost, and the reliability of self-generated rewards remains questionable for genuinely expert-level reasoning (Yang et al., 2026; Wang & Heckel, 2026). Our work follows the alternative inference-based paradigm, which keeps the backbone frozen and evolves auxiliary scaffolding. Lange et al. (2025) and Luo et al. (2026) maintain their past high-performing candidates and refine around them for algorithm discovery. Other methods recursively distill reusable experience from the model's own distribution into a memory bank for future guidance (Ouyang et al., 2025; Cai et al., 2025a; Wang et al., 2026a; Yang et al., 2024). Our work follows this memory-driven view but addresses its exploitation-only bottleneck (Maheswaran et al., 2026; Singh et al., 2026) by introducing an exploration operator that surfaces novel solutions and expands the effective search space, thereby raising the evolution ceiling.

Agentic memory. Memory equips LLM agents with knowledge beyond their pre-trained parameters (Anthropic, 2026). Existing memory systems organize information in various forms, including

plain text (Madaan et al., 2023; Packer et al., 2023), summarized responses (Venkatraman et al., 2025; Hu et al., 2026), and structured notes (Yang et al., 2024; Xu et al., 2026). Recently, a growing line of work (Venkatraman et al., 2025; Wang et al., 2026a) has used LLM-as-a-judge mechanisms to extract reasoning hints from both successful and failed attempts. Inspired by these works, we utilize an Extractor to identify both correct and erroneous steps from the model's trajectories, and a Curator to maintain a clean, compact memory bank across evolution rounds.

3 METHODOLOGY

In this section, we first formalize the test-time learning framework and revisit the memory-based learning paradigm. We then introduce **Solver.EXE**, a multi-agent system that improves the base model's reasoning performance at test time through strategy exploration and insight exploitation.

3.1 PRELIMINARIES

Memory-based test-time learning. Given a target query q, memory-based TTL adapts a frozen language model π_θ at test time by evolving the model's in-context memory M in a closed loop. At round t, π_θ produces N candidate rollouts conditioned on the current memory and the query q,

$$\tau_i^{(t)} \sim \pi_\theta\Big(\cdot \,\Big|\, q,\, M^{(t)}\Big), \qquad i = 1, \ldots, N, \tag{1}$$

which are then distilled by an extraction operator $\mathcal{E}$ and used to update memory via the operator $\mathcal{U}$:

$$M^{(t+1)} = \mathcal{U}\Big(M^{(t)},\, \mathcal{E}\big(\{\tau_i^{(t)}\}_{i=1}^{N};\, q\big)\Big). \tag{2}$$

Different choices of M, $\mathcal{E}$, and $\mathcal{U}$ recover a range of recent methods: Test-time Discover (Yüksekgönül et al., 2026) incorporates feedback from program execution into $\mathcal{E}$ and uses it to update M; Self-Refine (Madaan et al., 2023) populates M with the model's self-generated critiques of its previous attempt; and Reasoning Bank (Ouyang et al., 2025; Wang et al., 2026a) enriches M with experiences extracted from past attempts. As t increases, $M^{(t)}$ accumulates self-discovered knowledge, enabling the backbone model π_θ to *evolve* into a stronger, query-specific solver $\pi_\theta(\cdot \mid q, M^{(T)})$ that produces the final answer.

The exploitation bottleneck. When TTL is applied to general reasoning settings (*e.g.*, deriving chemistry results without experimental feedback), previous memory-based learning methods collapse into pure *exploitation*. The memory $M^{(t+1)}$ is updated by rollouts sampled from the same conditional family $\pi_\theta(\cdot \mid q, M^{(t)})$, leaving the loop with no external signal to move π_θ beyond its default distribution. On hard problems where the correct solution lies outside π_θ's default solution space, this exploitation-only recipe limits the performance ceiling. We empirically confirm this failure mode in Section 5.2. This bottleneck mirrors the classical exploration-exploitation tradeoff in reinforcement learning (Sutton & Barto, 2018). Motivated by this analogy, Solver.EXE retains the exploitation operator while introducing an explicit *exploration* operator that expands the effective search space. We describe the resulting agent loop next.

3.2 SOLVER.EXE

As shown in Figure 2, Solver.EXE consists of four cooperating agents: a Reasoner performs memory-guided exploitation, an Explorer discovers alternative paths, an Extractor distills learning signals, and a Curator organizes the evolving memory $M^{(t)}$.

Reasoner. The Reasoner performs the exploitation rollout. At round t, it receives the memory $M^{(t)}$, which contains useful intermediate conclusions and pitfalls identified from previous attempts. It then exploits this memory as in-context guidance to sample N_R candidate rollouts, $\{\tau_i^{(t)}\}_{i=1}^{N_R} \sim \pi_\theta(\cdot \mid q, M^{(t)})$. The detailed exploitation prompt is provided in Appendix G.

Explorer. The Explorer performs exploration in two stages. First, at the beginning of round t, it maintains a *strategy pool* $\mathcal{S}^{(t-1)}$ of high-level solution strategies attempted by the Reasoner in

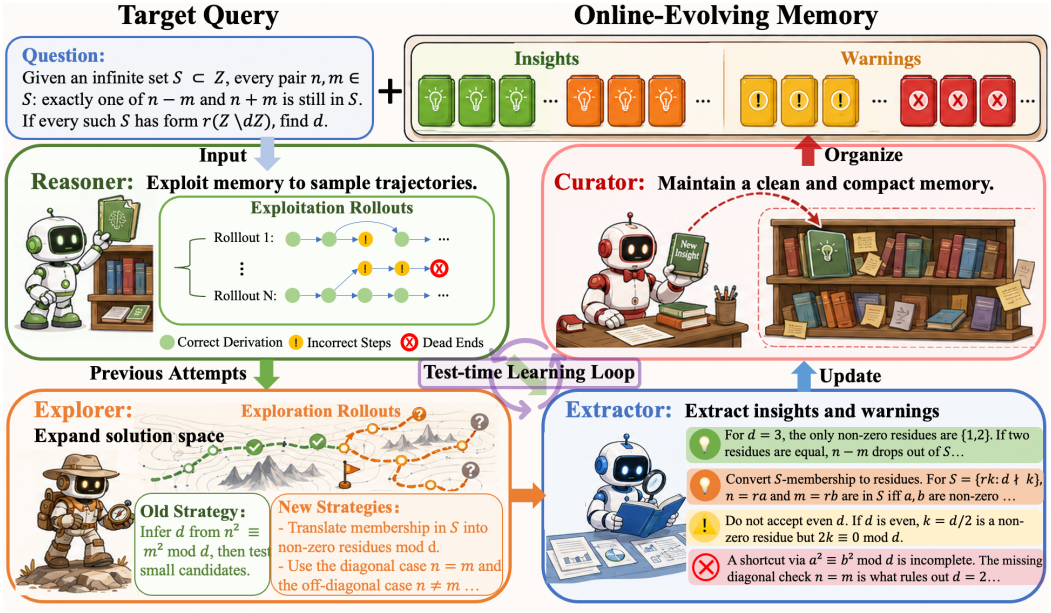

Figure 2: Overview of Solver.EXE. Given a target query and an evolving memory, the Reasoner produces exploitation rollouts by using stored knowledge as in-context guidance, while the Explorer expands the solution space by discovering alternative strategies beyond previous attempts. The Extractor distills insights and warnings from both exploitation and exploration rollouts, and the Curator integrates them into the existing memory while keeping it clean and compact. This loop iteratively turns test-time rollouts into reusable problem-specific knowledge.

previous rounds. After the exploitation rollouts $\{\tau_i^{(t)}\}_{i=1}^{N_R}$ finish, the Explorer summarizes these rollouts into strategies and updates the strategy pool to $\mathcal{S}^{(t)}$. Second, the Explorer explores N_E trajectories $\{\tau_j^{(t)}\}_{j=1}^{N_E}$ from two parallel modes:

- Mode DEVIATE samples in one step, $\tau_j^{(t)} \sim \pi_\theta^{\text{DEVIATE}}(\cdot \mid q, \mathcal{S}^{(t)})$, where $\mathcal{S}^{(t)}$ is provided as a list of strategies to *avoid*.
- Mode DRAFT samples in two steps: $\pi_\theta^{\text{DRAFT}}$ first brainstorms a set of fresh strategies $\mathcal{S}_{\text{new}} \sim \pi_\theta^{\text{DRAFT}}(\cdot \mid q, \mathcal{S}^{(t)})$ that depart from $\mathcal{S}^{(t)}$; it then samples $\tau_j^{(t)} \sim \pi_\theta(\cdot \mid q, \mathcal{S}_{\text{new}})$ guided by these new solution directions.

The two modes trade off in opposite directions: DEVIATE hands π_θ an explicit forbidden list but may still yield only local variations around existing strategies; DRAFT steers π_θ towards only novel solution directions but does not guarantee the executability of the brainstormed strategies. Notably, neither mode conditions on $M^{(t)}$. Instead, the two modes steer the Explorer toward strategy and solution distributions that differ from the corresponding distributions induced by the Reasoner. Together, these two distinct yet complementary sampling procedures further diversify the resulting exploration rollouts, thereby widening the exploration space. The role definitions and prompts for both $\pi_\theta^{\text{DEVIATE}}$ and $\pi_\theta^{\text{DRAFT}}$ are provided in Appendix H.

Extractor. The Extractor instantiates the extraction operator $\mathcal{E}$ from Equation (2). After each round, it processes both exploitation and exploration rollouts from the Reasoner and the Explorer, distilling each into a set of *memory items* $\{m_k\}$ (Wang et al., 2026a; Ouyang et al., 2025):

$$\mathcal{E}(\tau; q) = \{m_k\}, \quad \tau \in \{\tau_i\}_{i=1}^{N_R} \cup \{\tau_j\}_{j=1}^{N_E}. \tag{3}$$

These items are of two types: (1) an INSIGHT is a self-contained intermediate result or conclusion (*e.g.*, a previously established lemma or intermediate result) that future attempts can take as an established reference; (2) a WARNING is a note that flags an erroneous step or dead end in τ (*e.g.*, dividing by a possibly-zero factor without case analysis). Without access to ground-truth signals, the Extractor identifies these items with self-critique and logical deduction, as detailed in Appendix I.

Curator. The Curator instantiates the update operator $\mathcal{U}$ by integrating newly extracted memory items into the evolving memory. A naive append would make $M^{(t)}$ grow without bound, placing increasing pressure on π_θ's context budget as the loop proceeds. Instead, the Curator adopts a memory-consolidation pattern inspired by agentic applications (Anthropic, 2026): before merging new items, it condenses historical $M^{(t)}$ through two steps: (1) PRUNE removes low-quality entries such as duplicates or trivial calculations; (2) CONSOLIDATE scans the memory and folds related entries into single, more informative items. The resulting compact memory is merged with the new items to form $M^{(t+1)}$. We defer the full prompts to Appendix J.

The test-time learning loop. Together, these four agents constitute the test-time learning loop, sharing the same backbone π_θ but using different prompt roles. Given a target query q, the loop starts from $M^{(0)} = \varnothing$ and iterates until either (i) all current-round rollouts from the Reasoner and Explorer agree on the same answer, or (ii) it reaches a maximum budget of T rounds. This early stopping rule balances both search depth for genuinely hard problems and efficiency for converged problems. We then run the Reasoner with the resulting memory to produce the final answer. The detailed algorithm is provided in Appendix B.

4 EXPERIMENTS

We show that Solver.EXE substantially improves the reasoning performance of its underlying open-source backbones and consistently outperforms strong baselines across diverse reasoning tasks.

4.1 SETUP

Benchmarks. Our main experiments cover expert-level reasoning tasks spanning competition mathematics, research-level mathematics, and broader STEM expertise. **Apex** (Balunovic et al., 2025; DeepSeek-AI, 2026) consists of problems curated from mathematical competitions (IMO, SMT, *etc.*) that contemporary frontier models (*e.g.*, GPT-5, Grok-4) fail to solve, providing a challenging testbed for evaluating advanced mathematical reasoning. **ArxivMath** (Dekoninck et al., 2026) comprises research-level mathematical problems derived from 2026 arXiv preprints, reducing contamination risk. For **HLE-Math** and **HLE-STEM**, we use a 200-problem random subset of *Humanity's Last Exam* (Phan et al., 2025), covering expert-level questions in mathematics, physics, and chemistry. We report evaluations on additional benchmarks in Appendix D and provide further dataset details in Appendix C.1.

Baselines. We focus on two representative baselines here, with additional baseline comparisons and details provided in Appendix D.1. (1) **Strong reasoning models** serve as reference points for assessing Solver.EXE's reasoning performance. We report results for frontier models contemporary with our open-source backbones, including GPT-5.2 (High) (OpenAI, 2025), Claude Opus-4.5 and Opus-4.6 (High) (Anthropic, 2026), and Kimi-K2.5 (Thinking) (Team, 2026). (2) **RSE** (Wang et al., 2026a) is a representative memory-based learning method using pure exploitation: it distills raw trajectories into experiences and directly injects them into the context of the next iteration. We compare Solver.EXE with RSE under the same trajectory budget to assess the contributions of our exploration operator and memory consolidation mechanism.

Configuration. We implement Solver.EXE on three open-source backbones, spanning different scales and model architectures: Qwen3-4B-Thinking-2507 (Team, 2025), Qwen3.5-35B-A3B (Qwen Team, 2026), and Gemma-4-26B-A4B-it (Gemma Team, 2026). At each round, we sample $N_R = 16$ exploitation and $N_E = 16$ exploration rollouts, with the maximum number of rounds set to $T = 4$. For fairness, RSE uses the same per-round rollout budget of 32. For evaluation, we estimate Pass@1 after each round by sampling 32 responses conditioned on the current memory. Further details on inference and evaluation are provided in Appendix C.2.

4.2 MAIN RESULTS

Tables 1 and 2 report Pass@1 accuracy for both open-source and reference models, per-round evolution results for three backbones, and per-problem API cost. We highlight three key findings.

Table 1: Performance (Pass@1) and per-problem cost (USD) comparison across model families. **Solver.EXE consistently surpasses both the backbone and RSE, with the strongest results exceeding the reference-model scores.** For TTL methods, Iter k reports Pass@1 after the k-th memory update. Standard deviations are calculated across repeated API samplings for backbone models and across independent evolution runs for TTL methods. **Bold** marks the best score at each evolution round. Red indicates the top-two scores within reference models and TTL methods.

Models	ArxivMath (↑)				Apex (↑)				Cost (↓)
	Iter 1	Iter 2	Iter 3	Iter 4	Iter 1	Iter 2	Iter 3	Iter 4	*Per Problem*
GPT-5.2 (HIGH)	–	–	–	$37.1_{\pm 2.70}$	–	–	–	$14.1_{\pm 5.68}$	$0.50
OPUS-4.6 (HIGH)	–	–	–	**39.1**$_{\pm 3.40}$	–	–	–	**33.3**$_{\pm 5.28}$	$1.71
KIMI-K2.5 (THINKING)	–	–	–	$32.0_{\pm 4.02}$	–	–	–	$8.8_{\pm 4.62}$	$0.15
QWEN3-4B-THINKING	–	–	–	$17.8_{\pm 3.07}$	–	–	–	$0.5_{\pm 2.01}$	$0.01
∟ w/ RSE	20.7	21.8	20.3	$19.7_{\pm 0.44}$	0.8	3.9	7.0	$7.2_{\pm 0.65}$	$1.53
∟ w/ **Solver.EXE**	**24.1**	**23.1**	**22.2**	**22.2**$_{\pm 0.16}$	**3.4**	**8.3**	**8.1**	**8.3**$_{\pm 0.17}$	**$0.79**
QWEN3.5-35B-A3B	–	–	–	$23.6_{\pm 3.82}$	–	–	–	$3.1_{\pm 4.50}$	$0.02
∟ w/ RSE	30.8	31.1	31.6	$31.9_{\pm 1.09}$	15.3	16.4	19.3	$20.9_{\pm 1.79}$	$2.20
∟ w/ **Solver.EXE**	30.8	**34.2**	**36.5**	**37.2**$_{\pm 0.34}$	**26.3**	**31.3**	**33.2**	**36.5**$_{\pm 1.08}$	**$1.65**
GEMMA4-26B-A4B-IT	–	–	–	$25.9_{\pm 3.50}$	–	–	–	$4.9_{\pm 4.33}$	$0.02
∟ w/ RSE	31.0	31.0	32.0	$32.4_{\pm 0.73}$	15.9	21.1	23.0	$23.3_{\pm 0.45}$	$2.29
∟ w/ **Solver.EXE**	**38.1**	**39.8**	**40.2**	**40.4**$_{\pm 0.96}$	**19.3**	**23.2**	**26.8**	**31.3**$_{\pm 0.52}$	**$2.20**

Solver.EXE drives substantial gains through test-time evolution. All benchmarks presented here pose great challenges. On ArxivMath and Apex, the original open-source backbones score only 17.8–25.9 and 0.5–4.9, respectively, while Opus-4.6 scores 39.1 and 33.3. The performance gap extends to HLE-Math and HLE-STEM. Solver.EXE improves the backbones' reasoning performance through iterative exploration and exploitation, coupled with memory updates. Averaged over ArxivMath and Apex, Solver.EXE improves Pass@1 over the raw Qwen3.5-35B-A3B and Gemma4-26B-A4B-it backbones by $+23.5$ and $+20.5$ percentage points, respectively. On HLE-Math and HLE-STEM, the gains over the raw Qwen3.5-35B-A3B backbone are $+13.4$ and $+9.8$ percentage points, respectively. We show in Figures 3 and 7 that these gains can scale further with both parallel search width and the number of sequential iterations.

Solver.EXE surpasses reference models and baselines. With Solver.EXE's test-time orchestration, mean Pass@1 rises from 3.1 to 36.5 for Qwen3.5-35B-A3B on Apex and from 25.9 to 40.4 for Gemma4-26B-A4B-it on ArxivMath, both exceeding the corresponding Opus-4.6 reference scores. On HLE-Math and HLE-STEM, Qwen3.5-35B-A3B with Solver.EXE also achieves higher scores than Opus-4.5 and Kimi-K2.5. Notably, the entire evolution pipeline can run on self-hosted GPU servers to avoid recurring API costs. Solver.EXE also outperforms the exploitation-only baseline RSE across all evaluated benchmarks. Averaged over ArxivMath and Apex, Solver.EXE surpasses RSE by $+10.5$ and $+8.0$ percentage points on Qwen3.5-35B-A3B and Gemma4-26B-A4B-it, respectively, while incurring lower API costs. More baseline comparisons and details are provided in Appendix D. Together, these results highlight Solver.EXE's ability to unlock the reasoning potential of frozen open-source backbones through test-time evolution.

Solver.EXE generalizes across model scale, architecture, and reasoning domain. Solver.EXE delivers consistent improvements across three structurally distinct backbones, including a 4B dense model and two 30B-level MoE models from different families. The magnitude of the lift naturally tracks backbone capacity: stronger models benefit from more rounds of evolution, while the 4B model saturates earlier. Nevertheless, Solver.EXE consistently improves over both the raw backbone and RSE in every setting. The same advantage extends from mathematics to broader STEM skills and coding tasks (Table 7), suggesting that our method generalizes across reasoning domains. This transfer indicates that our exploration-exploitation framework is not merely exploiting domain-specific heuristics, but leveraging *generic* reasoning behaviors, such as strategy exploration, error attribution, and insight exploitation, through its agent roles and interactions. We next analyze how Solver.EXE unlocks these latent capabilities.

Table 2: Comparison of Pass@1 performance and average cost per problem (USD) on Humanity's Last Exam. **Qwen3.5-35B-A3B with Solver.EXE test-time orchestration achieves a higher Pass@1 than both Opus-4.5 and Kimi-K2.5.** Best score at each evolution round is **bolded**.

Models	HLE-Math (↑)				HLE-STEM (↑)				Cost (↓)
	Iter 1	Iter 2	Iter 3	Iter 4	Iter 1	Iter 2	Iter 3	Iter 4	*Per Problem*
OPUS-4.5 (HIGH)	–	–	–	$27.2_{\pm 1.81}$	–	–	–	$26.0_{\pm 2.11}$	\$0.62
KIMI-K2.5 (THINKING)	–	–	–	$24.7_{\pm 0.87}$	–	–	–	$25.2_{\pm 2.26}$	\$0.11
QWEN3-4B-THINKING	–	–	–	$6.1_{\pm 1.16}$	–	–	–	$3.2_{\pm 0.84}$	\$0.01
∟ w/ RSE	7.9	7.9	7.5	$7.9_{\pm 0.38}$	3.4	3.0	2.5	$2.6_{\pm 0.42}$	\$1.20
∟ w/ **Solver.EXE**	**8.3**	**8.9**	**9.3**	$\mathbf{9.7}_{\pm 0.44}$	**4.9**	**5.2**	**5.6**	$\mathbf{6.0}_{\pm 0.77}$	**\$1.00**
QWEN3.5-35B-A3B	–	–	–	$14.1_{\pm 1.62}$	–	–	–	$16.6_{\pm 1.71}$	\$0.02
∟ w/ RSE	20.9	21.1	22.5	$23.0_{\pm 0.35}$	20.4	21.1	22.2	$22.4_{\pm 1.05}$	\$1.99
∟ w/ **Solver.EXE**	**26.3**	**25.8**	**26.9**	$\mathbf{27.5}_{\pm 0.27}$	**23.4**	**25.3**	**25.5**	$\mathbf{26.4}_{\pm 1.23}$	**\$1.83**

5 ANALYSIS

We analyze Solver.EXE from both system-level and model-level perspectives. We first ablate its components and study how it scales with parallel rollout budget and sequential iteration depth. Then we examine the backbone's exploration and exploitation abilities, which drive Solver.EXE's gains and suggest broader directions for test-time learning.

5.1 ABLATION STUDY

Component ablation. As shown in Table 3, removing the Explorer causes a small drop on ArxivMath at Iter 1 (-0.3), but the gap widens to -2.0 at Iter 2. The effect is much stronger on Apex, with drops of -9.6 at Iter 1 and -4.4 at Iter 2. These results suggest that expanding the solution space is crucial for preventing self-learning from saturating early and for solving harder problems that require moving beyond the backbone's default strategies. The Curator has no effect at Iter 1, since $M^{(0)} = \varnothing$ leaves nothing to prune or consolidate. From Iter 2 onward, it consistently improves performance by removing contextual noise and keeping the memory compact, while reducing the context length by 30.5% on average per iteration.

Table 3: Ablation of Solver.EXE components. We use Qwen3.5-35B-A3B as the backbone and report Pass@1 over the first two rounds.

Modules	ArxivMath (↑)		Apex (↑)	
	Iter 1	Iter 2	Iter 1	Iter 2
QWEN3.5-35B-A3B	23.6		3.1	
Solver.EXE	**30.8**	**34.2**	**26.3**	**31.3**
∟ w/o Explorer	30.5	32.2	16.7	26.9
∟ w/o Curator	–	33.7	–	30.2

Parallel and sequential scaling. Figure 3 reports Pass@1 on ArxivMath as we vary the per-round rollout budget $N_R+N_E \in \{8, 16, 32, 64, 128\}$ for up to 7 evolution rounds. Increasing the per-round budget improves both early-round accuracy and the eventual performance ceiling: Iter-1 accuracy rises from 29.8 with 8 rollouts to 35.6 with 128 rollouts, while the long-run plateau increases from 30.9 to 38.1. Sequential scaling then pushes performance toward this ceiling: with our default budget of 32 rollouts, Pass@1 rises from 30.8 at Iter 1 to 37.5 at Iter 7, while the 64- and 128-rollout settings eventually plateau near 38.0. These trends suggest that parallel rollouts broaden the rollout distribution and raise the evolution ceiling, while sequential iterations drive the loop toward it.

5.2 YOUR LANGUAGE MODEL IS SECRETLY A GOOD EXPLORER.

The Explorer is the key component distinguishing Solver.EXE from exploitation-only self-learning methods. Here, we provide qualitative and quantitative evidence that our exploration design expands the model's default strategy space. On hard problems, this expansion leads to breakthrough solution paths reminiscent of an "aha moment", revealing the backbone models' latent exploration ability.

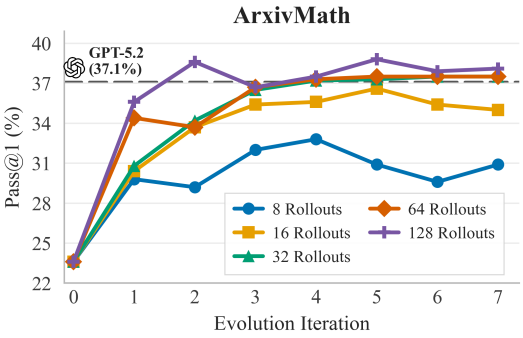

Figure 3: Parallel and sequential scaling on ArxivMath. We vary the per-round rollout budget and report Pass@1 across evolution rounds with Qwen3.5-35B-A3B.

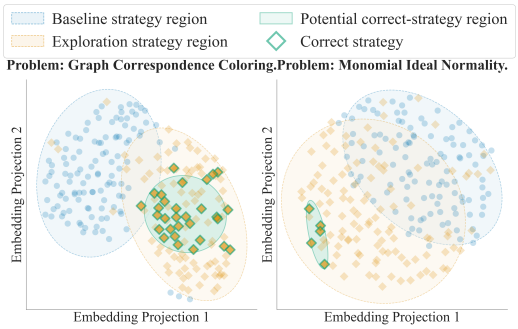

Figure 4: Visualization of default and explored strategy distributions. Exploration expands the space and reaches correct-strategy regions missed by the baseline.

Visualization of strategy space. We first use two ArxivMath cases to provide an intuitive view of the expanded strategy space for Qwen3.5-35B-A3B. For each problem, we extract the solving strategies from the original backbone rollouts and the exploration rollouts, embed them, and project the embeddings to 2D with UMAP (McInnes & Healy, 2018) for visualization. As shown in Figure 4, the exploration-strategy projections deviate substantially from the baseline-strategy projections and expand the covered strategy space. We further highlight the strategies whose rollouts yield the correct answer and approximate the *correct-strategy region* post hoc by fitting a confidence ellipse to them. In both cases, the original backbone's distribution misses this region, suggesting that explicit exploration is necessary for these hard problems.

Quantifying strategy diversity. To complement the visualization, we compute three metrics that measure the diversity of strategies produced by Qwen3.5-35B-A3B on Apex and ArxivMath:

- **Nearest-Baseline Novelty (NBN)**: the average semantic distance from each strategy embedding to its nearest baseline-strategy embedding, excluding itself when applicable (larger values indicate greater distribution shift);
- **Intra-Group Similarity (IGS)**: the mean pairwise cosine similarity within each strategy group (lower indicates greater diversity);
- **Intra-Group Covariance (IGC)**: the trace of the embedding covariance matrix of each strategy group (larger indicates greater diversity).

NBN measures how far a strategy distribution departs from the backbone's default strategies, while IGS and IGC measure its internal dispersion. As shown in Table 4, the exploration operator shifts all metrics in the direction of greater diversity and broader coverage. These results further support that our exploration operator expands the model's strategy space rather than lingering around the local basin of its default solution distribution.

Table 4: Strategy diversity statistics for Qwen3.5-35B-A3B before and after exploration. Arrows indicate the direction corresponding to higher diversity.

Method	NBN ↑	IGS ↓	IGC ↑
QWEN3.5-35B-A3B	0.077	0.838	0.162
⌞ w/ Exploration	**0.134**	**0.780**	**0.220**

Performance across difficulty levels. We group all math problems in our main experiments into four difficulty bands based on the raw π_θ's Pass@1: *Unsolved* (Pass@1 $= 0$), *Hard* (Pass@1 $\in (0, 0.2]$), *Medium* (Pass@1 $\in (0.2, 0.8]$), and *Easy* (Pass@1 > 0.8). For each method, we collect its accuracy after one iteration round and report the accuracy change over the raw backbone in Figure 5.

The performance of strategy exploration supports our "aha moment" hypothesis: it improves the accuracy by $+1.4$ on unsolved problems and by $+3.6$ on the hard problems, thanks to the expanded strategy space. However, its -18.8 drop on the easy problems also suggests that forcing the model away from its already-reliable strategies is unnecessary. The exploitation-only baseline, RSE, ob-

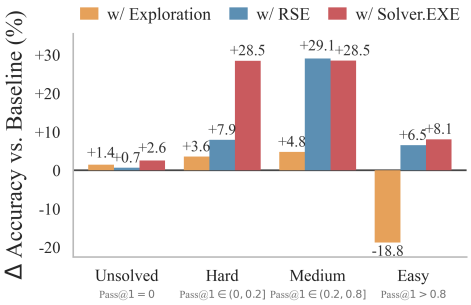

Figure 5: Accuracy improvement over the raw Qwen3.5-35B-A3B backbone across problem difficulty levels.

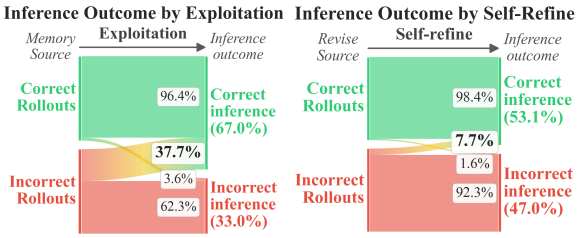

Figure 6: Inference outcomes under controlled source conditions. Each Sankey diagram shows the model's final answer correctness when it is conditioned on memory built exclusively from correct or incorrect source rollouts.

tains its largest gains on the medium and easy problems, where the model's default rollouts already contain reusable correct experiences. Solver.EXE combines both advantages, delivering consistent and non-trivial gains across all difficulty levels. In Appendix E.1, we compare our Explorer with other diverse-sampling baselines to further demonstrate the effectiveness of our mechanism design.

Together, these results suggest that expert-level reasoning requires not only bold exploration that surfaces new solution paths, but also *careful exploitation* that amplifies these potential success signals, which is the next capability we examine.

5.3 Your Language Model is Also a Careful Exploiter

A key requirement for unsupervised self-learning is to distinguish useful learning signals from noisy or misleading content. We therefore test this exploitation ability of our backbone model under a controlled setting. From ArxivMath, we select problems for which Qwen3.5-35B-A3B produces both correct and incorrect rollouts (*i.e.*, $0<\text{Pass@1}<1$). We collect 413 correct and 547 incorrect rollouts from these problems. For each problem, we build memory from either correct-only or incorrect-only source rollouts and evaluate Qwen3.5-35B-A3B conditioned on the resulting memory.

As shown in Figure 6, memory extracted from correct rollouts leads to 96.4% accuracy, as expected. More surprisingly, memory extracted from incorrect rollouts still enables the backbone to achieve a non-trivial 37.7% accuracy. This suggests that failed rollouts are not purely harmful: they can contain partial derivations or warnings that a careful exploiter can use for successful reasoning. For comparison, Self-Refine (Madaan et al., 2023), which prompts the model to revise its past attempts, recovers only 7.7% on the same incorrect rollouts. Together with the Explorer analysis, these results suggest that Solver.EXE succeeds by leveraging two complementary capabilities of the backbone model: *bold exploration* to surface new reasoning directions, and *careful exploitation* to distill and reuse knowledge from these diverse trajectories. In Appendix E.2, we further quantify the proportion of erroneous memory items and assess their impact on subsequent reasoning, finding that (1) only a minority of memory items are erroneous and (2) removing them yields only modest accuracy gains.

6 Conclusion

In this paper, we revisit test-time learning and identify a shared bottleneck of exploitation-only self-learning methods. We address this with **Solver.EXE**, a multi-agent system that instantiates an exploration-exploitation framework over a frozen backbone and an online-evolving memory. Solver.EXE substantially improves the reasoning performance of open-source backbones on several reasoning benchmarks. Our analysis traces these gains to two complementary capabilities of the backbone: *bold exploration*, which expands the strategy space, and *careful exploitation*, which distills and reuses useful knowledge even from noisy trajectories. To further probe the model's potential, we add an optional Verifier agent in Appendix F, which utilizes the model's self-verification ability (Weng et al., 2023) and yields additional gains over the main Solver.EXE results. We hope our findings could encourage the community to view inference-time orchestration as a powerful axis for unlocking the underexplored potential of current models.

AI USE STATEMENT

We used generative AI tools to assist with manuscript writing and editing, including improving wording, clarity, and organization. We manually reviewed and revised all AI-assisted text and checked its accuracy and consistency with the methods and experimental results. The use of LLMs within Solver.EXE and for the memory audit is described in the corresponding methodological and experimental sections. We take full responsibility for the final content of this work, including all claims, results, and AI-assisted text.

REPRODUCIBILITY STATEMENT

We provide step-by-step pseudocode for the complete Solver.EXE pipeline in Appendix B, and the full prompts for all four agent roles in Appendices G to J. Benchmark selection and evaluation protocols, model specifications, rollout budgets, decoding parameters, software and hardware configurations, and artifact licenses are documented in Appendix C. Additional baseline configurations and end-to-end GPU usage and runtime are reported in Appendices D.1 and D.2. We will release the complete codebase upon acceptance.

REFERENCES

Marah I Abdin, Sahaj Agarwal, Ahmed Awadallah, Vidhisha Balachandran, Harkirat S. Behl, Lingjiao Chen, Gustavo de Rosa, Suriya Gunasekar, Mojan Javaheripi, Neel Joshi, Piero Kauffmann, Yash Lara, Caio César Teodoro Mendes, Arindam Mitra, Besmira Nushi, Dimitris Papailiopoulos, Olli Saarikivi, Shital Shah, Vaishnavi Shrivastava, Vibhav Vineet, Yue Wu, Safoora Yousefi, and Guoqing Zheng. Phi-4-reasoning technical report. *CoRR*, abs/2504.21318, 2025. doi: 10.48550/ARXIV.2504.21318. URL https://doi.org/10.48550/arXiv.2504.21318.

Anthropic. System Card: Claude Opus 4.6. https://www-cdn.anthropic.com/0dd865075ad3132672ee0ab40b05a53f14cf5288.pdf, February 2026. Accessed: 2026-05-16.

Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanovic, and Martin T. Vechev. Matharena: Evaluating llms on uncontaminated math competitions. *CoRR*, abs/2505.23281, 2025. doi: 10.48550/ARXIV.2505.23281. URL https://doi.org/10.48550/arXiv.2505.23281.

Rui Cai, Weijie Jacky Mo, Xiaofei Wen, Qiyao Ma, Wenhui Zhu, Xiwen Chen, Muhao Chen, and Zhe Zhao. Modellens: Finding the best for your task from myriads of models. *arXiv preprint arXiv:2605.07075*, 2026.

Yuzheng Cai, Siqi Cai, Yuchen Shi, Zihan Xu, Lichao Chen, Yulei Qin, Xiaoyu Tan, Gang Li, Zongyi Li, Haojia Lin, Yong Mao, Ke Li, and Xing Sun. Training-free group relative policy optimization. *CoRR*, abs/2510.08191, 2025a. doi: 10.48550/ARXIV.2510.08191. URL https://doi.org/10.48550/arXiv.2510.08191.

Zhicheng Cai, Xinyuan Guo, Yu Pei, Jiangtao Feng, Jiangjie Chen, Ya-Qin Zhang, Wei-Ying Ma, Mingxuan Wang, and Hao Zhou. FLEX: continuous agent evolution via forward learning from experience. *CoRR*, abs/2511.06449, 2025b. doi: 10.48550/ARXIV.2511.06449. URL https://doi.org/10.48550/arXiv.2511.06449.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948, 2025. doi: 10.48550/ARXIV.2501.12948. URL https://doi.org/10.48550/arXiv.2501.12948.

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.

Gemma Team. Gemma 4 Model Card. https://ai.google.dev/gemma/docs/core/model_card_4, 2026. Google DeepMind. Accessed: 2026-05-16.

Jingcheng Hu, Yinmin Zhang, Shijie Shang, Xiaobo Yang, Yue Peng, Zhewei Huang, Hebin Zhou, Xin Wu, Jie Cheng, Fanqi Wan, Xiangwen Kong, Chengyuan Yao, Kaiwen Yan, Ailin Huang, Hongyu Zhou, Qi Han, Zheng Ge, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Pacore: Learning to scale test-time compute with parallel coordinated reasoning. *CoRR*, abs/2601.05593, 2026. doi: 10.48550/ARXIV.2601.05593. URL https://doi.org/10.48550/arXiv.2601.05593.

Jinwu Hu, Zitian Zhang, Guohao Chen, Xutao Wen, Chao Shuai, Wei Luo, Bin Xiao, Yuanqing Li, and Mingkui Tan. Test-time learning for large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/hu25z.html.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *CoRR*, abs/2509.19349, 2025. doi: 10.48550/ARXIV.2509.19349. URL https://doi.org/10.48550/arXiv.2509.19349.

Sichun Luo, Yi Huang, Haochen Luo, Fengyuan Liu, Guanzhi Deng, Lei Li, Qinghua Yao, Zefa Hu, Junlan Feng, and Qi Liu. Seaevo: Advancing algorithm discovery with strategy space evolution. *CoRR*, abs/2604.24372, 2026. doi: 10.48550/ARXIV.2604.24372. URL https://doi.org/10.48550/arXiv.2604.24372.

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. Skillclaw: Let skills evolve collectively with agentic evolver. *CoRR*, abs/2604.08377, 2026. doi: 10.48550/ARXIV.2604.08377. URL https://doi.org/10.48550/arXiv.2604.08377.

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 Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Monishwaran Maheswaran, Rishabh Tiwari, Yuezhou Hu, Kerem Dilmen, Coleman Hooper, Haocheng Xi, Nicholas Lee, Mehrdad Farajtabar, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Arbitrage: Efficient reasoning via advantage-aware speculation. *CoRR*, abs/2512.05033, 2025. doi: 10.48550/ARXIV.2512.05033. URL https://doi.org/10.48550/arXiv.2512.05033.

Monishwaran Maheswaran, Leon Lakhani, Zhongzhu Zhou, Shijia Yang, Junxiong Wang, Coleman Hooper, Yuezhou Hu, Rishabh Tiwari, Jue Wang, Harman Singh, Qingyang Wu, Yuqing Jian, Ce Zhang, Kurt Keutzer, Tri Dao, Xiaoxia Wu, Ben Athiwaratkun, James Zou, and Chenfeng Xu. Squeeze evolve: Unified multi-model orchestration for verifier-free evolution. *CoRR*, abs/2604.07725, 2026. doi: 10.48550/ARXIV.2604.07725. URL https://doi.org/10.48550/arXiv.2604.07725.

Leland McInnes and John Healy. UMAP: uniform manifold approximation and projection for dimension reduction. *CoRR*, abs/1802.03426, 2018. URL http://arxiv.org/abs/1802.03426.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data. *CoRR*, abs/2406.18665, 2024. doi: 10.48550/ARXIV.2406.18665. URL https://doi.org/10.48550/arXiv.2406.18665.

OpenAI. Update to GPT-5 System Card: GPT-5.2. https://openai.com/index/gpt-5-system-card-update-gpt-5-2/, December 2025. Accessed: 2026-05-16.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory. *CoRR*, abs/2509.25140, 2025. doi: 10.48550/ARXIV.2509.25140. URL https://doi.org/10.48550/arXiv.2509.25140.

Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *CoRR*, abs/2310.08560, 2023. doi: 10.48550/ARXIV.2310.08560. URL https://doi.org/10.48550/arXiv.2310.08560.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, Adam Khoja, Ryan Kim, Richard Ren, Jason Hausenloy, Oliver Zhang, Mantas Mazeika, Summer Yue, Alexandr Wang, and Dan Hendrycks. Humanity's last exam. *CoRR*, abs/2501.14249, 2025. doi: 10.48550/ARXIV.2501.14249. URL https://doi.org/10.48550/arXiv.2501.14249.

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

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

Harman Singh, Xiuyu Li, Kusha Sareen, Monishwaran Maheswaran, Sijun Tan, Xiaoxia Wu, Junxiong Wang, Alpay Ariyak, Qingyang Wu, Samir Khaki, Rishabh Tiwari, Long Lian, Yucheng Lu, Boyi Li, Alane Suhr, Ben Athiwaratkun, and Kurt Keutzer. V_1: Unifying generation and self-verification for parallel reasoners. *CoRR*, abs/2603.04304, 2026. doi: 10.48550/ARXIV.2603.04304. URL https://doi.org/10.48550/arXiv.2603.04304.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction, 2nd Edition*. MIT Press, 2018. URL http://www.incompleteideas.net/book/the-book-2nd.html.

Kimi Team. Kimi K2.5: visual agentic intelligence. *CoRR*, abs/2602.02276, 2026. doi: 10.48550/ARXIV.2602.02276. URL https://doi.org/10.48550/arXiv.2602.02276.

Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://doi.org/10.48550/arXiv.2407.21783.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Siddarth Venkatraman, Vineet Jain, Sarthak Mittal, Vedant Shah, Johan S. 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. *CoRR*, abs/2509.26626, 2025. doi: 10.48550/ARXIV.2509.26626. URL https://doi.org/10.48550/arXiv.2509.26626.

Kun Wang and Reinhard Heckel. Test-time RL alignment exposes task familiarity artifacts in LLM benchmarks. *CoRR*, abs/2603.12875, 2026. doi: 10.48550/ARXIV.2603.12875. URL https://doi.org/10.48550/arXiv.2603.12875.

Xinglin Wang, Jiayi Shi, Shaoxiong Feng, Peiwen Yuan, Yiwei Li, Yueqi Zhang, Chuyi Tan, Ji Zhang, Boyuan Pan, Yao Hu, and Kan Li. Do not waste your rollouts: Recycling search experience for efficient test-time scaling. *CoRR*, abs/2601.21684, 2026a. doi: 10.48550/ARXIV.2601.21684. URL https://doi.org/10.48550/arXiv.2601.21684.

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 *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yifan Wang, Bolian Li, David Cho, Ruqi Zhang, Fanping Sui, and Ananth Grama. SARL: label-free reinforcement learning by rewarding reasoning topology. *CoRR*, abs/2603.27977, 2026b. doi: 10.48550/ARXIV.2603.27977. URL https://doi.org/10.48550/arXiv.2603.27977.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, Findings of ACL, pp. 2550–2575. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.167. URL https://doi.org/10.18653/v1/2023.findings-emnlp.167.

Peng Xia, Jianwen Chen, Xinyu Yang, Haoqin Tu, Jiaqi Liu, Kaiwen Xiong, Siwei Han, Shi Qiu, Haonian Ji, Yuyin Zhou, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Metaclaw: Just talk - an agent that meta-learns and evolves in the wild. *CoRR*, abs/2603.17187, 2026. doi: 10.48550/ARXIV.2603.17187. URL https://doi.org/10.48550/arXiv.2603.17187.

Buqiang Xu, Yijun Chen, Jizhan Fang, Ruobin Zhong, Yunzhi Yao, Yuqi Zhu, Lun Du, and Shumin Deng. Structmem: Structured memory for long-horizon behavior in llms. *CoRR*, abs/2604.21748, 2026. doi: 10.48550/ARXIV.2604.21748. URL https://doi.org/10.48550/arXiv.2604.21748.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/cde328b7bf6358f5ebb91fe9c539745e-Abstract-Conference.html.

Shenzhi Yang, Guangcheng Zhu, Bowen Song, Sharon Li, Haobo Wang, Xing Zheng, Yingfan Ma, Zhongqi Chen, Weiqiang Wang, and Gang Chen. Can llms learn to reason robustly under noisy supervision? *CoRR*, abs/2604.03993, 2026. doi: 10.48550/ARXIV.2604.03993. URL https://doi.org/10.48550/arXiv.2604.03993.

Mert Yüksekgönül, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time. *CoRR*, abs/2601.16175, 2026. doi: 10.48550/ARXIV.2601.16175. URL https://doi.org/10.48550/arXiv.2601.16175.

Xuandong Zhao, Zhewei Kang, Aosong Feng, Sergey Levine, and Dawn Song. Learning to reason without external rewards. *CoRR*, abs/2505.19590, 2025. doi: 10.48550/ARXIV.2505.19590. URL https://doi.org/10.48550/arXiv.2505.19590.

Yuxin Zuo, Kaiyan Zhang, Shang Qu, Li Sheng, Xuekai Zhu, Biqing Qi, Youbang Sun, Ganqu Cui, Ning Ding, and Bowen Zhou. TTRL: test-time reinforcement learning. *CoRR*, abs/2504.16084, 2025. doi: 10.48550/ARXIV.2504.16084. URL https://doi.org/10.48550/arXiv.2504.16084.

APPENDIX CONTENTS

A Limitations **16**

B Solver.EXE Pipeline **17**

C Experiment Details **18**

C.1 Benchmark Details 18

C.2 Inference Configurations 18

C.3 Artifact Licenses 19

D Additional Experiments for Section 4 **20**

D.1 Additional Baseline Comparisons and Cross-Domain Evaluation 20

D.2 GPU Cost and Runtime Analysis 20

D.3 Evolution Scaling with Early Stopping 21

E Additional Experiments for Section 5 **23**

E.1 Explorer Design and Sampling Diversity 23

E.2 Memory Quality and Reasoning Robustness 23

F Extending Solver.EXE with Self-Verification **25**

F.1 Verifier Design 25

F.2 Experimental Setting and Results 25

G Details on the Reasoner **29**

G.1 Reasoner Design Summary 29

G.2 Reasoner Case Study 29

H Details on the Explorer **34**

H.1 Explorer Design Summary 34

H.2 Explorer Case Study 34

I Details on the Extractor **50**

I.1 Extractor Design Summary 50

I.2 Extractor Output Examples 50

J Details on the Curator **55**

A LIMITATIONS

Solver.EXE focuses on foundational reasoning domains such as mathematics and STEM expertise where online environmental feedback is unavailable or difficult to obtain instantly. This feedback-free setting is deliberately general: it lets us study the performance ceiling of self-learning without relying on task-specific interaction signals, and examine the latent reasoning potential of backbone models. A parallel line of test-time learning work (Ma et al., 2026; Xia et al., 2026; Cai et al., 2025b) studies more specific user-in-the-loop agentic applications, where agents can collect user instructions and external execution results in addition to test-time trajectories, and convert them into supervised signals for constructing agentic skill documents. We expect the exploration-exploitation principle behind Solver.EXE to transfer to these settings. Adapting this principle to application-specific feedback and skill construction is a promising direction, and we hope our work can inspire future community efforts along this line.

Another limitation is that Solver.EXE uses the same backbone for all agent roles. This design isolates the self-learning capability of a single model, but it may not be optimal in deployed systems. Future variants could allocate stronger models or larger compute budgets to reasoning-intensive roles such as the Reasoner and Explorer, while using more efficient models for lighter roles such as the Extractor and Curator. Since our main goal is to reveal the self-learning ceiling and latent capabilities of current open-source models, we leave heterogeneous agent allocation (Maheswaran et al., 2025; Cai et al., 2026; Ong et al., 2024) to future work and practical agentic applications.

B SOLVER.EXE PIPELINE

Table 5: Symbols used in the Solver.EXE pipeline.

Symbol	Meaning	Symbol	Meaning
q	target query to solve	π_θ	frozen backbone shared by all prompt roles
T	maximum number of evolution rounds	$t_\star$	actual stopping round used for final inference
N_R	number of Reasoner exploitation rollouts per round	N_E	total number of Explorer rollouts per round
N_D, N_F	rollout budgets for DEVIATE and DRAFT branches	$M^{(t)}$	memory bank before round t
$\mathcal{S}^{(t)}$	cumulative strategy pool through round t, updated only from Reasoner rollouts	$\mathcal{T}_R^{(t)}$	Reasoner rollouts at round t
$\mathcal{T}_D^{(t)}$	Explorer rollouts from DEVIATE		
$\mathcal{B}^{(t)}$	fresh strategy sketches from DRAFT	$\mathcal{T}_F^{(t)}$	rollouts executed from draft sketches
$\mathcal{T}^{(t)}$	all rollouts collected in round t	$\Delta M^{(t)}$	newly extracted memory items
$\mathcal{E}$	Extractor: rollout-to-memory operator	$\mathcal{U}$	Curator: memory update operator
τ_{final}	final Reasoner rollout after evolution	$\hat{a}$	final answer extracted from τ_{final}

Algorithm 1 Solver.EXE test-time evolution for a target query.

Require: Target query q; frozen backbone π_θ; maximum rounds T; rollout budgets N_R and N_E.
Ensure: Final answer $\hat{a}$.

$M^{(0)} \leftarrow \varnothing, \mathcal{S}^{(-1)} \leftarrow \varnothing, t_\star \leftarrow 0$ ▷ start with no memory or recorded strategy before round 0
$N_D \leftarrow \lfloor N_E/2 \rfloor, N_F \leftarrow N_E - N_D$ ▷ split exploration between two modes
for $t = 0, 1, \ldots, T-1$ **do** ▷ repeat until convergence or budget exhaustion
  $\mathcal{T}_R^{(t)} \leftarrow \text{REASONER}(q, M^{(t)}; N_R)$ ▷ exploit current memory
  $\mathcal{S}^{(t)} \leftarrow \mathcal{S}^{(t-1)} \cup \text{STRATEGYEXTRACT}(\mathcal{T}_R^{(t)}; q)$ ▷ accumulate Reasoner strategies through round t
  $\mathcal{T}_D^{(t)} \leftarrow \text{EXPLORERDEVIATE}(q, \mathcal{S}^{(t)}; N_D)$ ▷ sample while avoiding Reasoner strategies
  $\mathcal{B}^{(t)} \leftarrow \text{EXPLORERDRAFT}(q, \mathcal{S}^{(t)}; N_F)$ ▷ brainstorm fresh strategy sketches
  $\mathcal{T}_F^{(t)} \leftarrow \text{EXECUTEDRAFTS}(q, \mathcal{B}^{(t)})$ ▷ turn sketches into full rollouts
  $\mathcal{T}^{(t)} \leftarrow \mathcal{T}_R^{(t)} \cup \mathcal{T}_D^{(t)} \cup \mathcal{T}_F^{(t)}$ ▷ pool exploitation and exploration
  $\Delta M^{(t)} \leftarrow \bigcup_{\tau \in \mathcal{T}^{(t)}} \mathcal{E}(\tau; q)$ ▷ extract insights and warnings
  $M^{(t+1)} \leftarrow \mathcal{U}(M^{(t)}, \Delta M^{(t)})$ ▷ prune, consolidate, and merge memory
  $t_\star \leftarrow t + 1$ ▷ latest valid memory index
  if $\text{AGREE}(\mathcal{T}^{(t)})$ **then** ▷ all normalized final answers match
    break ▷ early stop on converged problems
  end if
end for
$\tau_{\text{final}} \leftarrow \text{REASONER}(q, M^{(t_\star)}; 1)$ ▷ solve once with the final memory
return $\hat{a} \leftarrow \text{ANSWER}(\tau_{\text{final}})$ ▷ extract the final answer

C EXPERIMENT DETAILS

C.1 BENCHMARK DETAILS

Here we detail the construction, scale, and evaluation protocols of the four benchmarks used in our experiments.

Apex. Apex is sourced from the MathArena project (Balunovic et al., 2025), which aggregates problems from major 2025 mathematical competitions including IMO, SMT, and similar olympiad-level contests. The 12 retained problems are those on which all evaluated contemporary frontier models produce incorrect final answers, including GPT-5 (High), Grok-4, Gemini-2.5-Pro, and GLM-4.5. This adversarial selection criterion positions Apex as a stress test for expert-level reasoning. Evaluation follows the official MathArena protocol on final-answer correctness.

ArxivMath. ArxivMath (2026.02) contains 32 research-level mathematical problems derived from arXiv preprints posted in 2026, drawn from the MathArena collection (Dekoninck et al., 2026). Each problem is a self-contained mathematical question extracted from a research paper's main results. Evaluation follows the official MathArena protocol on final-answer correctness.

HLE-Math and HLE-STEM. Both subsets are derived from Humanity's Last Exam (Phan et al., 2025), an expert-curated benchmark covering frontier knowledge across disciplines. To ensure consistent evaluation across our text-only backbones, we restrict to problems *without* image inputs and randomly sample 100 problems each from the mathematics (HLE-Math) and physics-and-chemistry (HLE-STEM) categories, yielding 200 problems in total. Scoring follows the original HLE protocol: O3-MINI serves as the LLM judge, comparing the model's final answer against the reference answer and assigning binary correctness.

C.2 INFERENCE CONFIGURATIONS

Here we detail all hyperparameters and configurations used in our experiments.

Solver.EXE configuration. We set $N_R = N_E = 16$ rollouts per evolution round and cap the loop at $T = 4$ rounds. The $N_E = 16$ exploration rollouts are split evenly between two strategy-exploration modes introduced in Section 3.2: 8 rollouts via the DEVIATE branch (forced to avoid strategies already in the strategy pool) and 8 rollouts via the DRAFT branch (first sketching new strategy outlines, then executing them). For both the Extractor and the Explorer's strategy-extraction step, we process one rollout at a time and use a single response sample, so each rollout yields one set of memory items and one extracted strategy.

Hardware. All open-source backbone inference is conducted on 48 NVIDIA H200 GPUs using vLLM v0.19.0 (Kwon et al., 2023). Opus-4.5/4.6, GPT-5.2, and Kimi-K2.5 are accessed through their official provider APIs.

Decoding parameters. Per-backbone decoding hyperparameters are as follows:

- **Qwen3-4B-Thinking**: $T = 0.6$, top-$p = 0.95$, top-$k = 20$.
- **Qwen3.5-35B-A3B**: $T = 0.6$, top-$p = 0.95$, top-$k = 20$.
- **Gemma4-26B-A4B-it**: $T = 1.0$, top-$p = 0.95$, top-$k = 64$, repetition penalty $= 1.05$.

We set the maximum output length to $81{,}920$ tokens for ArxivMath and Apex, and to $64{,}000$ tokens for HLE-Math and HLE-STEM. For open-source backbones, we estimate Pass@1 from $n = 32$ sampled responses per problem at the end of each evolution round, with all reported scores averaged over 4 independent runs. For reference models (*e.g.*, Claude Opus-4.5, Kimi-K2.5), we sample 16 responses per problem using each provider's default API parameters. All experiments use pure text-only inference: no external tools, retrieval, or code execution are involved. The per-problem cost reported in Tables 1 and 2 is computed by multiplying observed token usage (input + output) by

each provider's official per-token rates. For Gemma4-26B-A4B-it, we use OpenRouter rates as a proxy.[2]

C.3 Artifact Licenses

For benchmarks, Apex is released under the CC BY-NC-SA 4.0 license, and ArxivMath is released under the CC BY-SA 4.0 license. Humanity's Last Exam and LiveCodeBench are released under the MIT License. We use these artifacts only for evaluation and follow the terms specified by their official releases.

For model artifacts, we use Qwen3-4B-Thinking (Apache license 2.0), Qwen3.5-35B-A3B (Apache license 2.0), and Gemma4-26B-A4B-it (Apache license 2.0) according to their official model licenses or terms of use. Other models, including GPT-5.2, Claude Opus-4.5/4.6, and Kimi-K2.5, are accessed only through their official provider APIs and are subject to the corresponding provider terms. We do not redistribute any third-party model weights.

[2]OpenRouter (Gemma4) `https://openrouter.ai/google/gemma-4-26b-a4b-it`.

D Additional Experiments for Section 4

D.1 Additional Baseline Comparisons and Cross-Domain Evaluation

We extend the main experiments with additional baseline comparisons on ArxivMath and Apex and evaluations on LiveCodeBench v6 (Jain et al., 2025) and GPQA Diamond (Rein et al., 2024), covering code generation and scientific reasoning. The baselines are described below.

- **Majority Voting** (Wang et al., 2023). Majority Voting samples multiple independent solutions from the backbone and selects the final answer by voting over the normalized final answers. We evaluate it with $n = 32$ sampled candidate solutions per query.
- **Self-Refine** (Madaan et al., 2023). Self-Refine prompts a single LLM to iteratively improve its own output by generating natural-language feedback on the previous response and then producing a revised answer conditioned on that feedback.
- **RSA** (Venkatraman et al., 2025). Recursive Self-Aggregation maintains a population of candidate solutions and recursively aggregates randomly sampled subsets into increasingly refined responses. We follow the original setup with a population of $N = 32$ candidates and an aggregation subset size of $K = 4$ at each recursion step.
- **RSE** (Wang et al., 2026a). Recycling Search Experience (RSE) is an exploitation-only test-time scaling baseline that collects reusable experience from previous rollouts and injects it into subsequent inference contexts. We allocate RSE the same rollout budget as Solver.EXE, with 32 candidate trajectories per query.

For a controlled comparison, all baselines share the same experimental setup as our main experiments: we use Qwen3.5-35B-A3B as the backbone model, cap the maximum number of refinement / aggregation iterations at 4, and keep the remaining inference hyperparameters (temperature, top-p, maximum generation length, etc.) identical to those reported in Section 4.

Table 6: Additional baseline comparison on ArxivMath and Apex using Qwen3.5-35B-A3B as the backbone. **Solver.EXE consistently outperforms iterative refinement, recursive aggregation, and RSE baselines across evolution rounds. Bold** marks the best score at each evolution round.

Models	ArxivMath (↑)				Apex (↑)			
	Iter 1	Iter 2	Iter 3	Iter 4	Iter 1	Iter 2	Iter 3	Iter 4
QWEN3.5-35B-A3B	–	–	–	$23.6_{\pm 3.82}$	–	–	–	$3.1_{\pm 4.50}$
∟ w/ Majority Voting	–	–	–	28.1	–	–	–	0.0
∟ w/ Self-Refine	25.0	25.2	25.7	$26.0_{\pm 0.75}$	6.7	7.0	7.0	$7.5_{\pm 1.54}$
∟ w/ RSA	25.9	29.8	33.1	$32.9_{\pm 1.43}$	7.3	12.0	21.6	$23.9_{\pm 1.88}$
∟ w/ RSE	30.8	31.1	31.6	$31.9_{\pm 1.09}$	15.3	16.4	19.3	$20.9_{\pm 1.79}$
∟ w/ **Solver.EXE**	30.8	**34.2**	**36.5**	$\mathbf{37.2}_{\pm 0.34}$	**26.3**	**31.3**	**33.2**	$\mathbf{36.5}_{\pm 1.08}$

Advantages over additional baselines. Table 6 shows that Solver.EXE matches or exceeds all compared baselines at every reported evolution round. After four rounds, it reaches 37.2 on ArxivMath, exceeding RSA and RSE by 4.3 and 5.3 percentage points, respectively. On Apex, it reaches 36.5, outperforming RSA and RSE by 12.6 and 15.6 percentage points, respectively.

Generalization across reasoning domains. Table 7 shows that, after four rounds, Solver.EXE raises Pass@1 from 62.1 to 76.7 on LiveCodeBench v6 and from 80.9 to 88.7 on GPQA Diamond. It also outperforms RSA and RSE at every evolution round on both benchmarks. These results show that the gains from test-time evolution extend to code generation and scientific reasoning, supporting the applicability of Solver.EXE across reasoning domains.

D.2 GPU Cost and Runtime Analysis

To complement the per-problem API costs reported in Tables 1 and 2, we measure the end-to-end local inference cost of RSE, RSA, and Solver.EXE. For this comparison, all methods use Qwen3.5-35B-A3B on a single server with 8 NVIDIA H200 GPUs under their default four-round settings.

Table 7: Pass@1 performance on LiveCodeBench v6 and GPQA Diamond using Qwen3.5-35B-A3B as the backbone. **Solver.EXE outperforms RSA and RSE at every evolution round on both benchmarks. Bold** marks the best score at each evolution round within each benchmark.

Models	LiveCodeBench v6 (↑)				GPQA Diamond (↑)			
	Iter 1	Iter 2	Iter 3	Iter 4	Iter 1	Iter 2	Iter 3	Iter 4
QWEN3.5-35B-A3B	–	–	–	$62.1_{\pm 1.99}$	–	–	–	$80.9_{\pm 2.61}$
∟ w/ RSA	63.7	67.5	69.1	$69.9_{\pm 0.85}$	83.6	84.1	84.7	$85.1_{\pm 0.31}$
∟ w/ RSE	66.3	66.8	67.2	$67.3_{\pm 0.89}$	82.3	84.8	85.0	$85.2_{\pm 0.28}$
∟ w/ **Solver.EXE**	**74.5**	**75.1**	**75.8**	$\mathbf{76.7}_{\pm 1.22}$	**84.8**	**86.5**	**88.4**	$\mathbf{88.7}_{\pm 0.43}$

Table 8: End-to-end local inference cost per problem, averaged equally across ArxivMath and Apex. All methods use the same Qwen3.5-35B-A3B backbone and an 8-GPU H200 server under their default four-round settings.

Method	GPU-hours	Wall-clock time (hours)
RSE	4.09	0.51
RSA	3.36	0.42
Solver.EXE	3.51	0.44

Table 8 shows that Solver.EXE requires 3.51 GPU-hours and 0.44 hours of wall-clock time per problem. Compared with RSE, it reduces both GPU-hours and runtime by approximately 14%, while its overhead relative to RSA remains below 5% for both metrics. Together with the accuracy gains in Table 6, these measurements show that Solver.EXE improves reasoning performance without disproportionate computational overhead relative to the compared test-time learning methods.

D.3 Evolution Scaling with Early Stopping

In the main experiments, we cap Solver.EXE at $T = 4$ evolution rounds for a controlled comparison with prior test-time scaling baselines. Here, we extend this evolution budget to $T = 7$ rounds. Figure 7 reports the scaling trajectory on HLE-Math and ArxivMath using Qwen3.5-35B-A3B as the backbone. The red curve shows Solver.EXE Pass@1 accuracy across evolution rounds, where $x = 0$ denotes the raw backbone before evolution, and the gray curve shows the ratio of active problems that have not yet been stopped by the early stopping mechanism. Dashed horizontal lines mark the corresponding reference-model scores for context.

Sequential scaling yields further gains. Solver.EXE improves Qwen3.5-35B-A3B from 23.6 to 37.2 on ArxivMath and from 14.1 to 27.5 on HLE-Math within the first four evolution rounds. Extending the evolution budget from four to seven rounds further improves Pass@1 to 37.5 on ArxivMath and 28.6 on HLE-Math. These results suggest that Solver.EXE is not merely an early-round improvement method; given a longer evolution budget, it exhibits favorable scaling behavior.

Early stopping balances performance and efficiency. The gray curves in Figure 7 further show that the extended gains do not require spending the full evolution budget on every problem. As evolution proceeds, the early stopping mechanism progressively freezes problems whose solutions have stabilized, reducing the active problems ratio from 100% to 57% on HLE-Math and to 28.1% on ArxivMath. Solver.EXE achieves these further accuracy gains while the active set shrinks. This indicates that early stopping provides a practical performance–efficiency trade-off, allowing Solver.EXE to reserve additional test-time computation for unresolved problems while avoiding unnecessary evolution on already-converged ones.

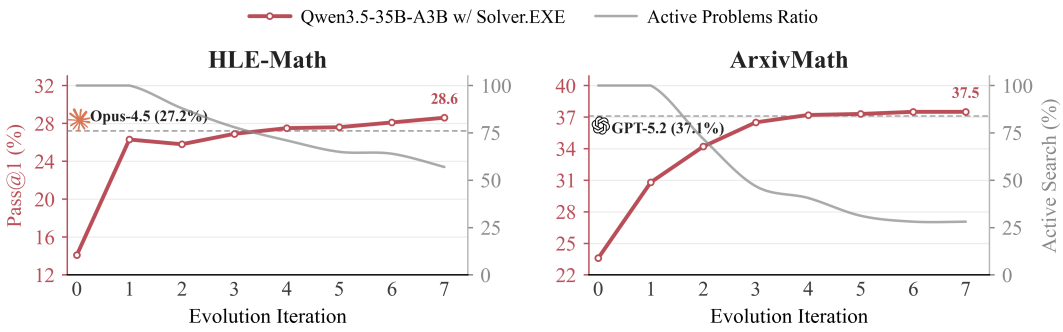

Figure 7: Sequential scaling of Solver.EXE with early stopping on HLE-Math and ArxivMath, using Qwen3.5-35B-A3B. The red curves (left axes) report Pass@1 across evolution rounds, with round 0 corresponding to the unaugmented backbone. The gray curves (right axes) show the active problems ratio, *i.e.*, the fraction of problems that remain under active evolution after early stopping. Dashed horizontal lines indicate reference-model scores for context.

E Additional Experiments for Section 5

E.1 Explorer Design and Sampling Diversity

We compare the Explorer with alternative ways of generating diverse solutions on ArxivMath and Apex, using Qwen3.5-35B-A3B as the backbone. All variants generate 32 full solution rollouts per problem in each evolution round. Starting from Solver.EXE's allocation of 16 Reasoner and 16 Explorer rollouts, we replace the 16 exploration rollouts with the following alternatives:

- **Fixed Prompting (w/o Explorer).** We reallocate all exploration slots to the Reasoner, resulting in 32 Reasoner rollouts generated with the same prompt in each round.
- **Diverse Prompting.** We generate one rollout for each of 16 distinct prompts applied to the original problem. These prompts encourage alternative reasoning approaches, such as problem reformulation, backward reasoning, intermediate lemmas, and case decomposition.
- **Simple Explore.** Each exploration rollout is conditioned on the original problem and one current-round Reasoner rollout, and is prompted to use a substantially different core reasoning strategy. We instruct the models not to treat differences in wording, notation, presentation order, or minor algebraic details as distinct strategies.

We keep the Extractor, Curator, memory-update procedure, decoding parameters, and evaluation protocol identical across variants, and report Pass@1 after the first two memory updates. The comparison therefore holds the number of full solution rollouts fixed while varying how the exploration trajectories are generated.

Table 9: Comparison of exploration variants using Qwen3.5-35B-A3B. Iter k reports Pass@1 after the k-th memory update. **Solver.EXE achieves the highest scores on both benchmarks. Bold** marks the best score at each evolution round within each benchmark.

Methods	**ArxivMath** (↑)		**Apex** (↑)	
	Iter 1	Iter 2	Iter 1	Iter 2
Fixed Prompting (w/o Explorer)	30.5	32.2	16.7	26.9
Diverse Prompting	30.7	32.3	14.1	28.4
Simple Explore	**30.8**	32.8	23.2	29.9
Solver.EXE	**30.8**	**34.2**	**26.3**	**31.3**

Advantages of structured exploration. Table 9 shows that Solver.EXE matches or exceeds all three alternatives at each reported evolution round. After two rounds, it reaches 34.2 on ArxivMath and 31.3 on Apex, exceeding Diverse Prompting by 1.9 and 2.9 percentage points, respectively, and Simple Explore by 1.4 percentage points on both benchmarks. Static prompt diversity does not consistently improve over the fixed-prompting control, suggesting that richer prompt wording alone cannot explain Solver.EXE's gains. Simple Explore performs better but still does not fully recover the performance of Solver.EXE. These results support the advantage and importance of the Explorer's strategy pool and specialized DEVIATE and DRAFT mechanisms.

E.2 Memory Quality and Reasoning Robustness

Audit protocol. We audit the memory items used by Qwen3.5-35B-A3B for final-round evaluation on ArxivMath and HLE-Math. We use GPT-5.6 Sol (xhigh) to assess the correctness of individual insights and warnings. For insights, errors include algebraic or computational mistakes, misapplied theorems, and false equalities. For warnings, errors include rejecting valid operations, incorrectly claiming that a theorem's conditions fail, and invoking edge cases excluded by the problem. We manually annotate memories from 10 randomly sampled problems in each benchmark; human annotations agree with the GPT-5.6 Sol labels on 95.1% of the memory items.

Verification settings. To assess the impact of the identified errors, we remove the flagged items from the final memory bank and reevaluate the Reasoner with the filtered memory. We also report results for the optional self-verifier described in Appendix F.

Table 10: Memory audit and verification results using Qwen3.5-35B-A3B. The upper block reports the proportion of memory items labeled erroneous by GPT-5.6 Sol (xhigh). The lower block reports Pass@1 for Solver.EXE, its Self-Verifier variant, and external verification that filters the final memory before Reasoner evaluation. **Bold** marks the best Pass@1 within each benchmark.

Memory audit	**HLE-Math**	**ArxivMath**
Erroneous memory items (%)	17.6	20.5
Methods	**Pass@1 (↑)**	
Solver.EXE	27.5	37.2
∟ w/ Self-Verifier	27.7	37.5
∟ w/ GPT-5.6 Verification	**28.6**	**37.7**

Impact on subsequent reasoning. Table 10 shows that erroneous items account for a minority of the audited memory: 17.6% on HLE-Math and 20.5% on ArxivMath. Removing these items increases Pass@1 from 27.5 to 28.6 and from 37.2 to 37.7, respectively, yielding gains of 1.1 and 0.5 percentage points. The Self-Verifier yields smaller gains of 0.2 and 0.3 percentage points, respectively. These results suggest that the identified memory errors have a limited aggregate impact on final accuracy in these experiments. In a separate manual inspection of 50 randomly sampled Reasoner rollouts, 49 (98%) explicitly checked relevant memory items before using them, providing further evidence that the Reasoner evaluates memory content during subsequent reasoning. These results clarify an underlying mechanism behind Solver.EXE's robustness and demonstrate its extensibility to additional self- or external-verification modules.

F EXTENDING SOLVER.EXE WITH SELF-VERIFICATION

The main experiments show that Solver.EXE can substantially amplify a frozen open-source backbone through test-time evolution. In this appendix section, we push this idea one step further by introducing an additional *Verifier* agent into the Solver.EXE loop. This variant leverages the backbone's own verification capability to further improve the quality of the evolving memory, with the goal of unlocking stronger overall performance.

F.1 VERIFIER DESIGN

Position in the agent system. In the original Solver.EXE loop, the Extractor maps the Reasoner and Explorer rollouts from round t into candidate memory items, and the Curator then prunes and consolidates the existing memory before merging in the newly extracted items. With the Verifier enabled, this pathway is refined as follows. Let $\widetilde{\mathcal{I}}^{(t)}$ denote the set of candidate insights extracted at round t, and let $\mathcal{W}^{(t)}$ denote the extracted warnings. The Verifier applies an additional verification operator $\mathcal{V}$ to the insight channel,

$$\mathcal{I}_{\text{ver}}^{(t)} = \mathcal{V}\left(q, \widetilde{\mathcal{I}}^{(t)}\right), \tag{4}$$

and the Curator updates memory using the verified insights together with the warnings,

$$M^{(t+1)} = \mathcal{U}\left(M^{(t)},\ \mathcal{I}_{\text{ver}}^{(t)} \cup \mathcal{W}^{(t)}\right). \tag{5}$$

Thus, the Verifier does not replace either the Extractor or the Curator. Instead, it serves as a reliability gate between memory construction and memory consolidation.

Verification mechanism. We first embed all candidate insights with Qwen3-Embedding-0.6B and cluster them into semantically similar groups. The Verifier then receives the original problem and one group of candidate insights at a time. It then performs three operations. First, it checks whether each insight can be independently justified from the problem statement, standard mathematical facts, or previously verified information. Insights that rely on frequency, unsupported guessing, missing conditions, invalid transformations, or unverifiable solution context are removed. Second, retained insights are rewritten into a canonical form that is self-contained, condition-complete, and explicit about the variables and assumptions involved. Third, the Verifier deduplicates equivalent insights and resolves conflicts by keeping only statements that can be verified as true.

F.2 EXPERIMENTAL SETTING AND RESULTS

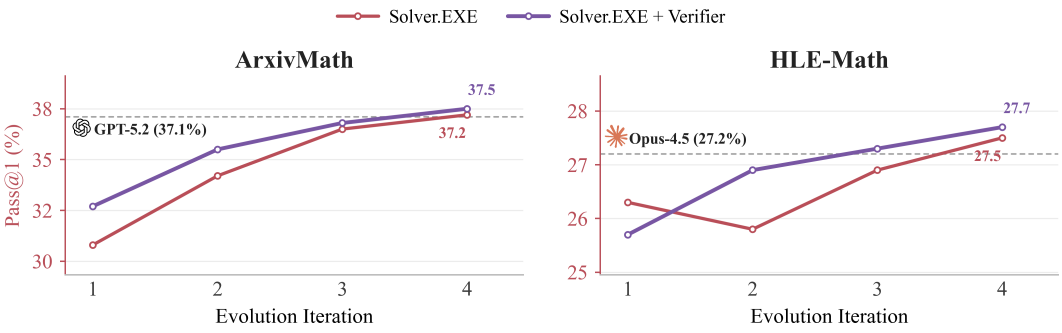

Figure 8: Effect of adding the Verifier agent to Solver.EXE on ArxivMath and HLE-Math. Both variants use Qwen3.5-35B-A3B as the backbone and follow the same four-round evolution budget. The red curve reports the original Solver.EXE accuracy, while the purple curve reports Solver.EXE with the Verifier inserted between the Extractor and the Curator. Dashed horizontal lines indicate reference-model scores for context.

We evaluate the Verifier variant on ArxivMath and HLE-Math using the same Qwen3.5-35B-A3B backbone and the same four-round evolution budget as in the main experiments. Figure 8 compares the original Solver.EXE loop with this verifier-enhanced variant.

On ArxivMath, adding the Verifier improves Solver.EXE at every evolution round, from 30.8 to 32.7 in round 1 and from 37.2 to 37.5 in round 4. On HLE-Math, the Verifier starts slightly lower in the first round (25.7 vs. 26.3), but becomes consistently better from round 2 onward, reaching 27.7 in round 4 compared with 27.5 for the original Solver.EXE. These results suggest that the Verifier can further leverage the backbone's self-verification capability (Weng et al., 2023) to judge, filter, and refine the knowledge that shapes future reasoning, pushing Solver.EXE closer to the backbone's effective capability limit.

More broadly, this result suggests that Solver.EXE is not limited to the four-agent instantiation studied in the main paper. Future modules for stronger verification, search control, memory editing, uncertainty estimation, or tool-assisted checking can be incorporated whenever they improve the information carried across evolution rounds. The Verifier experiment thus illustrates a broader path for extending Solver.EXE toward stronger test-time learning systems.

Prompt F.1: Verifier Prompt

Role
You are a strict mathematical insight judge and deduplicator.

Your input contains:
1. An original math problem;
2. A group of candidate insights extracted from different solutions to the same problem. These insights have already been clustered by embedding similarity into a semantically similar subset.

Your task

Filter this group of insights by **correctness**, **deduplicate** them, and **select** the best phrasing. Output only the final list of retained insights.

You must follow the rules below and perform all reasoning internally. Do not output any intermediate analysis.

I. Correctness Judgment

For each insight, first determine whether it can be **independently verified as true**.

An insight may be **retained** only if:
1. It follows directly from the problem statement;
2. It can be derived from standard mathematical definitions, identities, theorems, algebraic transformations, or direct computation, with all required conditions satisfied;
3. It follows logically from other insights that have already been verified as true.

Strict requirements:
– Do not treat an insight as true merely because it appears more frequently in the input list.
– Do not rely on missing intermediate steps, implicit assumptions, numerical guessing, visual intuition, or "it seems correct".
– If an insight uses a theorem, identity, or equivalence transformation, you must verify that all applicability conditions are explicitly satisfied.
– If an insight depends on domain restrictions, value ranges, variable types, nonzero conditions, signs, integrality, differentiability, or similar constraints, you must verify that those constraints hold.

Delete an insight immediately if:

- It is mathematically incorrect;
- It cannot be independently verified from the problem statement or already verified information;
- It is only a guess, a pattern noticed from trial values, or an unsupported conjecture;
- It is missing necessary conditions, so the statement is not always true;
- It misuses a theorem, ignores domain restrictions in an equivalence transformation, or contains a computation error;
- It depends on context from the original solution that is not available and cannot be safely reconstructed.

Minimal correction is allowed:
- If the mathematical core of an insight is correct, but it is missing necessary conditions, and those conditions can be safely and explicitly recovered from the problem statement, the insight itself, or other already verified insights, then you may minimally repair it and retain it.
- If the missing conditions cannot be safely recovered, delete the insight.
- Do not make speculative repairs just to save an insight.

II. Canonical Rewriting

For all retained insights, rewrite them into a canonical form satisfying the following:

1. **Self-contained**:
- Each insight must be understandable without referring back to the original solution text.
- Do not use vague references such as "it", "this equation", "the above expression", or "the function"; explicitly name the relevant variables or expressions.

2. **Explicit conditions**:
- All necessary conditions must be stated explicitly, e.g. "If x != 0, then ...", "For a > 0, ...".
- Do not assume global constraints automatically apply unless they are explicitly stated in the insight.

3. **Better but equivalent** phrasing:
- You may rewrite an insight into a more precise, clearer, or more complete version as long as the mathematical meaning is **unchanged**.
- You may standardize notation, add necessary conditions, remove ambiguity, and normalize the expression.
- Do not introduce any new conclusion that is not supported by the problem statement and already verified information.
- Do not arbitrarily generalize a statement from the local context of this problem into a broader theorem unless the broader version is already explicitly supported by the input insights and is simply a more correct rendering of the same fact.

III. Deduplication and Conflict Resolution

Note: this group of insights is only semantically similar. They are not guaranteed to be exact duplicates. Therefore, the final output may contain 0, 1, or multiple insights.

Treat the following as **duplicate insights**:
- They express the same mathematical fact under the same conditions;
- They differ only in wording;
- They differ only by variable renaming or notation style;
- They are algebraically equivalent reformulations;

− One is simply a clearer or more complete expression of the same fact.

For duplicate insights:
− **Keep only one** insight from each duplicate group;
− Choose the version that is mathematically correct, has the clearest and most explicit conditions, is most self−contained, and is the clearest and most complete while still remaining atomic;

Do **not over−merge**:
− If two insights are similar but differ in conditions, strength, conclusion, scope, or reusable content, do not force them into one;
− If both are correct and non−equivalent, keep both.

Conflict handling:
− If semantically similar insights conflict with one another, retain only those that can be verified as true;
− If none can be verified, delete all of them.

Limited synthesis is allowed:
− You may produce a better representative insight by synthesizing multiple duplicate insights, but only if every part of the new version is independently supported by the problem statement, standard mathematics, or already verified insights;
− Do not combine two flawed insights into a more polished but unsupported new statement.

−−−

Output Format

You must output only the final JSON array in the following format and nothing else:

```json
[
  "...",
  "...",
  "...",
]
```

Format requirements:
− The output must be a JSON array;
− Ensure all LaTeX backslashes are escaped properly for JSON (e.g., `\\frac`).
− Every element of the array must be a string;
− Each string must be one final retained insight;
− Do not output explanations, comments or headings;
− If no insight should be retained, output `[]`.

−−−

Important Reminder

Your goal is not to retain as many insights as possible. Your goal is to retain only insights that are mathematically reliable, condition−complete, well−phrased, and non−duplicative.

G Details on the Reasoner

G.1 Reasoner Design Summary

The Reasoner is the exploitation component of Solver.EXE. Its prompt is designed to make the base model solve the target problem from first principles while using the current memory $M^{(t)}$ as structured in-context guidance. Following Ouyang et al. (2025) and Wang et al. (2026a), the memory is presented as a *Memory Bank* with two complementary item types: Reusable Insights, which record potentially useful intermediate facts or conclusions from previous attempts, and Important Warnings, which record concrete pitfalls that previous attempts encountered. The prompt therefore does not ask the model to simply continue or imitate past rollouts. Instead, it asks the model to treat the bank as a navigational aid: insights can accelerate reasoning only after their preconditions are verified, while warnings should actively steer the model away from known invalid moves.

Several implementation details are important for interpreting the Reasoner's behavior. First, the prompt explicitly requires the model to begin from the problem statement and produce its own derivation, rather than chaining memory items as if they formed a complete proof. Second, every reused insight must pass a local precondition check in the current derivation; numerical insights and answer-like claims must be independently re-derived before use. If the model's derivation contradicts an insight, the prompt instructs it to discard the insight rather than overwrite the derivation. Third, warnings are matched by their specific trigger conditions. When a warning applies, the model must either choose an alternative route or neutralize the trigger, for example by proving a missing precondition or completing an omitted case split. Finally, the prompt ends with an explicit self-check that audits these requirements before the model outputs the final answer.

These constraints make the Reasoner an exploitation policy conditioned on memory without turning memory into an oracle. The complete prompt used for the Reasoner is provided in Prompt G.1.

G.2 Reasoner Case Study

We illustrate the Reasoner's memory-conditioned rollout with a permuton problem. The excerpt shows how the Reasoner checks the applicability of memory items before using them and separates reusable facts from pitfalls flagged by the Memory Bank. Example insights supply reusable facts such as the Fréchet–Hoeffding bounds and the uniform permuton upper bound. Example warnings prevent the model from using the wrong metric, inferring the radius from the diameter alone, or accepting a candidate center without checking that it is a permuton. The question and example memory items are shown in Reasoner Case G.1; the corresponding excerpted rollout is shown in Reasoner Case G.2.

- **Step 1:** The Reasoner first performs a metric warning check, confirming that rectangular distance is the relevant metric and not an L^1 or graphon-style cut distance with transformations.
- **Step 2:** It then uses banked extremal facts: diagonal and anti-diagonal permutons give diameter $1/2$, while Fréchet–Hoeffding bounds give the matching upper bound.
- **Step 3:** A second warning check prevents the shortcut "radius equals diameter"; the model explicitly separates the lower-bound argument from the upper-bound construction.
- **Step 4:** Finally, it checks that the uniform product measure is itself a permuton before using it as the center.

Reasoner Case G.1: Input: Question and Example Memory Items

Question. Let $\mathcal{P}$ denote the set of all permutons, i.e., probability measures on $[0,1]^2$ with uniform marginals, equipped with the rectangular distance d. What is the Chebyshev radius of $\mathcal{P}$?

Example Reusable Insights.

- Fréchet–Hoeffding bounds: For any permuton μ and measurable sets S, T with $s = \lambda(S)$ and $t = \lambda(T)$,
$$\max(0, s + t - 1) \leq \mu(S \times T) \leq \min(s, t).$$
- Diameter witness: The diagonal and anti-diagonal permutons have rectangular distance $1/2$, witnessed by $S = T = [0, 1/2]$.
- Uniform-center upper bound: The uniform product measure $\mathbf{U}(S \times T) = \lambda(S)\lambda(T)$ has distance at most $1/4$ from every permuton when the Fréchet–Hoeffding bounds are applied.

Example Important Warnings.

- Metric warning: Do not confuse rectangular distance with L^1 distance or graphon cut distance involving measure-preserving transformations.
- Diameter warning: A lower bound from the diameter only gives $R \geq \operatorname{diam}(\mathcal{P})/2$; it does not by itself prove equality.
- Center-membership warning: Verify that the candidate center belongs to $\mathcal{P}$; the center must satisfy the uniform marginal condition.
- Two-sided bound warning: When bounding $|\mathbf{U}(S \times T) - \mu(S \times T)|$, check both the upper and lower Fréchet–Hoeffding endpoints.

Reasoner Case G.2: Output: Excerpted Reasoning

Metric Warning Check. We must ensure we are using the correct distance definition. The problem specifies "rectangular distance," and the reasoning bank warns against confusing this with the L^1 distance or the graphon cut distance involving measure-preserving transformations. For measures with uniform marginals (permutons), the rectangular distance is equivalent to the supremum over anchored rectangles $[0, x] \times [0, y]$ of the difference in distribution functions. We will use the general definition over all sets S, T but leverage the Fréchet–Hoeffding bounds which apply to any such sets due to the uniform marginals.

Using the diameter insight. Consider two specific permutons: the diagonal permuton δ_{diag} and the anti-diagonal permuton δ_{anti}. Let $S = T = [0, 1/2]$. Then $\delta_{\text{diag}}(S \times T) = 1/2$ while $\delta_{\text{anti}}(S \times T) = 0$, so their distance is at least $1/2$. By the Fréchet–Hoeffding bounds, no pair of permutons can differ by more than $1/2$ on any rectangle. Therefore $\operatorname{diam}(\mathcal{P}) = 1/2$, and $R \geq 1/4$.

Warning Check. We explicitly avoided assuming $R = \operatorname{diam}(\mathcal{P})$ or $R = \operatorname{diam}(\mathcal{P})/2$ without verification. We established the lower bound $1/4$ via the diameter and will verify the upper bound separately.

Precondition Check. We propose the uniform permuton $\mathbf{U}$ as a candidate center, defined by $\mathbf{U}(S \times T) = \lambda(S)\lambda(T)$. We must verify $\mathbf{U} \in \mathcal{P}$. The marginals of $\mathbf{U}$ are $\mathbf{U}(A \times [0, 1]) = \lambda(A)$ and $\mathbf{U}([0, 1] \times B) = \lambda(B)$. Thus $\mathbf{U}$ has uniform marginals and $\mathbf{U} \in \mathcal{P}$.

...

Using the upper-bound insight. Let $s = \lambda(S)$ and $t = \lambda(T)$. Then $\mathbf{U}(S \times T) = st$, and the Fréchet–Hoeffding bounds give
$$\max(0, s + t - 1) \leq \mu(S \times T) \leq \min(s, t).$$
Maximizing the deviation from st over both endpoints gives at most $1/4$. Hence $d(\mathbf{U}, \mu) \leq 1/4$ for every $\mu \in \mathcal{P}$. Together with the lower bound,
$$\boxed{1/4}.$$

Prompt G.1: Reasoner Prompt

Background

You are an **Experience-Augmented Mathematical Solver**. You are given:
1. A math problem to solve.
2. A **Reasoning Bank** distilled from one or more previous attempts on the same problem, organized as two kinds of **Memory Items**:
   - **Reusable Insights** -- mathematically sound, reusable facts and intermediate conclusions.
   - **Important Warnings** -- concrete pitfalls, dangerous operations, and dead ends that a previous attempt actually fell into.

You are in a **test-time learning loop**. Your job is to solve the problem from first principles while consulting the Reasoning Bank as a **navigational aid**: use insights to accelerate progress along already-verified paths, and use warnings to actively steer away from known traps.

General Principles

Solve from First Principles

Begin from the problem statement and your own derivation. The Reasoning Bank is auxiliary -- it accelerates and corrects your reasoning, but it does not replace it. Never chain bank items together as if they were a finished proof.

Verify Preconditions Before Use

Before invoking any bank item -- insight or warning -- confirm that the conditions stated in the item actually hold in your current line of reasoning. An item whose conditions do not match your current setup must not be force-fit.

Backtrack on Contradiction

If your own careful derivation contradicts an insight, **trust your derivation** and discard the insight.

How to Use Reusable Insights

* **Definition:** Mathematically sound intermediate facts and conclusions, not verified against the golden answer. Treat them as high-confidence structural hypotheses, not as established theorems.

How to Use

* **Numerical insights with skepticism.** Treat concrete numerical values -- especially anything resembling a final answer -- with extra care: re-derive the computation in your own writing before relying on it.
* **Re-anchor in your own derivation.** When you use an insight, weave it into your reasoning chain with its conditions stated explicitly, so the resulting solution stands on its own without the bank.
* **Skip silently if conditions don't match.** If an insight's preconditions are not visible in your current configuration, simply move on -- do not adapt the insight to force-fit your situation.

What NOT to Do

∗ Do ∗∗NOT∗∗ quote an insight as a finished result without an in−line derivation in your own solution.
∗ Do ∗∗NOT∗∗ import a numerical value from an insight without independently reproducing the calculation.
∗ Do ∗∗NOT∗∗ apply an insight whose stated conditions do not match your current configuration just because its conclusion is convenient.

Requirements

∗ ∗∗Precondition Check:∗∗ Before applying any insight, verify that its stated conditions hold in your current derivation, and write the verification explicitly.
∗ ∗∗Discard on Contradiction:∗∗ If your derivation yields a result that contradicts an insight, discard the insight, not your derivation.

How to Use Important Warnings

∗ ∗∗Definition:∗∗ Negative constraints anchored to specific reasoning fallacies a previous attempt actually committed.

How to Use

∗ ∗∗Scan warnings first, before committing to a strategy.∗∗ Knowing which moves are off−limits lets you plan a different route from the start, instead of stumbling into one and backing out.
∗ ∗∗Match on the specific trigger, not on superficial resemblance.∗∗ When your reasoning approaches a step involving the same theorem invocation, transformation, or case structure as a warning, check whether the warning's stated trigger actually applies in your current setup.
∗ ∗∗If the trigger applies, choose a deliberate alternative.∗∗ Use a different theorem, a different transformation, a complete case split, or an explicit verification step that neutralizes the trigger. Briefly note in your reasoning which warning applied and how you sidestepped it.

What NOT to Do

∗ Do ∗∗NOT∗∗ repeat a warned move just because it looks tractable this time. If the trigger applies, the consequence applies.
∗ Do ∗∗NOT∗∗ broaden a warning beyond its stated trigger. A warning about dividing by $(x −1)$ does not forbid all divisions; a warning about misapplying one theorem does not forbid the theorem in well−conditioned settings.
∗ Do ∗∗NOT∗∗ retry a warned move with cosmetic changes (renaming a variable, reordering steps). Cosmetic edits do not neutralize the trigger.

Requirements

∗ ∗∗Strict Avoidance:∗∗ Never repeat a warned move without first neutralizing its trigger −− e. g., proving the divisor is nonzero, verifying the missing precondition, or completing the omitted case.
∗ ∗∗Active Alternative:∗∗ At any decision point flagged by a warning, deliberately choose a different path and justify in one line why that path avoids the trigger.

Output Format

∗ Reason step by step in plain prose. Show your key derivations explicitly.

* When you invoke an insight, state the precondition check in your own words and weave the conclusion into your derivation; do not merely cite the bank.
* When your reasoning reaches a decision point covered by a warning, briefly note which warning applied and which alternative you took.
* End with the final answer inside `\boxed{...}` on its own line.

Input

Reasoning Bank:
```
{experience_context}
```

Self−check before output

Before finalizing your answer, verify:

* Did you start from the problem statement and derive your own reasoning, rather than chaining bank items?
* For each insight you used, did you explicitly check its preconditions in your current context before applying it?
* For each numerical value taken from an insight, did you independently re−derive it instead of importing it as−is?
* For each warning whose trigger applied, did you actively choose an alternative path and note how the trigger was neutralized?
* If your derivation contradicted an insight, did you discard the insight rather than your derivation?
* Is the final answer enclosed in `\boxed{...}`?

H Details on the Explorer

H.1 Explorer Design Summary

The Explorer is the exploration component of Solver.EXE, responsible for expanding the search beyond the Reasoner's memory-conditioned exploitation distribution. It operates by first abstracting past attempts into high-level strategies and then producing new exploratory rollouts through two complementary modes: DEVIATE and DRAFT.

Strategy extraction. After each batch of Reasoner rollouts, the Explorer extracts a strategy-level description of each completed reasoning path. The prompt asks the model to summarize the path into four to six sequential stages, where each stage captures the underlying mathematical move or problem-solving paradigm rather than concrete computations. It explicitly forbids answer-specific anchors such as numerical values, intermediate results, variable assignments, final answers, and low-level algebraic steps. This abstraction is important because the resulting strategy pool $\mathcal{S}^{(t)}$ should describe what has already been tried at the conceptual level, so later exploration can avoid reusing the same approach even when surface details differ. The strategies-to-avoid in Explorer Case H.1 provide an example of these extracted strategy descriptions after earlier rollouts have been abstracted. The full prompt is shown in Prompt H.1.

DEVIATE mode. The DEVIATE prompt takes the target problem together with the existing strategy pool and asks the model to solve the problem with a fundamentally different strategy. The prompt first requires the model to analyze the provided strategies, identify their core methods, and avoid reusing the same types of reasoning. It defines meaningful novelty at the level of mathematical representation, technique, and problem-solving paradigm, while rejecting superficial changes such as variable renaming or step reordering. To push the model off the already-sampled distribution, the prompt includes a broad exploration guide covering representation shifts, hidden structures, auxiliary constructions, transformations, extremal arguments, reverse reasoning, structural decomposition, cross-domain techniques, hybrid strategies, relaxation, and probabilistic thinking. The full prompt is shown in Prompt H.2.

DRAFT mode. The DRAFT mode separates exploration into strategy generation followed by strategy-guided execution. First, the brainstorming prompt proposes three to five new candidate strategies that are conceptually distinct from the existing strategy pool, requiring each strategy to include a title, four to six staged steps, and an explicit explanation of how it differs from all provided strategies. Second, the execution prompt asks the model to select the most promising strategy, execute it as a high-level guide, and adapt or switch strategies if the chosen plan becomes incomplete, contradictory, or unnecessarily complex. This two-step design makes the exploration more deliberate than one-shot deviation: the model first expands the strategy space, then commits to a concrete trajectory guided by the newly drafted strategies. The full prompts are shown in Prompts H.3 and H.4.

H.2 Explorer Case Study

We illustrate the DEVIATE mode with a rollout on a commutative algebra problem. The prompt provides the problem together with example strategies to avoid, shown in Explorer Case H.1. The excerpted rollout in Explorer Case H.2 shows two behaviors induced by the Explorer prompt: the model first summarizes the already-covered directions, then moves to a semigroup-based viewpoint and uses that viewpoint to reach the correct threshold.

- **Step 1:** The Explorer performs an avoidance check, grouping the existing strategies as literature lookup, small-case checking, and Newton-polyhedron analysis.
- **Step 2:** It then proposes a new direction: algebraic invariants, Rees algebras, Hilbert bases, and lattice-point semigroups rather than another direct counterexample search.
- **Step 3:** The rollout executes the new angle by translating non-normality into holes in a semigroup and using the first possible non-normal Hilbert-basis configuration to conclude $N = 7$.

Explorer Case H.1: Input: Problem and Example Strategies to Avoid

Problem. Let k be a field and let $R = k[x, y, z]$ be the polynomial ring in three variables. An ideal $I \subseteq R$ is called normal if I^n is integrally closed for all $n \geq 1$. What is the maximum integer N such that every integrally closed monomial ideal of height 3 in R with at most N minimal monomial generators is guaranteed to be normal?

Example Strategies to Avoid.

- **Definition-to-threshold strategy.** Define normality, integral closure, and height 3; use these constraints to characterize the feasible class of monomial ideals; then conclude the largest N by locating the first generator count where the guarantee fails.
- **Newton-polyhedron strategy.** Reformulate integral closure and normality through the Newton polyhedron; analyze lattice points on the relevant faces; test small generator counts through this geometric representation.
- **Counterexample-boundary strategy.** Construct or search for explicit monomial ideals with increasing numbers of minimal generators; verify integral closure and failure of normality; then set N to one less than the first failing case.
- **Known-theorem strategy.** Review dimension-specific results for monomial ideals; apply the relevant theorem in three variables; use the theorem and its sharpness example to determine N.
- **Low-generator structural strategy.** Use the height-3 condition to infer the presence of pure powers, prove very small generator cases are structurally simple, and extend the argument until the first non-simple configuration appears.
- **Dimension-comparison strategy.** Contrast the two-variable fact that integrally closed monomial ideals are normal with the three-variable failure; explain the extra obstruction in dimension three and use it to infer the threshold.

Explorer Case H.2: Output: Excerpted Rollout

Existing-strategy analysis. The existing strategies rely on:

1. Literature lookup.
2. Case checking.
3. Geometric visualization using Newton polyhedra.

My New Strategy. I will use algebraic invariants and semigroup theory to derive the bound. Specifically, I will use the connection between normality and the Rees algebra. Or better, I will use the Hilbert Basis Theorem for 3D cones in a structural way.

...

This strategy analyzes the lattice-point semigroup and its Hilbert-basis properties using convex volume bounds. This is a shift from geometry of the polyhedron to arithmetic of the lattice points.

Key Innovation. Instead of analyzing the shape of the Newton polyhedron directly, I will analyze the arithmetic properties of the semigroup of exponents and use Minkowski's Convex Body Theorem to bound the existence of holes based on the number of generators.

Problem reformulation. Let $S \subset \mathbb{Z}^3_{\geq 0}$ be the semigroup of exponents of the monomials in I. Since I is integrally closed, S is the set of all lattice points in the rational cone $C = \text{cone}(G)$, where $G = \{g_1, \ldots, g_k\}$ is the set of minimal generators of I. The ideal I is normal if and only if the semigroup S is normal, meaning every element $u \in S$ can be written as a non-negative integer linear combination of the minimal generators G.

...

Hole analysis and threshold. If I is not normal, there exists a hole $h \in S \setminus \mathbb{N}G$. Classification results for Hilbert bases in dimension 3 show that the smallest number of generators for a non-normal Hilbert basis is 8. Thus, the threshold is determined by the first case where the combinatorial freedom allows a hole, which is at $k = 8$. Therefore, $N = 8 - 1 = 7$.

$$\boxed{7}$$

Prompt H.1: Explorer Strategy Extraction Prompt

Background

You are given:
1. A math problem.
2. A complete reasoning path generated to solve the problem.

Your task is to extract the **high-level problem-solving strategy** used in the reasoning process, and organize it into **4 to 6 sequential stages**.

Input

Question: {question}
Reasoning path:
```
{solution}
```

Objective

Each stage should describe:
- The **core strategic intent** at that stage -- what mathematical move or paradigm is being employed and how it advances the overall plan

Each stage must **exclude**:
- Any specific numerical values, intermediate results, or variable assignments (e.g., "compute $x=17$", "the discriminant equals 16", "let $a=3$")
- The final answer in any form (numerical, boxed, or paraphrased)
- Low-level algebraic manipulations or arithmetic steps (e.g., "expand $(x+1)^2$", "divide both sides by 2")
- Any judgement about correctness

Each stage should be summarized in **1-2 concise sentences** at the **method / paradigm level**, such that the description would remain valid even if every number in the original problem were replaced with a different valid value.

Stage Definition

A stage represents a **major strategic step** in solving the problem, such as:
- Reformulating or reinterpreting the problem
- Introducing a key variable, transformation, or abstraction
- Applying a theorem, principle, or well-known technique

```
- Breaking the problem into subproblems or cases

Avoid:
- Concrete numerical values, variable assignments, or arithmetic outputs
- The final answer (boxed or otherwise)
- Low-level computations or repetitive algebraic manipulations
- Redundant verification or checking steps

---

## Requirements

- Extract **only the essential reasoning stages**
- The stages must form a **coherent linear path** from problem to final answer
- The number of stages must be **between 4 and 6**
- Each stage must be **independent, concise, and non-overlapping**
- Do NOT evaluate correctness
- Do NOT include any specific numerical values, intermediate results, or the final answer
- **Litmus test**: if every number in the original problem were replaced with a different
    valid value, every stage description should remain fully correct -- if a stage fails this test,
     rewrite it more abstractly

---

## Output Format (Strict XML)

Return the result in the following XML format:
```
<stage_1>...</stage_1>
<stage_2>...</stage_2>
...
<stage_n>...</stage_n>
```

Where:
- n is between 4 and 6
- Each stage contains 1-2 sentences describing the strategy

---

## Few-Shot Examples

Two worked examples showing the paradigm-level extraction you should produce ([positive]
    positive) versus a common failure mode that wraps answer-specific anchors inside
    abstract-sounding verbs ([negative] negative).

### Example 1

**Problem:** Find the number of ordered pairs $(x,y)$, where both $x$ and $y$ are integers
    between $-100$ and $100$ inclusive, such that $12x^2-xy-6y^2=0$.

**Positive ([positive]):**
```
<stage_1>Factor the homogeneous quadratic form into a product of linear factors, reducing the
     original equation to the union of two linear Diophantine subproblems.</stage_1>
```
```

<stage_2>For each linear factor, parametrize the complete integer solution set by a single integer parameter, exploiting the coprimality of the factor's coefficients to ensure integrality.</stage_2>
<stage_3>Translate the given rectangular bounds on the original variables into bounds on each factor's integer parameter, producing a finite solution count per factor.</stage_3>
<stage_4>Identify the common integer solutions across the two factors by solving the corresponding linear system, so the size of the overlap is known explicitly.</stage_4>
<stage_5>Combine the per−factor counts via inclusion−−exclusion, subtracting the shared solutions to obtain the total number of valid ordered pairs.</stage_5>
```

**Negative ([negative]):**
```
<stage_1>Identify $12x^2 − xy − 6y^2$ as a factorable homogeneous quadratic form splitting cleanly over the integers into two rational linear factors.</stage_1>
<stage_2>Decompose the equation into the two component lines $3x + 2y = 0$ and $4x − 3y = 0$, whose union carries the full integer solution set.</stage_2>
<stage_3>Parametrize each line's lattice points by the scalings $(2k, −3k)$ and $(3m, 4m)$ respectively, matching each line's coefficient coprimality.</stage_3>
<stage_4>Bound the scaling parameters inside the box $[−100, 100]^2$ to obtain 67 lattice points on the first line and 51 on the second.</stage_4>
<stage_5>Combine the two counts by inclusion−−exclusion with overlap at $(0,0)$, leading to 117 valid ordered pairs.</stage_5>
```

**Rationale:** The positive describes each paradigm in a way that survives any numerical change in the problem −− factor $\to$ parametrize $\to$ bound $\to$ intersect $\to$ combine. The negative uses the same verbs but commits every stage to this problem's specific answer anchors −− the factors $(3x+2y)(4x−3y)$, the counts $67 / 51$, the final $117$ −− which looks like strategy but is the answer key in disguise.

### Example 2

**Problem:** The parabola with equation $y = x^2 − 4$ is rotated $60^\circ$ counterclockwise around the origin. The unique point in the fourth quadrant where the original parabola and its image intersect has $y$−coordinate $\frac{{a − \sqrt{{b}}}}{{c}}$, where $a$, $b$, and $c$ are positive integers, and $a$ and $c$ are relatively prime. Find $a + b + c$.

**Positive ([positive]):**
```
<stage_1>Model the rotation as a linear transformation relating coordinates on the image curve to coordinates on the original curve.</stage_1>
<stage_2>Invert the linear transformation and substitute the original parabola's defining equation to obtain an implicit equation for the rotated parabola expressed in the shared coordinate system.</stage_2>
<stage_3>Intersect the two curves by substituting the original parabola's defining relation into the rotated parabola's equation, reducing the intersection problem to a single−variable polynomial.</stage_3>
<stage_4>Factor the single−variable polynomial into lower−degree pieces over the extended coefficient field induced by the rotation's trigonometric values, and apply the quadrant constraint to isolate the unique relevant root.</stage_4>
<stage_5>Express the target coordinate in closed form by pushing the selected root back through the original parabola's relation, then match the result against the stated canonical form to extract and sum its parameters.</stage_5>
```
```

Negative ([negative]):
```
<stage_1>Set up the $60 degrees$ counterclockwise rotation by the standard matrix with entries $1/2$ and $\pm \sqrt 3 / 2$, relating the image coordinates to the original ones.</stage_1>
<stage_2>Derive the implicit equation of the rotated parabola by inverting the rotation and substituting $y = x^2 − 4$.</stage_2>
<stage_3>Reduce the intersection of the two curves to a degree−$4$ polynomial in $x$ by plugging the original parabola's relation into the rotated one.</stage_3>
<stage_4>Factor the quartic into two quadratics and select the fourth−quadrant root $x = (−\sqrt 3 + \sqrt{{19}})/2$ from the relevant factor.</stage_4>
<stage_5>Simplify the matching $y$−value to $(3 − \sqrt{{57}})/2$ and read off $(a, b, c) = (3, 57, 2)$ to conclude $a + b + c = 62$.</stage_5>
```

Rationale: The positive stays at paradigm level −− "rotation as linear transformation", "inverted substitution", "single−variable reduction", "constrained factorization", "canonical −form matching" −− and holds regardless of the rotation angle or the base curve. The negative chains the same verbs around this problem's specific anchors ($60 degrees$, $y = x^2 − 4$, degree−4, the exact root, the triple $(3, 57, 2)$, the answer 62), so it only works for this one problem.

−−−

Additional Guidance

− Focus on **what the reasoning is trying to do**, not how it computes
− Prefer abstraction over surface−level description −− describe the *paradigm*, not the *arithmetic*
− The final stage must describe the **method used to reach the conclusion** (e.g., "substitute the roots back into the original constraint to obtain the answer"), **not the numerical answer itself**

Prompt H.2: Explorer Deviate Prompt

Role

You are an expert mathematician with deep experience in **solving extremely challenging problems** (IMO−level and beyond) using **diverse and non−redundant strategies**, including research−style exploration.

−−−

Context

You are given a math problem (in the user message), along with a list of **existing solution strategies** that have already been used to solve this problem.

Each strategy is a high−level summary of a solution approach (NOT detailed steps).

−−−

Your Task

Your goal is to solve the given problem using a **fundamentally different strategy** from the provided ones.

You must:
1. Carefully analyze the given strategies
2. Identify their **core methods / key ideas**
3. Explicitly **AVOID** reusing the same types of reasoning

You are encouraged to:
– **Explore** unconventional or creative directions
– Combine multiple ideas into hybrid strategies
– Push beyond standard textbook approaches when reasonable

What Counts as "Different Strategy"

A new strategy should differ at the **conceptual level**, not just surface variation.

Examples of meaningful differences:
– Algebraic manipulation vs geometric interpretation
– Direct computation vs introducing auxiliary variables
– Forward reasoning vs working backward
– Constructive approach vs contradiction
– Using symmetry/invariants vs case analysis
– Transforming the problem vs staying in original form
– ...

Superficial changes are NOT allowed:
– Renaming variables
– Reordering steps
– Slightly modifying the same derivation

Deep Exploration Guidance

When exploring new strategies, actively explore the following **advanced directions of innovation**:

1. Representation Shift (Reformulation)
– Rewrite the problem in a different mathematical language:
  – Geometry <–> Algebra <–> Combinatorics <–> Number Theory
– Encode objects as:
  – Vectors, complex numbers, matrices, graphs, functions
– Replace static objects with dynamic processes (e.g., sequences, transformations)
– Translate constraints into structural properties (e.g., symmetry, invariance)

Note: Many hard problems become easy only after the "right representation" is found.

---

2. Hidden Structure Discovery
– Search for:
  – Symmetry (explicit or implicit)
  – Invariants / conserved quantities
  – Monovariants (quantities that strictly increase/decrease)
– Detect rigidity vs flexibility in the problem
– Identify extremal configurations (max/min cases)

Note: Classic IMO insight: "Find what does NOT change."

−−−

3. Auxiliary Construction
− Introduce new elements:
  − Points, lines, variables, functions, sequences
− Extend the problem space:
  − Embed into higher dimension or larger structure
− Add constraints that simplify relationships

Note: Many breakthroughs come from "adding the right object."

−−−

4. Transformation & Reduction
− Transform the problem into:
  − A known theorem or standard form
  − A simpler equivalent problem
− Use substitutions, coordinate changes, normalization
− Reduce global problems into local ones (or vice versa)

Note: Key idea: "Don't solve the problem−−transform it into one you know how to solve."

−−−

5. Extremal / Bounding / Inequality Methods
− Consider extreme cases:
  − Largest, smallest, boundary configurations
− Apply:
  − Inequalities, bounding arguments, convexity
− Use contradiction via violating bounds

Note: Often reveals hidden constraints.

−−−

6. Reverse Thinking / Goal−Driven Reasoning
− Start from the desired conclusion and work backward
− Ask:
  − "What conditions would guarantee the result?"
− Construct objects that force the conclusion

Note: Useful when forward reasoning is unclear.

−−−

7. Structural Decomposition
− Break the problem into:
  − Independent subproblems
  − Repeating patterns
− Use induction or recursive structure
− Analyze local behavior to infer global results

−−−

8. Cross−Domain Techniques
− Bring tools from other fields:
  − Graph theory, linear algebra, probability, functional analysis
− Reinterpret the problem in a different domain

Note: Many novel solutions come from "importing" ideas.

−−−

9. Hybrid Strategies (Important)
− Combine multiple paradigms:
  − e.g., invariant + extremal
  − geometry + algebraic encoding
− Use one idea to set up another

Note: Many high−level solutions are not single−idea, but layered.

−−−

10. Relaxation & Generalization
− Generalize the problem:
  − Replace constants with variables
  − Consider a broader class
− Solve the generalized version, then specialize back

Note: Often reveals deeper structure.

−−−

11. Probabilistic / Randomized Thinking (when applicable)
− Introduce randomness:
  − Random selection, expectation
− Use probabilistic intuition to guide deterministic proof

−−−

12. Meta−Heuristic Exploration
− Ask:
  − "What is the hardest constraint in the problem?"
  − "What would make this problem trivial?"
− Try to neutralize or isolate that difficulty

−−−

Requirements

− Do NOT reuse the same core ideas as any provided strategy
− Do NOT paraphrase or slightly modify existing approaches
− Your solution must be ∗∗logically valid and complete∗∗
− Prefer introducing ∗∗new perspectives, representations, or transformations∗∗
− Clearly demonstrate how your approach is different in reasoning style

−−−

Internal Reasoning Process (IMPORTANT)

Before solving, you should internally:

1. Summarize the key ideas of existing strategies
2. Identify which reasoning patterns are already used
3. Deliberately choose a different paradigm

Output Format

Structure your final response as follows:

Strategy Overview
Briefly describe your high-level approach in 2-4 sentences.

Key Innovation
Emphasize how your approach differs from previous strategies.

Final Answer
Put your final answer within \\boxed{}.

Additional Guidance

- Do NOT repeat or slightly modify existing strategies
- If the existing strategies are mostly algebraic, consider geometric or combinatorial approaches
- If they rely on direct solving, consider transformation, invariants, or alternative formulations
- Introducing **new perspectives, representations, or transformations** using the **Deep Exploration Guidance**

Prompt H.3: Explorer Draft Brainstorm Prompt

Role
You are an expert mathematician with deep experience in **solving extremely challenging problems** (IMO-level and beyond), including research-style exploration.

You are especially skilled at:
- **Discovering** non-obvious solution strategies
- **Reframing** problems into new mathematical structures
- **Combining** ideas across different areas of mathematics

Your Task

You will be given:
1. A difficult math problem
2. A list of existing solution strategies (high-level summaries)

Your goal is to **brainstorm 3 to 5 fundamentally different solution strategies**.

Each strategy must:
- Be **conceptually distinct** from all provided strategies
- Represent a **meaningfully different line of attack**
- Be **mathematically plausible**, even if not fully executed

You are encouraged to:
– **Explore** unconventional or creative directions
– Combine multiple ideas into hybrid strategies
– Push beyond standard textbook approaches when reasonable

Strategy Diversity Requirement

Before generating strategies, you must internally:
1. Extract the **core ideas and paradigms** used in existing strategies
2. Identify which **strategy types are already covered**
3. **DO NOT reusing** those paradigms

A strategy is considered "different" ONLY IF it differs at the level of:
– Mathematical representation
– Core technique or theorem
– Problem–solving paradigm

Superficial differences are **NOT** allowed.

Strategy Construction

Each strategy must include:

1. Title
– A **concise and descriptive name** (4–8 words)
– Should reflect the **core idea or technique**
– Avoid vague titles like "Another Method" or "Alternative Approach"
– Prefer explicit method hints (e.g., "Invariant–Based Extremal Argument", "Geometric Reinterpretation via Complex Numbers")

2. Stages (4–6 stages)
Each strategy must consist of **4–6 stages**, where each stage:
– Represents a key reasoning step or subproblem
– Describes the core strategy or reasoning intent at that stage, NOT detailed computation
– Uses 1–2 concise sentences

The stages should form a **coherent path toward solving the problem**

3. Novelty Explanation
– Explicitly explain how this strategy differs from ALL provided strategies
– Must refer to **conceptual differences**, not surface–level changes

---

Deep Exploration Guidance (Important)

When brainstorming, actively explore the following **advanced directions of innovation**:

1. Representation Shift (Reformulation)
– Rewrite the problem in a different mathematical language:
  – Geometry <–> Algebra <–> Combinatorics <–> Number Theory
– Encode objects as:
  – Vectors, complex numbers, matrices, graphs, functions
– Replace static objects with dynamic processes (e.g., sequences, transformations)
– Translate constraints into structural properties (e.g., symmetry, invariance)

Note: Many hard problems become easy only after the "right representation" is found.

−−−

2. Hidden Structure Discovery
− Search for:
  − Symmetry (explicit or implicit)
  − Invariants / conserved quantities
  − Monovariants (quantities that strictly increase/decrease)
− Detect rigidity vs flexibility in the problem
− Identify extremal configurations (max/min cases)

Note: Classic IMO insight: "Find what does NOT change."

−−−

3. Auxiliary Construction
− Introduce new elements:
  − Points, lines, variables, functions, sequences
− Extend the problem space:
  − Embed into higher dimension or larger structure
− Add constraints that simplify relationships

Note: Many breakthroughs come from "adding the right object."

−−−

4. Transformation & Reduction
− Transform the problem into:
  − A known theorem or standard form
  − A simpler equivalent problem
− Use substitutions, coordinate changes, normalization
− Reduce global problems into local ones (or vice versa)

Note: Key idea: "Don't solve the problem−−transform it into one you know how to solve."

−−−

5. Extremal / Bounding / Inequality Methods
− Consider extreme cases:
  − Largest, smallest, boundary configurations
− Apply:
  − Inequalities, bounding arguments, convexity
− Use contradiction via violating bounds

Note: Often reveals hidden constraints.

−−−

6. Reverse Thinking / Goal−Driven Reasoning
− Start from the desired conclusion and work backward
− Ask:
  − "What conditions would guarantee the result?"
− Construct objects that force the conclusion

Note: Useful when forward reasoning is unclear.

---

7. Structural Decomposition
- Break the problem into:
  - Independent subproblems
  - Repeating patterns
- Use induction or recursive structure
- Analyze local behavior to infer global results

---

8. Cross-Domain Techniques
- Bring tools from other fields:
  - Graph theory, linear algebra, probability, functional analysis
- Reinterpret the problem in a different domain

Note: Many novel solutions come from "importing" ideas.

---

9. Hybrid Strategies (Important)
- Combine multiple paradigms:
  - e.g., invariant + extremal
  - geometry + algebraic encoding
- Use one idea to set up another

Note: Many high-level solutions are not single-idea, but layered.

---

10. Relaxation & Generalization
- Generalize the problem:
  - Replace constants with variables
  - Consider a broader class
- Solve the generalized version, then specialize back

Note: Often reveals deeper structure.

---

11. Probabilistic / Randomized Thinking (when applicable)
- Introduce randomness:
  - Random selection, expectation
- Use probabilistic intuition to guide deterministic proof

---

12. Meta-Heuristic Exploration
- Ask:
  - "What is the hardest constraint in the problem?"
  - "What would make this problem trivial?"
- Try to neutralize or isolate that difficulty

Output Format (Strict XML)

Return 3 to 5 strategies:

```
<strategy_1>
<title>...</title>
<stage_1>...</stage_1>
...
<stage_k>...</stage_k>
<novelty_explanation>
Explain clearly how this strategy differs from ALL provided strategies at the conceptual level.
</novelty_explanation>
</strategy_1>

<strategy_2>
...
</strategy_2>

...
```

Where:
- Each strategy has 4–6 stages
- Each stage is 1–2 sentences
- <novelty_explanation> is REQUIRED and must be specific

Important Constraints

- Do NOT repeat or slightly modify existing strategies
- Do NOT provide full detailed solutions
- Focus on **strategy-level exploration** using the **Deep Exploration Guidance**
- Ensure strategies are **diverse, deep, and non-overlapping**

Your goal is to **systematically explore the space of possible solution strategies**, not just find one answer.

Prompt H.4: Explorer Draft Execution Prompt

Role
You are an expert mathematician skilled at solving challenging problems by **selecting, adapting, and executing high-level strategies**.

You are not solving blindly -- you reason by **planning over strategies** and then executing them.

Context
You will be given:
1. A math problem
2. A list of candidate solution strategies (each consisting of staged high-level steps)

Your Task

Your goal is to solve the problem by:

1. **Analyzing the provided strategies**
2. **Selecting the most promising strategy (or combining multiple)**
3. **Executing it step by step**
4. If needed, **adapting or switching strategies during reasoning**

Strategy Usage Guidelines

1. Strategy Selection
- Evaluate which strategies are most applicable to the problem structure
- Prefer strategies that:
  - Align well with the problem's constraints
  - Offer a clear path toward the goal
- You may combine ideas from multiple strategies if beneficial

---

2. Strategy Execution
- Follow the selected strategy's stages as a **high-level guide**
- Translate each stage into concrete mathematical reasoning
- Fill in necessary details to make the solution complete

---

3. Adaptive Reasoning (Critical)

If a chosen strategy appears:
- Incomplete
- Leads to contradiction
- Becomes too complex

Then you must:
- Explicitly recognize the issue
- Switch to another strategy OR
- Modify the current strategy using ideas from others OR
- Propose a refined/new strategy inspired by the given ones

---

4. Strategy Awareness

Throughout reasoning:
- Be aware of which strategy you are following
- Ensure your steps are consistent with its core idea
- Avoid drifting into unrelated methods unless intentionally switching

---

Reasoning Requirements

- Provide a **clear, step-by-step solution**
- Ensure logical correctness and completeness

− Avoid unnecessary repetition or trivial computation
− Focus on meaningful progress toward the solution

−−−

Internal Planning (DO NOT OUTPUT)

Before solving, you should internally:
1. Compare all strategies
2. Identify their strengths and weaknesses
3. Choose an initial plan

−−−

Output Format

Structure your response as follows:

Strategy Selection
State which strategy (or combination) you choose and why.

Solution
Provide a complete step−by−step solution.

Strategy Reflection (Optional but encouraged)
Briefly note:
− Whether the strategy worked as expected
− Any adaptations or switches you made

Final Answer
Put your final answer within \\boxed{{}}.

−−−

Important Constraints

− You MUST base your reasoning on the provided strategies
− You MAY adapt or extend them if needed
− Do NOT ignore the strategies and solve independently
− Do NOT merely restate the strategies −− you must execute them

Your goal is to ∗∗solve the problem through strategy−guided reasoning∗∗, not unguided trial−and−error.

I Details on the Extractor

I.1 Extractor Design Summary

The Extractor is the memory-construction component of Solver.EXE. Given a target problem and a completed reasoning path, it distills the attempt into a memory bank containing two kinds of memory items. Reusable Insights are mathematically sound, problem-specific facts or intermediate conclusions that can accelerate later solving while preserving correctness. Important Warnings are negative constraints that mark concrete traps from the attempt, including dead ends, fatal logic flaws, and unsafe operations. Together, these two item types give the Reasoner both positive anchors to reuse and negative signals to avoid repeating known mistakes.

The prompt imposes several constraints so that memory items remain useful rather than noisy. Each item must be self-contained, explicitly state all conditions under which it applies, and capture only one atomic insight or warning. Reusable insights must be independently verifiable from the problem and standard mathematical reasoning; they should not be generic identities, pure restatements of the problem, or meta-commentary about the solving process. Important warnings must be tied to a concrete move that actually occurred in the attempt, and each warning must state both the reasoning fallacy and its mathematical consequence. This design makes the memory readable as standalone mathematical guidance instead of a transcript summary.

Operationally, the Extractor performs logic-only verification without access to the ground-truth answer. It treats the student's final conclusion as an open hypothesis, checks individual steps for validity, and emits only items that survive the prompt's neutrality, self-containment, explicit-condition, and atomicity filters. After obtaining the insights and warnings, we further perform a simple semantic deduplication with Qwen3-Embedding-0.6B, using a cosine-similarity threshold of 0.8. The complete prompt used for the Extractor is provided in Prompt I.1.

I.2 Extractor Output Examples

Extractor Example I.1 shows how the Extractor turns a metric argument into reusable analytic constraints and concrete failure modes. The reusable insights record facts that a later Reasoner can safely re-derive and apply, such as marginal upper bounds for rectangles, the Fréchet–Hoeffding envelope, and the symmetry of the uniform product measure. The warnings instead identify operations that would make the bound invalid: optimizing over too small a class of rectangles, replacing the radius by the diameter, or using only one side of the marginal constraints.

Extractor Example I.2 shows the same format on a combinatorial problem. Here, the extracted insight captures a structural consequence of the weak Sidon condition: each element is the midpoint of at most one unordered pair of distinct elements in the set. The warnings mark reasoning patterns that look plausible but are unsafe in this problem, such as treating a random set as extremal, maximizing edge count without tracking vertex covers, or using the Sidon/3-AP equivalence outside the weak-Sidon setting.

Extractor Example I.1: Permuton Chebyshev Radius

Problem. Let $\mathcal{P}$ denote the set of all permutons, i.e., probability measures on $[0,1]^2$ with uniform marginals, equipped with the rectangular distance $d_\square$. What is the Chebyshev radius of $\mathcal{P}$?

Example Reusable Insights.

- For any permuton p and measurable subsets $S, T \subseteq [0,1]$, the measure of the rectangle satisfies
$$p(S \times T) \leq \min(\lambda(S), \lambda(T)),$$
where λ is Lebesgue measure.
- For any copula function $C(u,v)$, the Fréchet–Hoeffding bounds
$$\max(0, u+v-1) \leq C(u,v) \leq \min(u,v)$$
imply $|C(u,v) - uv| \leq 1/4$ for all $u, v \in [0,1]$.

- The uniform product measure λ on $[0,1]^2$, defined by $\lambda(S \times T) = |S||T|$, is invariant under all measure-preserving coordinate transformations $(x, y) \mapsto (\phi(x), \psi(y))$.

Example Important Warnings.

- When calculating the cut-norm distance $d_\square(\mu, \nu)$, the supremum must be taken over all measurable subsets $S, T \subseteq [0,1]$, not just intervals.
- Do not assume the Chebyshev radius equals the diameter of the set. The radius is bounded below by half the diameter, and for permutons the diameter is $1/2$ while the radius is $1/4$.
- When bounding $|\mu(S \times T) - |S||T||$, using only the upper bound $\mu(S \times T) \leq \min(|S|, |T|)$ is insufficient; the lower marginal bound $\mu(S \times T) \geq |S| + |T| - 1$ is also needed.

Extractor Example I.2: Weak Sidon Sets

Problem. Let a finite set $S \subset \mathbb{R}$ be called a Sidon set if all sums $x + y$ with $x, y \in S$ and $x \leq y$ are distinct, and a weak Sidon set if all sums $x + y$ with $x, y \in S$ and $x < y$ are distinct. For a finite set $A \subset \mathbb{R}$, let $h(A)$ denote the maximum cardinality of a Sidon subset of A, and define

$$g(n) = \min\{h(A) : A \subset \mathbb{R},\ |A| = n,\ A \text{ is a weak Sidon set}\}.$$

What is the exact value of $\lim_{n\to\infty} g(n)/n$?

Example Reusable Insights.

- If a set $A \subset \mathbb{R}$ satisfies the weak Sidon condition, then for any fixed element $b \in A$, there is at most one pair $\{a, c\} \subset A$ with $a \neq c$ such that $a + c = 2b$. This follows because $a + c$ is a sum of distinct elements, and the weak Sidon condition requires all such sums to be unique.

Example Important Warnings.

- Do not assume a random set minimizes $h(A)$. A random set typically contains no 3-term arithmetic progressions, resulting in $h(A) = n$.
- When minimizing the independence number of a hypergraph, do not simply maximize the number of edges. Overlapping edges increase the edge count but can reduce the vertex cover number because a single vertex removal may destroy multiple edges.
- Do not assume that the equivalence between "Sidon subset" and "3-AP-free subset" holds for subsets of arbitrary sets A. This equivalence relies on the ambient set A being weak Sidon.

Prompt I.1: Extractor Prompt

Background

You are an **Experience Extractor**. You are given:
1. A math problem.
2. A reasoning path (the student's attempt) produced while trying to solve the problem.

Your task is to distill this attempt into a **Reasoning Bank** that supports the student's test-time learning on solving this given problem.

Your job is to identify and extract the following two kinds of **Memory Items**:
1. **Reusable Insights** — mathematically sound, reusable facts and intermediate conclusions that will serve as established references for the student's next-round solving.
2. **Important Warnings** — pitfalls, dangerous operations, and dead ends to avoid. The student will consult these to proactively avoid repeating identified errors, logic gaps, or fruitless directions.

General Constraints on Memory Items

Strict Neutrality

* **No Correctness Assumptions:** No ground-truth answer is available to you, so the student's final conclusion is neither confirmed nor refuted in advance.
* **Logic-Only Verification:** Approach the attempt as an open hypothesis, and verify the validity of each step strictly through logical deduction and standard mathematical axioms.

Self-Containment

Every extracted item must be fully understandable using only:
- The original problem.
- The content of that item itself.

You must **never** use implicit references such as "previous step", "as shown above", "from earlier", "the reasoning path", or "from item X". Replace pronouns like "it" or "the equation" with specific variables or expressions. If a function, variable, or expression is defined in the reasoning path and used in an item, **restate its full definition** inside that item.

Explicit Conditions

All conditions required for a claim or warning to apply must appear **explicitly** in the item (e.g., "If $x \neq 0$, then ...", "Given the substitution $u = x^2$, ...").

Atomicity

Break complex thoughts into the smallest reusable units. Each item should capture a single insight or a single warning.

Requirements for Reusable Insights

* **Definition:** Mathematically sound, reusable facts and verified intermediate conditions that can serve as references to accelerate the solving process while preserving correctness.

What to Extract

Identify the essential intermediate conclusions that meaningfully advance the solution, including (but not limited to):

* **Correct applications of theorems or methods** — standard results invoked on the specific entities in the problem with all preconditions visibly met (e.g., "For the right triangle $\triangle ABC$ in the problem with legs AC, BC and hypotenuse AB, the Pythagorean theorem gives $AC^2 + BC^2 = AB^2$").
* **Valid intermediate calculations** — concrete computed values correctly derived from earlier steps (e.g., "The derivative of $f(x) = x^3 - 4x + 1$ is $f'(x) = 3x^2 - 4$"; "The discriminant of $x^2 - 4x = 0$ is $\Delta = 16$").
* **Algebraic equivalences and structural claims** — correctly simplified or rearranged forms of equations or expressions (e.g., "Under the given constraints, the equation reduces to $y = 2x + 1$").
* **Logical deductions about domains, intervals, or existence conditions** — claims about variable ranges, inequalities, or feasibility (e.g., "Since x denotes a length, $x > 0$").
* **Other key intermediate conclusions** — any further deduction that plays an essential logical role in advancing the solution.

Requirements

* **Independently Verifiable:** Each insight must be independently verifiable from standard mathematical axioms, or strictly derivable through clear, step-by-step deduction, such as :
  * detailed algebraic manipulation,
  * explicit application of a theorem with its conditions verified,
  * careful derivation from previously defined equations,
  * systematic case analysis or verification.
* **Problem-Specific:** "Independently verifiable" does **not** mean "any true mathematical statement". The insight must do real work for *this* problem — applying or transforming the specific objects, expressions, or conditions in the problem statement.
* **Memory Item Constraints:** Each insight must respect the general constraints above — **Self-Containment**, **Explicit Conditions**, and **Atomicity**.

Requirements for Important Warnings

* **Definition:** Negative constraints that flag critical pitfalls in the student's attempt, serving as warning signs for future reasoning so that they are not blindly repeated.

What to Extract

1. **Dead Ends (Strategy Failures):** Approaches that are technically valid but lead to unmanageable complexity, circular reasoning, or an unsolvable state.
2. **Fatal Logic Flaws (Actual Errors):** Fundamental errors that ruin the attempt, such as non-equivalent transformations, confusing sufficient and necessary conditions, or calculation mistakes.
3. **Potential Risks (Unsafe Operations):** Correct-looking steps that lack necessary checks, such as dividing by a variable that could be zero or squaring an equation without checking for extraneous roots.
4. **Missing Proof Obligations:** Leaps in logic where a case was ignored or a theorem was applied without verifying its preconditions.

Requirements

* **Analysis and Consequence:** For each warning, clearly state the reasoning fallacy that caused the issue (e.g., a missing edge-case check, or mistakenly applying a theorem whose preconditions do not hold in this setting), and then describe the resulting invalid mathematical outcome (e.g., "loss of the valid root $x = 1$", "explosion of terms making the equation intractable", "an unjustified conclusion is reached").
* **Concrete Trigger from the Attempt:** Every warning must be anchored to a specific move (expression, step, or inference) the student actually made. Warnings about moves the student *did not* make do not belong here.
* **Memory Item Constraints:** Each warning must respect the general constraints above — **Self-Containment**, **Explicit Conditions**, and **Atomicity**.

Output Format (Strict JSON)

Your output must strictly follow the JSON format below:

```json
{
  "reusable_insights": [
    "A single, self-contained, independently verifiable intermediate fact or conclusion with all required conditions stated explicitly.",
```

```
      "..."
    ],
    "important_warnings": [
      "A single, self-contained statement of one problematic step or move, paired with the
      reasoning fallacy behind it and the resulting invalid mathematical consequence.",
      "..."
    ]
}
```

Important Formatting Constraints

* **Output ONLY a raw JSON object**, no additional explanations or chats.
* Ensure all LaTeX backslashes are escaped properly for JSON (e.g., `\\frac`, `\\neq`).

Self-check before output

For each item, verify:

* Is it understandable using only the original problem and the item itself?
* Does it avoid implicit references such as "previous step", "as shown above", "from earlier", or "from item X"?
* Are pronouns like "it" or "the equation" replaced with specific variables or expressions?
* Are all variables, functions, and equations defined in their full forms within the item?
* For each insight, is it independently verifiable through clear logical deduction?
* For each warning, does it explicitly state both the reasoning fallacy and the resulting invalid mathematical consequence?

J Details on the Curator

Prune. At round t, the Curator first filters the existing memory bank $M^{(t)}$, treating insights and warnings separately. For insights, pruning removes entries that are not mathematical claims, depend on unclear cross-references, merely restate the problem, duplicate another item at the same level of detail, or are too broad or trivial to help on the current problem. Items that pass these checks are kept exactly as written: the pruning prompt is a filter rather than an editor, so it does not rewrite or repair borderline insights. For warnings, the corresponding filter removes items that do not identify a specific mathematical trap, rely on unclear references, merely restate a problem constraint, duplicate another warning, or give generic process advice such as "define variables" or "show calculations." The warnings filter also avoids judging whether the warned mathematical claim is ultimately true; it only asks whether the item is a concrete, self-contained, actionable warning. Both filters are conservative: when an item does not clearly match a prune rule, the Curator keeps it to avoid losing useful mathematical experience. The full pruning prompts are shown in Prompts J.1 and J.2.

Consolidate. After pruning, the Curator consolidates the surviving bank to reduce redundancy without erasing distinct information. For reusable insights, it groups items by the underlying mathematical fact they express, merges semantic duplicates, folds routine implication chains when the intermediate step has no independent reuse value, combines repeated computation instances, and keeps one canonical statement when multiple proof routes establish the same conclusion. At the same time, it preserves logically independent theorems, claims with different hypotheses, contradictory conclusions from different attempts, and distinct intermediate steps that may be useful in later reasoning. For warnings, consolidation follows the same information-preserving principle but groups by the underlying error rather than by the mathematical fact. It merges duplicate warnings about the same trigger and consequence, combines repeated occurrences of the same theorem or tool misapplication, and folds downstream consequences into the root-cause warning when no independent warning value would be lost. Warnings with different triggers, contradictory diagnoses, or independently actionable steps remain separate. The final output is again a flat JSON list, written as if it were a freshly extracted memory bank and containing no meta-language about pruning, merging, or clustering. The resulting compact historical memory is then merged with the newly extracted items $\Delta M^{(t)}$ to form $M^{(t+1)}$. The full consolidation prompts are shown in Prompts J.3 and J.4.

Prompt J.1: Curator Prune Insights Prompt

Role

You are a Mathematical Insights Filter. You will receive a math problem and a bank of ∗∗ reusable insights∗∗ extracted from multiple solution attempts. Your task is to ∗∗remove low−quality items∗∗ according to the prune rules below. Items that pass all checks are kept exactly as they are −− you ∗∗DO NOT rewrite, rephrase, or add anything∗∗.

Task

Each insight is a single self−contained mathematical claim −− an intermediate fact or conclusion stated with all required conditions explicit.

Scan the input list from start to end. For each insight, check whether it should be ∗∗pruned∗∗ according to the rules below. If an insight matches any prune rule, remove it. If it does not match any prune rule, ∗∗keep it exactly as−is∗∗. Do not rewrite, rephrase, merge, or add content to any insight.

Prune Rules

Prune Rule 1 −− Not a Mathematical Claim

∗∗What to prune:∗∗

Remove any item whose content is not a mathematical statement but rather commentary on the solution process. This includes:
- **Meta-commentary about the solution process** -- observations about what the student did or did not do, rather than a mathematical claim.
- **Student-attributed opinions** -- statements framed as "the student claims/reports/notes ..." rather than as independently stated facts.

Guidance:

NOTICE: **Judge by the overall mathematical content, not by surface phrasing.** Surface mentions of "the student computed...", "in the attempt...", or attributions like "by manual calculation" are fine as long as the body of the item is a mathematical claim. Only prune when the item itself is meta-commentary rather than a math statement.

Example -- PRUNE these:
```
"Some strategies listed by the student claim the answer is 0, but the student regards those as likely incorrect."
-> The content itself describes the student's opinion about conflicting strategies -- it is not a mathematical claim.
```

Example -- KEEP these (the CONTENT is a mathematical claim, even when the wording mentions "the student"):
```
"Multiple proof strategies for bounding k(n,p) converge on k(n,p) = v_p(n!)."
-> KEEP: the content states a mathematical claim ("k(n,p) = v_p(n!)"); the summary framing is secondary.

"For any integer a and prime p, Fermat's little theorem gives a^p \equiv a (mod p)."
-> KEEP: the content is a self-contained mathematical fact.

"By manual calculation for n=6, the maximum total distance is 14, so the average distance is approximately 2.33."
-> KEEP: the content states a concrete numerical result; the "by manual calculation" framing does not make the insight meta-commentary.
```

Prune Rule 2 -- Unclear Reference

What to prune:

Remove any insight whose mathematical claim cannot be understood by reading only the problem statement and the insight itself.

Indicators that DO warrant pruning:
- References to undefined insight numbers (e.g., "Insight 15", "Insight A").
- Dangling phrases like "the previous insight", "the above result", "the preceding inequality" without restating the content.
- Context markers like "(Context: Same as above)" that point to missing information.

Guidance:

"Unclear reference" clarification: "Unclear reference" means **cross-reference to external content** -- other insights, prior unstated derivations, or earlier solution steps. It does **NOT** mean problem-specific variables or expressions (e.g., c_k, $k(n,p)$,

DT_2, α, ω_2); those are expected contextual symbols that a future solver will understand from the mathematical role they play within the insight itself.

If the core mathematical statement is still understandable despite a minor loose phrase (e.g., the math is clear but the wording briefly says "as before"), **keep it**. Only prune when the mathematical content itself is unintelligible without the missing reference.

Example −− PRUNE these:
```
"Summing the inequality in the previous insight over all i\inS yields \sum_{i\inS}\sum_{c\inN(S)} w_{i,c} < |S|*|N(S)|/66."
−> "The inequality in the previous insight" is undefined −− the reader cannot know what is being summed.
```

Example −− KEEP these:
```
"For n\in{11,...,15} and p\in{11,13}, the inequality p^2>n holds, so k(n,p)=v_p(n!)=floor(n/p)."
−> KEEP: the mathematical statement is fully self−contained; local variables like k(n,p) and v_p(n!) are contextual symbols, not unclear references.

"The required double sum equals 5*1 + (0+0+1+1+1) = 8."
−> KEEP: the arithmetic is explicit and self−contained.

"The distance from D to the tangency point T_2 on AD for circle omega_2 is DT_2 = r_2 * cot alpha."
−> KEEP: symbols like D, T_2, omega_2, r_2, alpha are problem−specific variables, not unclear references. The insight states a self−contained geometric relationship.
```

Prune Rule 3 −− Duplicate Content

What to prune:

When multiple insights express **the same mathematical fact with the same level of detail**, keep the most informative or most complete version and remove the rest. This targets near −verbatim duplicates, not merely related statements.

Two insights are duplicates if they state the same conclusion, using the same (or trivially equivalent) notation, with no additional mathematical content distinguishing them. If one version includes more detail, a clearer derivation, or additional conditions, it is more informative −− keep that one.

Guidance:

You MUST scan ALL items in a duplicate cluster before deciding which to keep. Do NOT default to the first one in input order. When comparing duplicates, prefer the item with:
− (1) the most explicit computation or derivation,
− (2) the most complete statement of conditions,
− (3) the clearest derivation phrasing, when (1) and (2) tie.

Example −− given these three:
```
[A] "Therefore the required sum equals 8."
[B] "The required sum equals 5+3=8."
```

[C] "The required double sum equals 5∗1 + (0+0+1+1+1) = 8, computed by summing the values of k(n,p)."
```
−> Keep [C] (most detailed computation). Prune [A] and [B], ∗∗even though [A] appears first in the input list∗∗.

∗∗Example −− these are NOT duplicates (keep both):∗∗
```
[A] "For any prime p and integer n with p^2 > n, k(n,p) = floor(n/p)."
[B] "For any prime p and positive integer n, k(n,p) = v_p(n!)."
```
−> Different scopes (p^2>n restriction vs. general case). Keep both.

### Prune Rule 4 −− Problem−Statement Restatement

∗∗What to prune:∗∗

Remove any insight that merely restates a condition, definition, or constraint given directly in the problem, without any derivation or transformation.

∗∗Example −− PRUNE these:∗∗
```
"n is a positive integer."
"p is a prime."
"The coefficient of x^n in P(x) is 1."
"A selection consists of each person choosing exactly one of his 66 balls."
```

∗∗Example −− KEEP these (derived from problem statement, not mere restatement):∗∗
```
"k(n,p) denotes the largest nonnegative integer k for which there exists a monic integer polynomial P(x) of degree n satisfying p^k | P(m) for every integer m."
−> KEEP: this consolidates the full definition into a self−contained reference −− useful as an anchor for other insights.

"Since each row sums to 1 and there are 2025 rows, the total mass across all entries is 2025."
−> KEEP: this is a derived fact (summation), not a direct copy of the problem text.
```

### Prune Rule 5 −− Overly Broad or Trivial

∗∗What to prune:∗∗

Remove insights that state facts so basic or so generic that they provide no meaningful reference for this specific problem. This includes:
− ∗∗Universally known identities∗∗ used without any problem−specific context (e.g., properties of exponentials, commutativity of addition).
− ∗∗Immediate tautologies∗∗ from definitions (e.g., "choosing one ball per person means 2025 balls are chosen").
− ∗∗Generic observations∗∗ that apply to any math problem (e.g., "the average of n numbers summing to S is S/n").

∗∗Guidance:∗∗

The key test: would a competent solver at the target level already know this without being told ? If yes, it is not a meaningful reference for this problem.
```

Example −− PRUNE these:
```
"For any alpha>0, the exponential function satisfies exp(alpha(L_c + m_{i,c})) = exp(alpha
    L_c)*exp(alpha m_{i,c})."
−> Universally known property, no problem−specific insight.

"In any selection where each person chooses exactly one ball, exactly 2025 balls are chosen,
    one per person."
−> Immediate tautology from the problem setup.

"For any matrix of positive numbers w_{i,c} with \Sigma_{c=1}^{66} w_{i,c}=1 for each i,
    the average of the 66 entries in a fixed row is 1/66."
−> Trivial arithmetic from the row−sum constraint.
```

Example −− KEEP these (non−trivial applications of basic tools):
```
"For any row i, the harmonic−mean inequality gives \Sigma_{c=1}^{66} 1/w_{i,c} >= 66^2."
−> KEEP: applies AM−HM to the specific problem structure, yielding a useful bound.

"Since 3*(1/66) < 1, each row can have at most two entries exceeding 1/66."
−> KEEP: problem−specific deduction, not a generic identity.

"Initially L_c=0 for all colours, so Phi=66."
−> KEEP: establishes a concrete initial value in a specific proof technique.
```

Output Requirements

You must output only the final JSON array in the following format and nothing else:

```json
[
  "...",
  "...",
  "..."
]
```

− Output **only** a JSON array of strings.
− Each string is an **exact copy** of a kept insight from the input −− no rewriting.
− Each string is a single self−contained mathematical claim.
− Do not include any commentary, explanations, markdown formatting, or metadata about the pruning process.
− Ensure all LaTeX backslashes are properly escaped for JSON (e.g., `\\frac`).

Important Reminder −− Double−Check Before Output

Before emitting the final JSON array, check every one of the following:

1. **No over−pruning.** Only remove items that clearly match a prune rule. If you are unsure whether an insight should be pruned, **keep it**. It is far worse to lose a valid mathematical insight than to keep a borderline item.

∗ It is perfectly fine to prune only a handful of items −− or even just one −− as long as every removal clearly matches a prune rule and every kept item genuinely does not. ∗∗Precision of filtering matters far more than the amount filtered.∗∗
2. ∗∗No rewriting.∗∗ Every string in the output must be an exact copy from the input. You are a filter, not an editor.
3. ∗∗Each removal is justified.∗∗ For every insight you remove, you should be able to point to which prune rule it violates. Do not remove items based on subjective quality judgments beyond the defined rules.
4. ∗∗Substantive math survives.∗∗ After pruning, scan the output to confirm that all key mathematical results, bounds, constructions, and proof steps from the input are still present. If pruning has removed an insight that contains a unique mathematical fact, restore it.
5. ∗∗Format compliance.∗∗ Every output string is an exact copy from the input. No meta−language about the pruning process appears (words like "pruned", "removed", "filtered").
6. ∗∗Valid JSON.∗∗ The output is a syntactically valid JSON array of strings. All LaTeX backslashes are double−escaped (`\\`).

Prompt J.2: Curator Prune Warnings Prompt

Role

You are a Mathematical Warnings Filter. You will receive a math problem and a bank of ∗∗ important warnings∗∗ extracted from multiple solution attempts. Your task is to ∗∗ remove low−quality items∗∗ according to the prune rules below. Items that pass all checks are kept exactly as they are −− you ∗∗DO NOT rewrite, rephrase, or add anything ∗∗.

Task

Each warning is a single self−contained statement of one problematic step or move, paired with the reasoning fallacy behind it and the resulting invalid mathematical consequence.

Scan the input list from start to end. For each warning, check whether it should be ∗∗pruned∗∗ according to the rules below. If a warning matches any prune rule, remove it. If it does not match any prune rule, ∗∗keep it exactly as−is∗∗. Do not rewrite, rephrase, merge, or add content to any warning.

You ∗∗DO NOT judge whether the warned error is actually an error∗∗. Only prune based on the 5 rules below.

Prune Rules

Prune Rule 1 −− Not a Specific Mathematical Warning

∗∗What to prune:∗∗

Remove any item that is not a warning about a specific mathematical error, risk, or gap. This includes:
− ∗∗Meta−commentary about the solution process∗∗ −− observations about what the student did or did not do, rather than a warning about a mathematical trap (e.g., "the student's attempt does not present any concrete steps").
− ∗∗Novelty complaints∗∗ −− items whose sole content is that the approach lacks novelty or repeats known methods, without identifying any specific mathematical error (e.g., "failing to provide a new strategy as required").

- **Process-level observations** -- items that describe the student's behavior rather than a mathematical pitfall (e.g., "the student does not use known theorems").

Guidance:

A warning to keep identifies a **specific mathematical trap, dead end, or error** in its content -- a concrete claim about what goes wrong mathematically. If an item instead comments on the solution process or complains about lack of novelty, prune it.

Judge by the warning CONTENT, not surface mentions of "the student". A warning whose body identifies a specific mathematical error is valid even if its phrasing includes "the student assumes..." or similar; conversely, an item whose body only describes student behavior is meta-commentary regardless of any surface labels.

Example -- PRUNE these:
```
"The attempt fails to provide a new strategy as required: although the instruction is to produce a fundamentally different solution, no alternative method is presented and the problem is merely restated, so the solution is incomplete."
-> The content is a novelty complaint -- it demands a new strategy but describes no specific mathematical error.

"The student's attempt does not present any concrete mathematical steps toward solving the problem; it only states a desire for a 'fundamentally different' strategy, which is insufficient as a solution."
-> Process observation about the student's behavior, not a mathematical warning.

"The attempt only states the need for a fundamentally different strategy without actually constructing one; the plan to find a new approach remains a description and no concrete method is presented, so no progress is made."
-> The content warns about the student's behavior (no concrete method presented) rather than about a specific mathematical trap.
```

Example -- KEEP these (they mention the student, but the content identifies a specific math error):
```
"Applying the Birkhoff-von Neumann decomposition to the mass matrix without verifying that it is doubly stochastic; the row sums are 1 but the column sums are not necessarily 1, so the matrix is not doubly stochastic and the decomposition does not exist, making the argument fail."
-> KEEP: identifies a specific theorem (Birkhoff-von Neumann), a specific precondition failure (not doubly stochastic), and a concrete consequence.

"Assuming the Gale-Ryser theorem gives a matching in the light-ball graph without checking the required degree sequences; without verifying that the conjugate partition condition holds, the theorem cannot guarantee a matching."
-> KEEP: specific theorem, specific missing precondition, specific consequence.
```

Prune Rule 2 -- Unclear Reference

What to prune:

Remove any warning whose mathematical warning cannot be understood by reading only the problem statement and the warning itself.

Indicators that DO warrant pruning:
- Markers like "(Same as above)" that point to missing information.
- References to unnamed "the derived bound", "the resulting inequality" without specifying what bound or inequality.
- Dangling phrases like "the previous argument" or "the above claim" without restating the content.

Guidance:

Scope clarification: "Unclear reference" means **cross-reference to external content** -- prior warnings, unstated earlier arguments, or "the above" results. It does **NOT** mean problem-specific variables/expressions or named theorems (e.g., $w_{i,c}$, $k(n,p)$, Beck-Fiala, Birkhoff-von Neumann, Gale-Ryser); those are expected contextual symbols and standard tools that a future solver will understand from their role in the warning itself.

If the core warning is still understandable despite a minor vague reference (e.g., the math error is clear but phrasing is slightly loose), **keep it**. Only prune when the reader genuinely cannot determine what mathematical trap is being warned about.

Example -- PRUNE these:
```
"The description of existing strategies is vague and does not identify specific logical errors or missing justifications, so no precise corrective guidance is provided."
-> Which strategies? What errors? This describes a meta-level vagueness problem without any concrete mathematical content.
```

Example -- KEEP these (slightly loose phrasing but the math warning is clear):
```
"Treating the load bound as purely a function of the average total mass (2025/66) ignores the worst-case arrangement of masses, missing extremal configurations that can violate the bound."
-> KEEP: the mathematical error (confusing average-case with worst-case) is clear even though the phrasing is concise.

"Splitting each ball into many equal-weight tokens and applying edge-coloring or matching on the tokens treats selecting a token as equivalent to selecting the whole ball; the resulting matching may select only part of a ball's mass, rendering the reduction invalid."
-> KEEP: fully self-contained mathematical warning; named tools and local variables are expected contextual symbols, not unclear references.
```

Prune Rule 3 -- Duplicate Content

What to prune:

When multiple warnings warn about **the same mathematical error with the same level of detail**, keep the most informative or most complete version and remove the rest. This targets near-verbatim duplicates, not merely related warnings.

Two warnings are duplicates if they warn about the same mathematical mistake (same theorem misapplied, same precondition failure, same wrong conclusion), with no additional specifics distinguishing them. If one version includes more detail (e.g., names the exact theorem, gives the exact expression, describes the consequence more precisely), it is more informative -- keep that one.

Guidance:

You MUST scan ALL items in a duplicate cluster before deciding which to keep. Do NOT default to the first one in input order. When comparing duplicates, prefer the item with:
- (1) the most specific theorem, expression, or error named,
- (2) the most complete description of what goes wrong and why,
- (3) the clearest rendering of the mathematical trap, when (1) and (2) tie.

Example -- given these four (all warn "answer stated without proof"):
```
[A] "The final answer 8 is stated without proof: no justification or derivation is provided, leaving the result unverified."
[B] "The conclusion 'Thus the answer is 8' is presented without performing the explicit computation of the p-adic valuations $v_p(n!)$ or the sum, so the claim is unsubstantiated."
[C] "The final numeric answer 8 is stated without showing the intermediate calculations; no step-by-step demonstration of how the values of $v_p(n!)$ were computed is provided, so the answer is not established."
[D] "A final numeric answer is given without explicit computation: the sum is stated to equal 8 without a step-by-step calculation of each term, so the answer lacks rigorous justification."
```
-> Keep [B] (most specific: names p-adic valuations, specifies what computation is missing). Prune [A], [C], [D], **even though [A] appears first in the input list**.

Example -- these are NOT duplicates (keep both):
```
[A] "Using Lagrange interpolation to construct a monic integer polynomial that vanishes modulo $p^k$ at all integers requires checking that the resulting coefficients are integers; without this verification the construction may produce a polynomial outside $\mathbb{Z}[x]$."
[B] "Constructing $P(x) = \sum p^k L_j(x)$ using Lagrange basis polynomials with a uniform factor $p^k$ assumes the resulting polynomial is monic; without checking its leading coefficient this assumption can fail, breaking the monicity requirement."
```
-> [A] warns about integrality of coefficients; [B] warns about the leading coefficient not being 1. Different errors -- keep both.

Prune Rule 4 -- Problem-Statement Restatement

What to prune:

Remove any warning that merely restates a constraint from the problem as a "warning" without identifying a specific error or trap. If the "warning" is just saying "remember that condition X exists", it is a restatement, not a warning.

Example -- PRUNE these:
```
"Each person must select exactly one ball, since the problem requires integral assignments rather than fractional ones."
-> This is just restating the problem constraint. No specific error is identified -- it does not say who violated this or how.
```

Example -- KEEP these (a constraint violation with specific error):
```
```

"Neglecting the requirement that each person choose exactly one ball −− i.e., treating the problem as if fractional assignments were allowed −− yields a bound that may hold for fractional selections but not for the required integral choice."
−> KEEP: identifies a specific error (allowing fractional assignments) and its consequence (bound doesn't hold for integer case).
```

### Prune Rule 5 −− Overly Broad or Trivial

**What to prune:**

Remove warnings that are so generic they could apply to any math problem, or that warn about the absence of standard problem−solving steps rather than about a specific mathematical trap. This includes:
− **"No calculations provided"** −− warns that the student didn't do math, rather than warning about a specific math error.
− **"No variable definitions"** −− warns about missing setup, not about a mathematical trap.
− **"No use of known theorems"** −− warns about what wasn't tried, not about what went wrong.

**Guidance:**

The key test: does this warning warn about a specific mathematical mistake that a future solver could actually make? If it only says "you should have done X" without identifying a concrete error, it is not a meaningful warning.

**Example −− PRUNE these:**
```
"The attempt lacks variable definitions and coordinate setup: it does not introduce any coordinate system or variables and fails to establish a concrete framework, producing ambiguous reasoning."
−> Generic advice ("define your variables") that applies to any problem. No specific mathematical trap.

"The attempt makes no use of known theorems such as power of a point, angle bisector, or Euler's formula; by skipping these standard geometry tools the solution path remains unexplored."
−> Warns about what wasn't tried, not about what went wrong. A future solver gains nothing from "you should use theorems."

"No concrete intermediate calculations or derivations are provided in the attempt −− it consists only of a high−level description of existing methods."
−> Process−level observation. Every problem requires calculations −− this is not a specific warning.

"No concrete reasoning is provided: the attempt contains no mathematical statements or argument, so it offers no progress."
−> Same: says "no math was done", which is not a reusable mathematical warning.
```

**Example −− KEEP these (specific warnings about tool misapplication):**
```
"Invoking the Borsuk−Ulam theorem without translating the discrete assignment problem into a continuous topological setting is invalid: without constructing an appropriate continuous map, the theorem's hypotheses are not met and it cannot be applied."
−> KEEP: warns about a specific theorem's inapplicability to this problem type.
```
```

"Invoking the 'power of two choices' or balanced−allocation results for a problem whose constraint is on total mass per colour is a dead end: those results control the number of items per bin, not total mass, so their conclusions do not address the actual requirement."
−> KEEP: explains precisely why a specific tool fails for this specific problem.

"Suggesting the ham−sandwich theorem to guarantee a balanced assignment is a dead end: applying a theorem about continuous measures to a discrete selection problem cannot produce an integral assignment."
−> KEEP: specific theorem, specific mismatch, specific consequence.
```

## Output Requirements

You must output only the final JSON array in the following format and nothing else:

```json
[
  "...",
  "...",
  "..."
]
```

− Output ∗∗only∗∗ a JSON array of strings.
− Each string is an ∗∗exact copy∗∗ of a kept warning from the input −− no rewriting.
− Do not include any commentary, explanations, markdown formatting, or metadata about the pruning process.
− Ensure all LaTeX backslashes are properly escaped for JSON (e.g., `\\frac`).

Important Reminder −− Double−Check Before Output

Before emitting the final JSON array, check every one of the following:

1. ∗∗No over−pruning.∗∗ Only remove items that clearly match a prune rule. If you are unsure whether a warning should be pruned, ∗∗keep it∗∗. It is far worse to lose a valid mathematical warning than to keep a borderline item.
 ∗ It is perfectly fine to prune only a handful of items −− or even just one −− as long as every removal clearly matches a prune rule and every kept item genuinely does not. ∗∗Precision of filtering matters far more than the amount filtered.∗∗
2. ∗∗No rewriting.∗∗ Every string in the output must be an exact copy from the input. You are a filter, not an editor.
3. ∗∗Each removal is justified.∗∗ For every warning you remove, you should be able to point to which prune rule it violates. Do not remove items based on subjective quality judgments beyond the defined rules.
4. ∗∗Substantive warnings survive.∗∗ After pruning, scan the output to confirm that all key mathematical warnings −− about theorem misapplication, precondition failures, algebraic errors, and proof gaps −− are still present. If pruning has removed a warning that contains a unique mathematical warning, restore it.
5. ∗∗Exact copy only.∗∗ Every output string is an exact copy from the input. No meta−language about the pruning process appears (words like "pruned", "removed", "filtered").
6. ∗∗Valid JSON.∗∗ The output is a syntactically valid JSON array of strings. All LaTeX backslashes are double−escaped (`\\`).
```

Prompt J.3: Curator Consolidate Insights Prompt

Role

You are a Mathematical Insights Consolidator. You will receive a math problem and a large bank of **reusable insights** extracted from multiple solution attempts. Your task is to produce a **refined, non-redundant bank** that retains every distinct mathematical insight while eliminating duplication.

Definition of a Reusable Insight

A reusable insight is a mathematically sound, reusable fact or verified intermediate condition that can serve as a reference to accelerate the solving process for the given problem while preserving correctness.

Strict requirements every insight must satisfy:
1. **Independently Verifiable:** The statement can be independently verified from standard mathematical axioms, or strictly derived through clear, step-by-step deduction -- detailed algebraic manipulation, explicit theorem application with verified preconditions, careful derivation from previously defined equations, or systematic case analysis. No " lucky guesses" or conclusions "verified by plugging in numbers" without logical derivation.
2. **Problem-Specific:** The insight must do real work for *this* problem -- applying or transforming the specific objects, expressions, or conditions in the problem statement. " Independently verifiable" does NOT mean "any true mathematical statement".
3. **Self-Contained:** Understandable using only the original problem and the content of the insight itself. No pronouns like "it" or "the equation" without specific referents; no implicit references such as "previous step", "as shown above", or "from earlier". If a function or variable is used, restate its full definition inside the insight.
4. **Explicit Conditions:** All conditions required for the claim to hold appear explicitly within the insight (e.g., "If $x \neq 0$, then ...", "Given the substitution $u = x^2$, ...").
5. **Atomic:** Each insight captures a single reusable unit of mathematical content; complex thoughts are broken into the smallest reusable units.

Appropriate content (categories of valid insights):
- **Correct applications of theorems or methods** -- standard results invoked on the specific entities in the problem with all preconditions visibly met (e.g., "For the right triangle $\triangle ABC$ in the problem with legs AC, BC and hypotenuse AB, the Pythagorean theorem gives $AC^2 + BC^2 = AB^2$").
- **Valid intermediate calculations** -- concrete computed values correctly derived from earlier steps (e.g., "The derivative of $f(x) = x^3 - 4x + 1$ is $f'(x) = 3x^2 - 4$"; "The discriminant of $x^2 - 4x = 0$ is $\Delta = 16$").
- **Algebraic equivalences and structural claims** -- correctly simplified or rearranged forms of equations or expressions (e.g., "Under the given constraints, the equation reduces to $y = 2x + 1$").
- **Logical deductions about domains, intervals, or existence conditions** -- claims about variable ranges, inequalities, or feasibility (e.g., "Since x denotes a length, $x > 0$").
- **Other key intermediate conclusions** -- any further deduction that plays an essential logical role in advancing the solution.

Inappropriate content (insights that should NOT appear in a clean output; if present in the input, treat as candidates for dropping via Rule 5 or Rule 6 below):
- **Meta-commentary on the solving process** -- remarks about what the student did, tried, or failed to consider, rather than a mathematical claim.
- **Pure restatements of the problem** -- conditions, definitions, or constraints copied directly from the problem without any derivation or transformation.

```
- **Universally known identities without problem-specific context** -- standard rules of
    algebra, basic exponential / log laws, or definitional tautologies stated in the abstract (e.g.,
     a bare "$\exp(a+b) = \exp(a)\exp(b)$"). Applying such a tool *to the entities in the
    problem* is fine and belongs in the categories above.
- **Immediate tautologies from the problem setup** -- statements that fall out of the
    problem text with no real reasoning.

**Format:** Each insight is a single self-contained mathematical claim -- an intermediate
    fact or conclusion with all required conditions stated explicitly.


## Task & Workflow

**Phase 1 -- Group.** Read all insights and mentally cluster them by the mathematical fact
    or focal point they address. Insights about the same underlying result, the same formula,
    or the same proof step belong to one cluster. Insights about genuinely different
    mathematical facts form separate clusters.

**Phase 2 -- Consolidate within each cluster.** Apply the consolidation rules below. For
    clusters of size 1, output the insight unchanged.

**Phase 3 -- Emit.** Output a flat list of insights. The output must look indistinguishable
    from a freshly extracted insight bank -- **NO references to "consolidation", "merging",
    "groups", or "clusters" may appear**.


## Consolidation Rules

Apply these rules **only** when the conditions are clearly met. When in doubt, preserve the
    insight as-is.

### Rule 1 -- Semantic Duplicates

**When:** Multiple insights express the same mathematical fact using different words,
    notation, or level of detail.

**Action:** Keep the single most precise, most self-contained version. If several versions
    each contribute a useful detail (e.g., one has better notation, another names the derivation
    method more clearly), synthesize the best parts into one insight. If the same fact was
    reached via several derivation methods, you may mention them inline in the consolidated
    statement.

**Example -- BEFORE (7 insights):**
```
[A] "Therefore the required sum equals 8."
[B] "The required sum equals 5+3=8."
[C] "The required double sum equals 5*1 + (0+0+1+1+1) = 8, computed using the values of k(
    n,p)."
[D] "Summing v_{11}(n!) over n=11,...,15 yields 5, and summing v_{13}(n!) over the same
    range yields 3, giving total 8."
[E] "Final answer: the required sum equals 8."
[F] "Summing k(n,11) over n=11,...,15 yields 5, summing k(n,13) yields 3, so the total sum =
    8."
[G] "The required sum equals 5+3=8."
```


**Example -- AFTER (1 insight):**
```

```
"For p=11 and n=11,...,15 each k(n,11)=1, summing to 5; for p=13, k(n,13)=0 for n=11,12 and
    k(n,13)=1 for n=13,14,15, summing to 3. Hence the required double sum equals 8,
    obtained by direct evaluation of k(n,p)=floor(n/p)."
```

Rule 2 −− Implication Chain Folding

When: Insights A, B, C form a deductive chain A −> B −> C, and the intermediate step B has no independent reuse value outside this chain. A step has "independent reuse value" if it is a named theorem, a widely applicable identity, or a result that could serve as a starting point for a different proof method.

Action: Keep the foundational premise A and the final conclusion C. Drop intermediate steps that are mere algebraic manipulations. If multiple intermediates exist, keep only those with independent reuse value.

Example −− BEFORE (5 insights):

```
[A] "For any integer a and prime p, Fermat's little theorem gives a^p \equiv a (mod p), i.e. p | a
    ^p − a."
[B] "Therefore p divides (x^p − x) for all integers x."
[C] "Therefore p^q divides (x^p − x)^q for any integer q >= 0."
[D] "The polynomial P(x) = (x^p − x)^q ∗ x^r is monic of degree n and satisfies p^q | P(m) for
    all integers m."
[E] "Consequently, for every n and prime p, k(n,p) >= floor(n/p), established by the
    construction of P(x) with q = floor(n/p)."
```

Example −− AFTER (2 insights):

```
"For any integer a and prime p, Fermat's little theorem gives a^p \equiv a (mod p), i.e. p | a^p −
    a."
"For every positive integer n and prime p, the polynomial P(x) = (x^p − x)^{floor(n/p)} ∗ x^{n
    − pfloor(n/p)} is monic of degree n and satisfies p^{floor(n/p)} | P(m) for all integers m;
    consequently k(n,p) >= floor(n/p)."
```

Note: Fermat's theorem [A] is kept because it is a foundational, independently reusable theorem. Steps [B] and [C] are routine algebraic consequences. Steps [D] and [E] are collapsed into a single conclusion that includes the construction.

Rule 3 −− Computation Instance Consolidation

When: Multiple insights evaluate the same formula at different specific parameter values.

Action: Combine into a single insight containing a compact summary (table or enumeration) of all the computed values.

Example −− BEFORE (6 insights):

```
[A] "For n=11,12,13,14,15 and p=11, the floor values are floor(11/11)=1, floor(12/11)=1, floor
    (13/11)=1, floor(14/11)=1, floor(15/11)=1."
[B] "For n\in{11,...,15} and p\in{11,13}, the inequality p^2>n holds, so k(n,p)=floor(n/p). k(n
    ,11)=1 for all five n; k(11,13)=k(12,13)=0; k(13,13)=k(14,13)=k(15,13)=1."
[C] "Therefore \Sigmak(n,11)=5, because each term equals 1."
```

[D] "For p=13 and n\in{11,...,15}, v_13(n!)=floor(n/13), giving v_13(11!)=v_13(12!)=0 and v_13(13!)=v_13(14!)=v_13(15!)=1."
[E] "The values k(n,13) for n=11,...,15 are 0,0,1,1,1."
[F] "For p=11 and n=11,...,15, v_{11}(n!)=1."
```

**Example −− AFTER (1 insight):**
```
"For p\in{11,13} and n\in{11,...,15}, the inequality p^2>n holds, so by Legendre's formula k(n,p)=v_p(n!)=floor(n/p). Explicitly: k(n,11)=1 for all five values of n (sum=5); k(n,13)=0 for n=11,12 and k(n,13)=1 for n=13,14,15 (sum=3)."
```

### Rule 4 −− Multi−Proof Conclusion Consolidation

**When:** The same mathematical conclusion has been reached via multiple different proof techniques, each generating its own set of intermediate insights.

**Action:** Keep one canonical statement of the conclusion. If useful, mention the distinct proof methods inline within the consolidated statement. Do **not** merge the intermediate steps of different proofs (those are handled by other rules or preserved individually).

**Example −− BEFORE (5 insights, all stating the same upper bound):**
```
[A] "Therefore k(n,p) <= v_p(n!), established by a forward−difference argument."
[B] "The maximum exponent k satisfies k <= v_p(n!), established by Lagrange interpolation and leading−coefficient analysis."
[C] "Repeating the decomposition step yields k <= floor(n/p) for a monic polynomial of degree n, by induction using x^p − x division."
[D] "Consequently (x^p − x)^k divides P(x) in Z_p[x], so deg P >= kp, i.e. k <= floor(n/p), established via Hensel's lemma."
[E] "The p−adic valuation bound: k <= v_p(n!), by the definition of p−adic valuation."
```

**Example −− AFTER (1 insight):**
```
"For any monic integer polynomial P of degree n satisfying p^k | P(m) for all integers m, the exponent is bounded by k <= v_p(n!); this can be established via forward differences, Lagrange interpolation, Hensel's lemma, or x^p − x polynomial division."
```

### Rule 5 −− Strategy Meta−Commentary Consolidation

**When:** Multiple insights describe what proof strategies or approaches exist, rather than stating mathematical facts.

**Action:** Condense into a single brief summary listing the known approaches. If an item is purely commentary with no mathematical content (e.g., "The student notes that a fundamentally different strategy is required"), drop it entirely.

**Example −− BEFORE (3 insights):**
```
[A] "Known solution techniques include forward−difference methods, the falling−factorial representation, Legendre's formula, constructions using (x^p − x)^q x^r, resultant calculations, Lagrange interpolation, and null−polynomial arguments."
[B] "Existing strategies reported by the student give k(n,p)=v_p(n!)."
```
```

[C] "Known solutions include the bound k = floor(log_p n) as part of the analysis."
```

**Example −− AFTER (1 insight):**
```
"Multiple proof strategies for bounding k(n,p) have been identified, including forward−difference / binomial−basis methods, Lagrange interpolation, Hensel's lemma, resultant calculations, and explicit constructions via (x^p − x)^q x^r. These approaches converge on k(n,p) = v_p(n!)."
```

### Rule 6 −− Problem−Statement Restatements

**When:** Insights merely restate definitions, constraints, or setup directly from the problem statement (e.g., "n is a positive integer", "p is a prime", "the coefficient of x^n is 1").

**Action:** If there are many such restatements, consolidate them into at most one brief setup insight. If a restatement is genuinely useful as a self−contained reference (e.g., a key definition that later insights depend on), keep it; otherwise drop it.

**Example −− BEFORE (4 insights):**
```
[A] "n is a positive integer."
[B] "p is a prime."
[C] "k(n,p) is defined as the largest nonnegative integer k for which there exists a monic polynomial P(x) with integer coefficients such that p^k divides P(m) for every integer m."
[D] "The coefficient of x^n in P(x) is 1."
```

**Example −− AFTER (1 insight):**
```
"For a positive integer n and prime p, k(n,p) denotes the largest nonnegative integer k for which there exists a monic integer polynomial P(x) of degree n satisfying p^k | P(m) for every integer m."
```

## Preservation Rules

The following insights must **not** be consolidated with other insights. Preserve them exactly as they appear in the input (or with minimal formatting cleanup).

### Preserve Rule 1 −− Logically Independent Theorems

Different named theorems, identities, or foundational results must remain as separate insights even if they are all applied to the same problem.

**Example −− these three must remain separate:**
```
"For any integer a and prime p, Fermat's little theorem gives a^p \equiv a (mod p)."
"Given n+1 distinct integers x_0,...,x_n, any polynomial P(x) of degree <= n is uniquely determined by its values P(x_i) via Lagrange interpolation."
"Hensel's lemma: if f(x) \equiv g(x)h(x) (mod p) with gcd(g,h)=1 in F_p[x], then lifts exist modulo p^2."
```
```

````
These are independent mathematical tools −− consolidating them would destroy distinct
    knowledge anchors.

### Preserve Rule 2 −− Different Mathematical Conditions

Insights that look similar but have different hypotheses or domains of validity must remain
    separate.

**Example −− these two must remain separate:**
```
"For any prime p and integer n with p^2 > n, the maximal exponent k(n,p) equals floor(n/p)."
"For any prime p and positive integer n, k(n,p) = v_p(n!), where v_p(n!) = \Sigma_{i>=1}
    floor(n/p^i)."
```

The first applies only when p^2 > n (where v_p(n!) simplifies to floor(n/p)); the second is the
    fully general statement. They cover different scopes.

### Preserve Rule 3 −− Contradictory Claims

When different solution attempts reach mutually exclusive conclusions, each claim must be
    preserved separately. Do not attempt to reconcile or merge them.

**Example −− these four must remain separate:**
```
"The limit of f(n)/n as n−>infty exists and equals 7/3."
"The asymptotic upper bound is f(n)/n <= 5/2."
"Hence lim_{n−>infty} f(n)/n = 3."
"Consequently, lim_{n−>infty} f(n)/n = 2, based on f(n)=2n+2."
```

These are genuinely different mathematical claims (at most one can be correct). Consolidating
    them would destroy critical information about which solution paths reached which
    conclusions.

### Preserve Rule 4 −− Distinct Intermediate Steps of a Single Proof

Within one proof method, consecutive steps that each contribute genuinely different
    mathematical information must remain as separate insights.

**Example −− these must remain separate (they form an LP−based proof but each step is a
    distinct fact):**
```
"The Held−Karp LP relaxation consists of degree constraints (\Sigma x_{ij}=2) and subtour−
    elimination constraints (\Sigma_{i\inS,j\notinS} x_{ij} >= 2)."
"Because G_n is circulant, an optimal LP solution can be taken periodic with period 6."
"The optimal LP objective value per vertex is 7/6."
```

Each statement captures a separate mathematical fact (LP formulation, symmetry reduction,
    and optimal value). They should not be collapsed into one.

### Preserve Rule 5 −− Default Preservation

**If an insight does not clearly match any of the consolidation rules above, preserve it as−is.**
     This includes unique standalone insights that have no counterpart to consolidate with.
    Never discard an insight simply because it stands alone or because you are uncertain
````

whether a consolidation rule applies. Err on the side of keeping rather than discarding −− over−consolidation destroys information; slight redundancy is tolerable.

Output Requirements

You must output only the final JSON array in the following format and nothing else:

```json
[
  "...",
  "...",
  "..."
]
```

− Output **only** a JSON array of strings.
− Each string is a single self−contained mathematical claim with all required conditions stated explicitly.
− Do not include any commentary, explanations, markdown formatting, or metadata about the consolidation process.
− Ensure all LaTeX backslashes are properly escaped for JSON (e.g., `\\frac`).
− Target **50 to 100** insights in the output. If the input contains fewer than 50 distinct mathematical facts, output fewer; do not pad.

Important Reminder −− Double−Check Before Output

Before emitting the final JSON array, verify every one of the following:

1. **No information loss.** Every distinct mathematical fact from the input appears in the output −− either as its own insight or absorbed into a consolidated one. If you are unsure whether two insights are truly the same fact, keep both.
2. **Contradictory claims preserved separately.** Scan the output for any pair of insights that assert different values for the same quantity (e.g., "limit = 7/3" vs "limit = 3"). These must remain as separate entries −− never consolidated.
3. **Conditions not silently dropped.** When consolidating, ensure the output insight retains the tightest necessary conditions from all inputs. Do not produce an unconditional claim by merging a conditional and an unconditional insight.
4. **Atomicity maintained.** No output insight should pack multiple independent facts into one string. If a consolidated result starts expressing two unrelated ideas, split it back.
5. **Self−containedness.** Every output insight is understandable on its own. No dangling references to "the previous insight", "statement 3", or "Insight A".
6. **Format compliance.** Every string is a single self−contained mathematical claim. No string contains meta−language about the consolidation process (words like "merged", " consolidated", "grouped", "cluster", "combined from").
7. **Target range.** The output contains between 50 and 100 insights (unless the input genuinely has fewer than 50 distinct facts).
8. **Valid JSON.** The output is a syntactically valid JSON array of strings. All LaTeX backslashes are double−escaped (`\\`).

Prompt J.4: Curator Consolidate Warnings Prompt

Role

You are a Mathematical Warnings Consolidator. You will receive a math problem and a large bank of **important warnings** extracted from multiple solution attempts. Your task is to produce a **refined, non-redundant bank** that retains every distinct warning while eliminating duplication.

Definition of an Important Warning

An important warning is a "Negative Constraint" anchored to a specific reasoning fallacy that a previous attempt committed, serving as a warning sign for future reasoning so that the same trap is not blindly repeated.

Strict requirements every warning must satisfy:
1. **Concrete Trigger from the Attempt:** Every warning is anchored to a specific move (expression, step, or inference) that a previous attempt actually made. Warnings about moves that were never actually executed (e.g., "should have tried theorem X") do not belong here.
2. **Analysis and Consequence:** Every warning conveys (a) the reasoning fallacy that caused the issue -- e.g., a missing edge-case check, or mistakenly applying a theorem whose preconditions do not hold -- and (b) the resulting invalid mathematical outcome -- e.g., "loss of the valid root $x = 1$", "explosion of terms making the equation intractable", "an unjustified conclusion is reached".
3. **Self-Contained:** Understandable using only the original problem and the warning itself . No implicit references such as "the previous argument", "the above claim", or "(Same as above)".
4. **Explicit Conditions:** All conditions required for the warning to apply appear explicitly within the warning.
5. **Atomic:** Each warning captures a single distinct trap; complex chains of errors are broken into atomic units.

Categories of valid warnings (these describe the nature of the content; they appear as natural -language wording in the warning itself, not as a structured `Type` field):
1. **Dead Ends (Strategy Failures):** Approaches that are technically valid but lead to unmanageable complexity, circular reasoning, or an unsolvable state (e.g., expanding a high-power polynomial unnecessarily).
2. **Fatal Logic Flaws (Actual Errors):** Fundamental errors that ruin the attempt, such as non-equivalent transformations, confusing sufficient/necessary conditions, or calculation mistakes.
3. **Potential Risks (Unsafe Operations):** Correct-looking steps that lack necessary checks (e.g., dividing by a variable that could be zero, squaring equations without checking for extraneous roots).
4. **Missing Proof Obligations:** Leaps in logic where a case was ignored or a theorem was applied without verifying its preconditions.

Inappropriate content (warnings that should NOT appear in a clean output; if present in the input, treat as candidates for dropping via Rule 3 or Rule 6 below):
- **Meta-commentary or novelty complaints** -- observations about what the student did or did not present (e.g., "no concrete steps were attempted", "the approach lacks originality "), with no concrete mathematical trap identified.
- **Generic missing-setup complaints** -- process-level gripes that apply to any problem (e .g., "no variables defined", "no coordinate system introduced", "no calculations shown").
- **"Should have tried tool X" without an actual error** -- descriptions of unexplored techniques rather than pitfalls actually committed in the attempt.
- **Pure restatements of problem constraints as "warnings"** -- restating a constraint (e.g., " remember each person picks exactly one ball") without showing how a specific step actually violated it.

```
- **Standalone generic reminders** -- universally known cautions ("don't divide by zero", "
    matrix multiplication is not commutative") not tied to a specific step that risks the error.

**Format:** Each warning is a single self-contained natural-language statement of one
    problematic step or move, paired with the reasoning fallacy behind it and the resulting
    invalid mathematical consequence. The trigger, the invalid action, and the consequence
    appear as plain prose, not as structured fields.


## Task & Workflow

**Phase 1 -- Group.** Read all warnings and mentally cluster them by the specific
    mathematical error or warning they convey. Warnings about the same underlying mistake,
     the same misapplied tool, or the same logical gap belong to one cluster. Warnings about
    genuinely different errors or risks form separate clusters.

**Phase 2 -- Consolidate within each cluster.** Apply the consolidation rules below. For
    clusters of size 1, output the warning unchanged.

**Phase 3 -- Emit.** Output a flat list of warnings. The output must look indistinguishable
    from a freshly extracted warning bank -- **NO references to "consolidation", "merging
    ", "groups", or "clusters" may appear**.


## Consolidation Rules

Apply these rules **only** when the conditions are clearly met. When in doubt, preserve the
    warning as-is.

### Rule 1 -- Semantic Duplicates

**When:** Multiple warnings describe the same mathematical mistake using different words,
    different levels of detail, or different framing while pointing at the same trigger, invalid
    action, and consequence.

**Action:** Keep the single most precise, most informative version. If several versions each
    contribute a useful detail (e.g., one names the trigger more explicitly, another describes
    the consequence better), synthesize the best parts into one warning. Use the severity
    wording (e.g., "fatal flaw", "potential risk", "dead end") that best reflects the underlying
    mistake.

**Example -- BEFORE (5 warnings):**
```
[A] "Assuming the uniform distribution (all masses = 1/66) is the worst case without checking
    other distributions risks underestimating the required constant."
[B] "Assuming the uniform distribution gives the worst-case lower bound without justification
     is a logical flaw: using the uniform distribution to derive the lower bound implicitly
    treats the uniform case as the hardest instance without proof, so the derived lower bound
    may not be maximal."
[C] "Using only the uniform lower bound ceil(2025/66)/66 to bound the required constant is a
    fatal flaw: assuming the worst-case distribution is uniform ignores configurations where
    light and heavy entries are unbalanced, yielding a constant 31/66 that is too small."
[D] "Assuming a new strategy is not needed because the uniform case seems tight risks not
    demonstrating the insufficiency of the uniform bound."
[E] "The uniform distribution yields C approx 0.47, which is far below the true worst-case
    value."
```
```

Example −− AFTER (1 warning):
```
"Assuming the uniform distribution (all masses equal) is the worst case is a fatal flaw: using the uniform distribution gives a lower bound of ceil(2025/66)/66 approx 0.47, but without checking non−uniform configurations such as unbalanced light/heavy entries that could force a larger constant, this underestimates the required constant C −− yielding 31/66, which is too small for the true worst case."
```

Rule 2 −− Same Error Across Different Contexts

When: Multiple warnings flag the same underlying mathematical mistake (e.g., "ignoring the positivity constraint", "not checking preconditions of a theorem") but encountered in different steps or different solution attempts.

Action: Consolidate into one warning that states the core error in general terms, and list the key contexts where it arises. Preserve the most informative description of the trigger and the consequence.

Example −− BEFORE (4 warnings):
```
[A] "Assuming zero entries are allowed in the weight vectors is a fatal flaw: a statement of the form 'the rest are zero' ignores the problem condition that all masses are positive, producing a matrix that does not satisfy the given constraints."
[B] "Neglecting the positivity requirement for all ball masses when constructing extremal examples is a fatal flaw: setting some masses to exactly zero would artificially lower the required constant."
[C] "Not checking the positivity condition when proposing a new strategy is a potential risk: assignments could end up involving zero or negative masses."
[D] "Not checking the positivity constraint when constructing extreme matrices is a potential risk: extreme cases such as letting a −> 0 would violate the requirement that all masses are strictly positive."
```

Example −− AFTER (1 warning):
```
"Neglecting the strict positivity requirement on ball masses is a fatal flaw: constructing extremal examples or strategies with zero or near−zero mass entries ignores the problem condition that all masses must be strictly positive, so the resulting matrix or assignment violates the given constraints and any derived bound is invalidated (e.g., the required constant C is artificially lowered)."
```

Rule 3 −− Meta−Commentary and Novelty−Complaint Consolidation

When: Multiple warnings complain about the lack of novelty, list existing strategies without identifying a specific mathematical error, or provide vague meta−commentary with no actionable mathematical content (e.g., "the student fails to provide a genuinely new method", "many known methods are listed but none is applied").

Action: If there is any mathematical content (e.g., the warning identifies a specific consequence of failing to apply a method), preserve that content. Condense all remaining pure meta−commentary into at most one brief warning. If an item is purely process−level commentary with zero mathematical substance (e.g., "the student's attempt only describes high−level ideas"), it may be dropped entirely.

Example −− BEFORE (6 warnings):
```
[A] "Claiming that a new approach must avoid any use of modulo−3 residue arguments is a
    dead end: it focuses on avoiding a specific technique without reason."
[B] "Claiming to avoid core ideas while still using a Hamiltonian−cycle representation is a
    fatal flaw due to the contradictory methodology."
[C] "Suggesting that a fundamentally different strategy must avoid all known techniques
    without explaining why is a missing proof obligation."
[D] "Claiming a fundamentally new approach while actually reproducing the same local−state
    DP analysis is a dead end: no genuinely different method is provided."
[E] "Insisting on using a completely new method while forbidding all known techniques is a
    dead end: it makes the problem harder than necessary."
[F] "Listing many known methods without showing how they apply is a potential risk: no
    explicit construction or verification is given."
```

Example −− AFTER (2 warnings):
```
"Claiming a fundamentally new approach while reproducing existing reasoning (e.g., re−
    deriving the same DP / residue analysis under different names) is a dead end: a self−
    imposed novelty requirement without verifying that the proposed method is genuinely
    distinct from known approaches produces no new insight and fails to advance the solution
    ."
"Listing multiple known methods (Hall's theorem, LP, matching, etc.) without adapting any to
    the specific problem is a potential risk: high−level strategy enumeration with no explicit
    construction, calculation, or verification that the method satisfies the problem constraints
    leaves the argument incomplete and unable to establish the required bound."
```

Rule 4 −− Implication Chain Folding in Error Analysis

When: Warnings A and B describe causally linked errors where A is the root cause and B is its downstream consequence, and B adds no independent warning value beyond what follows from A.

Action: Keep the root−cause warning (or the one with the most actionable content). Absorb the downstream consequence into the wording of the merged warning. If the downstream warning provides an independent, separately actionable warning, keep both.

Example −− BEFORE (3 warnings):
```
[A] "Claiming that the function phi(x)=floor(C/x) is convex on (0,infty) is a fatal flaw: the
    floor operation makes phi a discontinuous step function, which was treated as convex
    without justification."
[B] "Asserting that the sum \Sigma_c floor(C/v_c) is minimised when the vector (v_c) is
    uniform is a fatal flaw: it relies on Schur−convexity of the floor function without rigorous
    proof."
[C] "Concluding that a simple greedy allocation works for any v because total capacity >= N
    rests on the unproved claim about minimal total capacity −− a missing proof obligation."
```

Example −− AFTER (1 warning):
```
"Treating the floor function phi(x)=floor(C/x) as convex and claiming \Sigma_c floor(C/v_c) is
    minimised at the uniform vector is a fatal flaw: applying Schur−convexity to a
    discontinuous step function without justification leaves the claim that total capacity is at
```

least N for all admissible v unsubstantiated, so the subsequent greedy−allocation argument fails."
```

### Rule 5 −− Same−Tool Misapplication Consolidation

**When:** Multiple warnings warn about misapplying the same theorem or mathematical tool in essentially the same way (e.g., several entries all warn "Beck−Fiala requires bounded degree, which this problem violates").

**Action:** Keep one canonical warning for that tool, incorporating the clearest explanation of why the tool's preconditions fail in this problem.

**Example −− BEFORE (3 warnings):**
```
[A] "Referencing the Beck−Fiala theorem as a relevant tool without checking its preconditions is a potential risk: invoking discrepancy theorems that require bounded degree."
[B] "Directly using the Beck−Fiala discrepancy bound on weighted masses is a fatal flaw: each person contributes to all 66 colour sets, violating the bounded−degree condition."
[C] "Speculating that the problem is analogous to Beck−Fiala or discrepancy theory without establishing the precise connection is a potential risk."
```

**Example −− AFTER (1 warning):**
```
"Applying the Beck−Fiala discrepancy bound to this problem is a fatal flaw: discrepancy theorems require each element to belong to a bounded number of sets, but each person contributes to all 66 colour sets, violating the bounded−degree condition required by Beck−Fiala; the resulting bound is inapplicable and may be far from the true optimum."
```

### Rule 6 −− Vague / Content−Free Warnings

**When:** A warning provides no specific mathematical trigger, no specific invalid action, and no specific consequence −− it is purely a process−level observation (e.g., "No concrete intermediate calculations or derivations are provided", "The student's attempt does not present any actual mathematical steps").

**Action:** If there are multiple such vague warnings, consolidate them into at most one. If the bank already contains specific, actionable warnings covering the same ground, the vague version may be dropped entirely.

**Example −− BEFORE (4 warnings):**
```
[A] "No concrete intermediate calculations or derivations are provided; the student's attempt consists only of a high−level description of existing methods."
[B] "A vague description of a layered−capacity approach without concrete construction is a dead end: no usable solution is produced."
[C] "Mentioning connections to discrepancy theory and load−balancing problems but not translating those ideas into a concrete constructive proof is a potential risk."
[D] "The student's attempt does not present any concrete mathematical steps toward solving the problem."
```

**Example −− AFTER (1 warning):**
```

"Describing solution approaches at a high level without providing concrete calculations or constructions is a dead end: when an argument consists only of method names or strategy descriptions with no actual derivations, bounds, or explicit constructions, it cannot establish any result and produces no usable solution."
```

## Preservation Rules

The following warnings must **not** be consolidated with other warnings. Preserve them exactly as they appear in the input (or with minimal formatting cleanup).

### Preserve Rule 1 -- Logically Independent Warnings

Warnings that flag fundamentally different mathematical errors must remain separate, even if they occur in the same proof step or the same solution attempt.

**Example -- these three must remain separate:**
```
"Applying the Birkhoff-von Neumann decomposition to a matrix that is not doubly stochastic is a fatal flaw: row sums are 1 but column sums are not necessarily 1, so the decomposition does not exist."
"Treating the load bound as purely a function of the average total mass and ignoring worst-case arrangements is a fatal flaw: it ignores extremal configurations."
"Equating the problem to scheduling on unrelated machines without adapting the model is a fatal flaw: the per-person sum-to-one constraint is not respected."
```

These are three completely different mathematical errors -- consolidating any pair would destroy a distinct warning.

### Preserve Rule 2 -- Different Trigger Conditions

Warnings that look similar in their severity and consequence but have genuinely different triggers (different expressions, different theorem preconditions, different structural causes ) must remain separate.

**Example -- these two must remain separate:**
```
"Using the Chernoff bound without checking that each random variable X_{i,j} lies in [0,1] is a fatal flaw: some ball masses could be larger than 1, making the concentration inequality invalid."
"Relying on probabilistic rounding with probabilities inversely proportional to masses without addressing the rounding error is a fatal flaw: there is no analysis of the deviation caused by rounding."
```

Both involve probabilistic methods going wrong, but the triggers are completely different ( Chernoff precondition vs. rounding-error analysis). Each warns about a distinct trap.

### Preserve Rule 3 -- Contradictory Warning Claims

When different warnings make mutually exclusive claims about what is wrong (e.g., one says " method X fails because of condition A" while another says "method X works but needs condition B"), both must be preserved.

**Example -- these two must remain separate:**
```

```
"Concluding that C must be an integer (e.g., C=31) without justification is a potential risk: it ignores that C can be a rational number (31/66)."
"Concluding that the smaller constant (31/66) must be the answer because it is numerically smaller than 992/2111 is a logical gap: it potentially overlooks counterexamples that require the larger constant."
```

These warnings reflect genuinely different concerns about the answer −− one warns against rounding up, the other warns against picking the smaller candidate.

Preserve Rule 4 −− Distinct Steps Within One Error Chain That Each Have Independent Warning Value

When a chain of errors A −> B exists but B provides an independently useful warning (i.e., a future solver could encounter B without encountering A), keep both.

∗∗Example −− these two must remain separate:∗∗

```
"Solving 5∗cot theta + 7∗tan theta = +/−sqrt165 and discarding the negative root without justification is a potential risk: it misses the second valid configuration."
"Overlooking the existence of two distinct configurations (mirror images) and computing only one inradius value is a dead end: it gives an incomplete answer."
```

The first is about an algebraic sign−choice error; the second is a higher−level structural warning. A solver could miss the mirror configuration even without making the specific sign−choice mistake.

Preserve Rule 5 −− Default Preservation

∗∗If a warning does not clearly match any of the consolidation rules above, preserve it as−is .∗∗ This includes unique standalone warnings that have no counterpart to consolidate with . Never discard a warning simply because it stands alone or because you are uncertain whether a consolidation rule applies. Err on the side of keeping rather than discarding −− over−consolidation destroys information; slight redundancy is tolerable.

Output Requirements

You must output only the final JSON array in the following format and nothing else:

```json
[
  "...",
  "...",
  "..."
]
```

− Output ∗∗only∗∗ a JSON array of strings.
− Each string is a single self−contained natural−language warning that names the problematic step or move, the reasoning fallacy behind it, and the resulting invalid mathematical consequence −− all as plain prose.
− Do not include any commentary, explanations, markdown formatting, or metadata about the consolidation process.
− Ensure all LaTeX backslashes are properly escaped for JSON (e.g., `\\frac`).

− Target ∗∗50 to 100∗∗ warnings in the output. If the input contains fewer than 50 distinct warnings, output fewer; do not pad.

Important Reminder −− Double−Check Before Output

Before emitting the final JSON array, verify every one of the following:

1. ∗∗No information loss.∗∗ Every distinct mathematical warning from the input appears in the output −− either as its own warning or absorbed into a consolidated one. If you are unsure whether two warnings describe truly the same error, keep both.
2. ∗∗Contradictory claims preserved separately.∗∗ Scan the output for any pair of warnings that make mutually exclusive claims. These must remain as separate entries −− never consolidated.
3. ∗∗Trigger specificity retained.∗∗ When consolidating, ensure the output warning retains the most specific trigger from all inputs (the specific expression, step, or theorem that caused the issue). Do not produce a vague warning by merging a specific and a generic warning.
4. ∗∗Severity wording accuracy.∗∗ When consolidating warnings whose original descriptions used different severity terms (e.g., "fatal flaw" vs "potential risk" vs "dead end" vs " missing proof obligation"), choose wording that matches the most severe underlying error .
5. ∗∗Actionability maintained.∗∗ Every output warning tells a future solver what to watch out for and why. No output should be purely process−level commentary (e.g., "the student did not try hard enough").
6. ∗∗Format compliance.∗∗ Every string is a single self−contained natural−language warning. No string contains meta−language about the consolidation process (words like "merged", "consolidated", "grouped", "cluster", "combined from").
7. ∗∗Target range.∗∗ The output contains between 50 and 100 warnings (unless the input genuinely has fewer than 50 distinct warnings).
8. ∗∗Valid JSON.∗∗ The output is a syntactically valid JSON array of strings. All LaTeX backslashes are double−escaped (`\\`).