arXiv:2605.19338v1 [cs.MA] 19 May 2026

STAR-PólyaMath: Multi-Agent Reasoning under Persistent Meta-Strategic Supervision

Jiaao Wu[1*] **Xian Zhang**[2†] **Hanzhang Liu**[3]
Sophia Zhang[4] **Fan Yang**[2] **Yinpeng Dong**[1†]
[1]Tsinghua University [2]Microsoft Research [3]New York University [4]MIT
wuja25@mails.tsinghua.edu.cn dongyinpeng@mail.tsinghua.edu.cn
{zhxian, fanyang}@microsoft.com hl4963@nyu.edu zsophia@mit.edu

Abstract

Frontier AI models and multi-agent systems have led to significant improvements in mathematical reasoning. However, for problems requiring extended, long-horizon reasoning, existing systems continue to suffer from fundamental reliability issues: hallucination accumulation, memory fragmentation, and imbalanced reasoning-tool trade-offs. In this paper, we introduce **STAR-PólyaMath**, a multi-agent framework that systematically addresses these challenges through meta-level supervision and structured Reasoner-Verifier interaction. STAR-PólyaMath is structured as an orchestrated state machine with nested challenge-step-replan loops, governed by a reasoning-free Python orchestrator that separates control from inference and bounds error propagation through trace-back and re-planning. Our key innovation is a **persistent Meta-Strategist** that maintains cross-attempt memory and exercises meta-level control by issuing high-level strategic guidance or mandatory directives, so the system can escape unproductive loops rather than stagnate or over-rely on tools. STAR-PólyaMath achieves state-of-the-art results on all eight top-tier competition benchmarks: AIME 2025-2026, MathArena Apex Shortlist, MathArena Apex 2025, Putnam 2025, IMO 2025, HMMT February 2026, and USAMO 2026. It obtains perfect scores on AIMEs, Putnam, and HMMT, and shows its largest margin on Apex 2025, scoring 93.75% compared with 80.21% by the strongest baseline GPT-5.5. Ablation studies show that the gains arise from the framework's orchestration rather than from model-level diversity since removing key components or substituting in mixed backbones consistently weakens performance. Code is available at https://github.com/Julius-Woo/STAR-PolyaMath.

1 Introduction

Large Language Models (LLMs) now perform strongly on competition mathematics, with frontier systems reaching near-perfect scores on Olympiad-style benchmarks and delivering strong results on the Putnam and FrontierMath [1–4]. Progress has come from two complementary directions: natural-language reasoning strengthened by chain-of-thought prompting and reinforcement learning with verifiable rewards [5, 6], and tool-augmented reasoning via theorem provers or executable code [7–11]. Nevertheless, long-horizon problems remain challenging because free-form reasoning is difficult to verify, while tool use often incurs substantial computational and temporal costs.

A central limitation lies in the difficulty of reliable self-correction: LLMs often fail to revise flawed reasoning without external feedback, and explicit self-correction can even degrade performance [12].

*Work done during internship at Microsoft Research.
†Corresponding authors.

This limitation has motivated multi-agent frameworks that decompose reasoning into specialized roles, typically pairing a solution generator with a verifier that provides critique signals [13]. Such verification-centered workflows, including training-free verify-and-refine pipelines [14] and multi-agent debate [15], have demonstrated strong performance on competition-level mathematics. Related agentic systems further incorporate tool use [16], enabling code execution or formal verification during problem solving. Collectively, recent training-free and training-based agentic frameworks [17, 9, 18], together with frontier proprietary models [1, 2], suggest that multi-agent systems have become a leading paradigm for complex mathematical reasoning.

Despite these advances, existing systems continue to exhibit three recurring failure modes:

- **Hallucination accumulation.** Errors in intermediate reasoning propagate across long solution trajectories, as models tend to generate confident but incorrect arguments. Although debate [15] and self-verification [19] can mitigate this effect, reliably discriminating superficially plausible reasoning from ultimately invalid derivations remains difficult [20].
- **Memory fragmentation.** Difficult problems often require repeated backtracking across multiple unsuccessful lines of attack, yet most agentic systems either retain excessive context or fail to preserve salient information about prior attempts. Although certain formal systems address this through lemma caching and recursive decomposition [9, 17], these solutions are specialized for formal verification and do not generalize to open-ended mathematical reasoning.
- **Imbalanced reasoning-tool trade-offs.** Although code execution is reliable, excessive reliance on brute-force search can obscure mathematical structure and lead to intractable exploration. Prior work shows that models trained on tool-use trajectories develop a systematic bias toward code [21]. Existing remedies typically require additional training or ensemble overhead [18, 22]. In practice, agents often lack *metacognition*, the ability to step back and revise their strategy, leading to inefficient computation and failures on abstract problems.

Frontier agentic AI models have already possessed the knowledge and capacity to solve long-horizon mathematical competition problems. We argue that the missing ingredient is *persistent meta-level supervision*. In human problem-solving, a strong supervisor does not carry out every technical step, but guides at a meta level: selecting promising proof strategies, tracking failed approaches, and intervening when progress stalls. We instantiate this role as a **persistent Meta-Strategist** that retains global oversight throughout the entire problem-solving process. It maintains strategic memory across attempts and issues *meta-strategies*: decisions about the problem-solving strategy, formed as either high-level guidance grounded in domain knowledge or mandatory directives when the agents enter unproductive loops, such as repeatedly generating code instead of actively reasoning.

Based on this design, we introduce **STAR-PólyaMath**[3], an agentic framework with three LLM roles: a *Reasoner*, a *Verifier*, and a persistent *Meta-Strategist*, all coordinated by a reasoning-free Python orchestrator, as illustrated in Figure 1. The orchestrator controls execution flow: it advances a problem through an explicit state machine with nested challenge-step-replan loops, dispatching the Reasoner and Verifier through structured bidirectional debate loops, while the Meta-Strategist supervises the entire trajectory and intervenes through strategic guidance or mandatory directives. Empirically, this combination yields consistent gains across both final-answer and proof-based benchmarks: STAR-PólyaMath achieves the best reported scores across AIME 2025, MathArena Apex Shortlist, MathArena Apex 2025, Putnam 2025, IMO 2025, AIME 2026, HMMT February 2026, and USAMO 2026, with its largest margin on MathArena Apex 2025 (93.75 vs. 80.21 by the strongest baseline GPT-5.5). We further conduct systematic ablation studies to demonstrate that each component of the framework contributes meaningfully to the overall performance, and that swapping the standard shared backbone for mixed-model configurations does not improve results, indicating the gains from the framework's orchestration rather than from model selection.

2 Related Work

Mathematical Reasoning with LLMs. Early neural theorem proving focused on formal systems such as Lean and Isabelle [7]. Recent work has shifted toward informal reasoning: chain-of-thought

[3]The name pays homage to George Pólya's *How to Solve It* [23], which decomposes problem solving into *understanding the problem*, *devising a plan*, *carrying out the plan*, and *looking back*, and emphasizes heuristics such as working on analogous or auxiliary problems and revising the plan when it stops making progress. Our *exploration*, *planning*, *step-wise execution with verification*, and *re-plan* phases instantiate the first three; the persistent Meta-Strategist enacts the "look back" stage by examining the trajectory across attempts and switching to an auxiliary plan when the current one stalls.

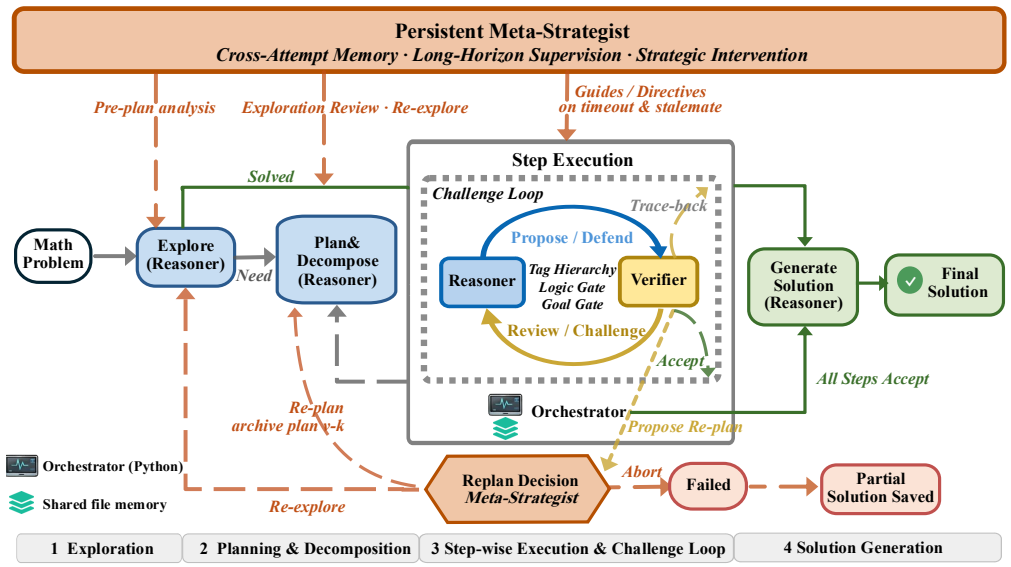

Figure 1: **STAR-PólyaMath system workflow.** STAR-PólyaMath advances each problem with four phases: *exploration*, *planning and decomposition*, *step-wise execution with challenge loops*, and *solution generation*. A Python orchestrator dispatches the LLM agents and decides advance, trace-back, re-plan, and abort transitions. The Reasoner probes the problem, proposes a step-wise plan, and executes each step with hierarchical verification tags (`[verified]` / `[easy-verify]` / `[hard-verify]`). The Verifier reviews via logic and goal gates, and emits ACCEPT, CHALLENGE, TRACE-BACK, or PROPOSE-REPLAN; persistent Reasoner-Verifier debate resolves disagreements within a step, while trace-back and re-planning bound error propagation across steps. A persistent Meta-Strategist supervises the trajectory with cross-attempt memory, issuing *pre-planning analysis*, *strategic guidance or mandatory directives* on timeout or stalemate, and *re-plan* verdicts.

prompting [5] and self-consistency [24] improve multi-step problem solving, while program-aided methods such as PAL [11] and ToRA [10] integrate code execution. Open-source models have also reached IMO gold-medal performance: DeepSeekMath-V2 [19] uses Group Relative Policy Optimization (GRPO) with meta-verification to internalize generation and verification in one model. Code-based Self-Verification (CSV) [25] grounds probabilistic LLM outputs via deterministic Python execution, bridging informal reasoning and formal verification. In contrast, our work introduces a multi-agent architecture with specialized roles and persistent memory.

Multi-Agent Systems for Reasoning. Debate-based approaches show that model argumentation improves accuracy [26, 15]. CAMEL [27] and AutoGen [16] explore role-playing agents for collaborative problem-solving. MACM [13] introduces condition mining with multi-agent coordination. Recent training-based approaches include MALT [28], which trains a Generator-Verifier-Refiner pipeline through multi-agent reinforcement learning, and ReMA [29], which uses hierarchical meta-reasoning with a high-level planner and low-level solver. However, these systems typically embed control inside one of the reasoning agents and reset context between sessions, leaving no component that supervises across attempts. STAR-PólyaMath's contribution is a structured separation of concerns: a reasoning-free Python orchestrator owns control flow, the Reasoner and Verifier interact through a structured debate protocol, and a persistent Meta-Strategist maintains cross-attempt memory and exercises meta-level supervision over the whole trajectory.

Self-Correction and Verification. Self-refinement [30] explores iterative self-improvement, but self-verification has fundamental limits: the model that erred often cannot detect its own errors [12]. Recent work on LLM evaluation [20, 31] reveals that models frequently fail to distinguish correct reasoning from plausible-sounding but flawed arguments, underscoring verification challenges. STAR-PólyaMath addresses this through structured Reasoner-Verifier debate with full session continuity and explicit verification tags that scale scrutiny to claim type, so that code-grounded results, easily checkable claims, and purely mathematical arguments are each evaluated appropriately.

Competition-Level Mathematics. AlphaGeometry [32] combines neural models with symbolic deduction for geometry olympiads, achieving silver-medal performance at IMO. AlphaProof [8]

uses reinforcement learning with Lean for formal theorem proving. FunSearch [33] employs LLM evolutionary search for mathematical discovery. Agentic reasoning frameworks [34] integrate tool use with multi-agent coordination. Analyses of code-reasoning trade-offs [21] reveal systematic biases toward brute-force approaches. STAR-PólyaMath differs by targeting all competition mathematics (not just geometry) and using debate-based verification with persistent metacognitive oversight.

3 Method

In this section, we present the design of STAR-PólyaMath. We first present the system architecture in Section 3.1, then provide detailed descriptions of the problem-solving workflow, including *exploration* (Section 3.2), *planning and decomposition* (Section 3.3), *step-wise execution with the challenge loop* (Section 3.4), and *solution generation* (Section 3.5).

3.1 System Overview

STAR-PólyaMath is implemented on top of a CLI agent platform that natively supports code execution, file editing, and session resumption. The system has three specialized agents coordinated by a Python orchestrator. This separation between control and reasoning enables robust long-horizon mathematical reasoning. Implementation details are given in Appendix B.

Orchestrator (Python). Owns all control flow and governs agent interactions. It maintains the per-problem state (current phase, plan, accepted results, prior failures), advances the problem through an explicit state machine, dispatches the LLM agents at the appropriate transitions, and parses their outputs into deterministic ADVANCE / TRACE-BACK / RE-PLAN / ABORT decisions.

Reasoner. The primary problem-solving agent. It explores the problem to gain insights, proposes a step-wise plan, executes each step (in natural language, symbolic manipulation, or executable code), defends its arguments in debate, and writes the final solution. It is not assumed to be reliable in isolation and interacts closely with the Verifier through a structured challenge loop.

Verifier. A strict reviewer evaluating the correctness and consistency of the Reasoner's outputs. For each step, it issues one of four verdicts: ACCEPT to proceed, CHALLENGE with specific objections, TRACE-BACK TO STEP M to re-examine earlier reasoning, or PROPOSE-REPLAN when no choice of step output could rescue the current plan. To mitigate verifier brittleness, the Reasoner is allowed to respond to challenges through a structured debate protocol with full session continuity before any control decision is committed.

Meta-Strategist. A human-like supervisor that operates at the meta level of the reasoning process. Unlike the Reasoner and Verifier, whose sessions can be reset or re-instantiated during execution, the Meta-Strategist maintains a single persistent session for the entire problem lifecycle, so that prior attempts, abandoned strategies, and chronic failure patterns remain in its working context. It issues *meta-strategies* at key decision points about which strategy the agents should pursue, retire, or revisit, either as light strategic guidance or, when agents enter unproductive loops, as mandatory directives. It is also the sole authority on re-planning verdicts: any proposal to abandon the current plan is routed to it for a specific decision.

3.2 Exploration

Before committing to a fresh new problem-solving plan, STAR-PólyaMath enters an *exploration* phase in which the Reasoner spends a bounded budget on cheap, non-committal investigation, such as enumerating small cases, looking for patterns, drafting candidate conjectures, and recording approaches that immediately fail. Each round produces a deliverable with a self-assigned assessment SOLVED, PARTIALLY-SOLVED, or NEED-PLAN which the orchestrator uses to decide what comes next.

If the Reasoner self-reports SOLVED, the orchestrator immediately routes the polished sketch through a whole-solution review by the Verifier; if accepted, the problem terminates here without ever entering the planning phase. In practice, this approach allows relatively simple problems to be solved quickly, improving overall system efficiency. PARTIALLY-SOLVED rounds are reviewed by the Meta-Strategist, who decides whether to invest in another exploration round or to proceed to plan with the accumulated evidence. NEED-PLAN and UNKNOWN rounds fall through directly to

planning. Across all branches, the structured findings of every round (what was tried, what worked, what failed, candidate approaches) are persisted as shared context that subsequent phases consume.

This phase is also re-entered after a re-plan. When the Meta-Strategist authorizes abandoning a plan, the orchestrator can run a fresh, evidence-driven exploration round before a new plan is generated, so that the next plan does not silently re-anchor on the same failed conjecture as the previous one.

3.3 Planning and Decomposition

Given the problem and any accumulated exploration findings, planning optionally begins with a *pre-planning analysis* in which the Meta-Strategist selects applicable methods from a predefined strategy library (e.g., invariant-based reasoning), flags common pitfalls, and warns of likely misinterpretations. These suggestions enter the Reasoner's context.

The Reasoner then formulates an explicit plan by decomposing the problem into a sequence of numbered steps (typically 6–10), where each step corresponds to a sub-goal. The orchestrator validates only the plan's structure, checking that the steps are properly numbered, parseable, and arranged in a coherent sequence, without evaluating the mathematical content. If the plan is structurally invalid, the Reasoner is asked to revise it; otherwise the system proceeds to step-wise execution.

This explicit decomposition mitigates *hallucination accumulation* by forcing the solution process to proceed through smaller, inspectable sub-goals rather than a single unconstrained reasoning trace.

3.4 Step Execution and Challenge Loop

STAR-PólyaMath executes the plan in a step-wise manner. For each step, the Reasoner produces a detailed report containing the proposed reasoning, computations, or constructions. Each report is annotated with a *verification tag* indicating the appropriate level of scrutiny:

- `[verified]`: results obtained from executed Python code and treated as directly grounded;
- `[easy-verify]`: claims that can be checked with code snippets or straightforward calculations;
- `[hard-verify]`: claims that require careful mathematical review.

The use of hierarchical verification tags supports a principled trade-off between computation and reasoning: claims suited for reliable execution are grounded in code, while purely mathematical arguments are subjected to stricter logical scrutiny.

The Verifier reviews the report through a *two-gate* evaluation. The *Goal Gate* compares the step's stated goal against what the report actually delivered, guarding against *semantic drift* where locally valid arguments miss the step (e.g., proving only one direction of a sharpness claim). The *Logic Gate* then checks correctness on the report's own terms, scaling scrutiny to each verification tag. Failure of either gate cannot be accepted. The Verifier emits one of four verdicts: ACCEPT, CHALLENGE, TRACE-BACK TO STEP M, or PROPOSE-REPLAN. The orchestrator dispatches based on the verdict; the four mechanisms below cover all non-ACCEPT cases.

Challenge Loop. When a step is challenged, the Reasoner and Verifier enter a structured debate with full session continuity. The Reasoner may defend, clarify, or revise its argument; the Verifier re-evaluates and re-issues a verdict each round. Both agents retain the in-step debate history so the same argument is not litigated twice. The loop terminates either when the Verifier accepts the revised report or when a fixed maximum number of rounds is reached without convergence; the latter is treated as a *stalemate* signal and surfaced to the Meta-Strategist.

Trace-Back. When the Verifier determines that an error originates from an earlier step M, the orchestrator archives steps $M, M{+}1, \ldots$ along with their debate logs, resets the Reasoner session so the next attempt does not carry abandoned context forward, and re-enters the step loop at step M. Verified intermediate results are auto-rescued and made available to the new attempt.

Re-Plan. A re-plan can be *proposed* by any of three sources: the Verifier PROPOSE-REPLAN, the Reasoner emits explicit `[plan-blocked]` tag in a response, or the orchestrator (a stalemate, chronic step-execution timeouts, or a Meta-Strategist intervention that already requested replanning). All proposals are routed to a single *Replan Decision* made by the Meta-Strategist, who returns one of four verdicts: CONTINUE, TRACE-BACK TO STEP M, APPROVE-REPLAN (archive the current plan, mark previously failed directions forbidden, and start a new plan attempt), or ABORT (terminate the problem). This unified verdict mechanism prevents the system from re-planning into directions it

has already tried and abandoned, and keeps the Meta-Strategist as the single point of authority on plan-level decisions.

Timeouts and pure-reasoning mode. When a long-running computation exceeds its time budget, the Meta-Strategist intervenes by extracting partial results and providing strategic guidance for the next attempt. A particularly forceful directive it can issue is to switch the Reasoner into a *pure-reasoning mode* that disables further code execution, forcing it to analyze the existing computational evidence by mathematical means. This mechanism directly addresses the *reasoning-tool trade-off* by enabling metacognitive intervention precisely when brute-force tool use becomes counterproductive.

Together, these mechanisms reduce *hallucination accumulation* by enforcing early error detection, preventing downstream error propagation through trace-back, and enabling the system to escape invalid lines of reasoning via re-planning. They also mitigate *memory fragmentation*: the challenge loop preserves debate context within each step, file-based progress tracking maintains continuity across session resets, and the Meta-Strategist's persistent session lets it recognize recurring failure patterns across attempts and respond with history-informed interventions rather than generic guidance.

3.5 Solution Generation

Once all steps have been accepted, the Reasoner consolidates the accepted step reports into a complete final solution. The Meta-Strategist can perform an optional *whole-solution review*, evaluating the assembled solution as a single document. This guard catches problems that step-level review can miss, such as cross-step inconsistencies. If the whole-solution review rejects the draft, the orchestrator re-invokes the Reasoner with the rejection visible up to a small retry budget, and otherwise saves the solution and terminates with success.

4 Experiments

We evaluate STAR-PólyaMath on competition-level mathematical reasoning benchmarks, including recently released 2026 contests. These benchmarks represent the pinnacle of mathematical competition, featuring the most recent and challenging problems that serve as rigorous tests of AI capabilities.

4.1 Setup

Benchmarks. Five benchmarks are *final-answer*: **AIME 2025** and **AIME 2026** (30 problems each from the American Invitational Mathematics Examination, integer answers in 000–999); **HMMT February 2026** (33 problems from the Harvard–MIT Mathematics Tournament); and **MathArena Apex 2025** (12 problems) together with **Apex Shortlist** (48 problems) [35, 36], which curate the most challenging recent contest problems. Three benchmarks are *proof-based*: **Putnam 2025** (12 problems from the William Lowell Putnam Mathematical Competition, demanding rigorous proofs in advanced undergraduate mathematics), **IMO 2025** (6 problems from the world's most prestigious high-school Olympiad), and **USAMO 2026** (6 problems from the U.S. Mathematical Olympiad).

Baselines. We compare against the strongest closed-source agentic models reported on these benchmarks, including GPT-5.5 (used at xhigh effort, `xh`), GPT-5.4 (`xh`), GPT-5.2 (high, `h`), Gemini 3.1 Pro, Gemini 3 Pro, Claude Opus 4.7 (`xh`), and Claude Opus 4.6 (`h`). We also include several leading open-source baselines: DeepSeek-v4-Pro (Max), DeepSeek-v3.2-Speciale, Kimi K2.6 (Think), Kimi K2.5 (Think), and GLM 5.1.

Model configuration. Unless otherwise stated, our standard **STAR-PólyaMath** configuration instantiates all three LLM roles with the GPT-5.5 at xhigh effort. Role separation is achieved through role-specific prompts, skills and per-step session isolation. Ablations on backbone diversity (Table 2) substitute alternative shared backbones or mixed-model configurations.

Inference budget. We impose per-call time limits of 1800 seconds for the Reasoner, 1200 seconds for the Verifier, 600 seconds for code execution, and 600 seconds for the Meta-Strategist. These limits apply to individual calls; wall-clock time per problem may exceed any single per-call budget. Full configuration, implementation details, and prompts are in Appendices A–C.

Table 1: Benchmark results (best reports in bold). Apex SL denotes Apex Shortlist; `xh` denotes xhigh reasoning effort, `h` denotes high; † denotes our own four-run averaged measurements.

Model	AIME 25	Apex SL	Apex 25	Putnam 25	IMO 25	AIME 26	HMMT Feb. 26	USAMO 26
GPT-5.5 (xh)	96.67†	93.75	80.21	89.58†	70.83†	97.50	97.73	98.21
GPT-5.4 (xh)	97.50†	78.12	54.17	83.33†	78.57†	99.17	97.73	95.24
GPT-5.2 (h)	**100.00**	78.12	13.54	76.04†	58.33†	98.33	96.97	50.00†
Gemini 3.1 Pro	93.33†	89.06	60.94	88.54†	67.26†	98.33	94.70	74.40
Gemini 3 Pro	95.00	67.19	23.44	75.83	41.67†	91.67	86.36	–
Claude Opus 4.7 (xh)	98.33†	63.02	40.62	77.08†	70.24†	95.83	93.94	63.10†
Claude Opus 4.6 (h)	96.67†	85.94	34.45	64.58†	30.36†	96.67	96.21	47.02
DeepSeek-v4-Pro (Max)	–	86.46	28.12	–	–	95.83	93.94	60.71
DeepSeek-v3.2-Speciale	95.83	68.23	9.38	59.17	–	–	–	–
Kimi K2.6 (Think)	89.17†	75.52	23.96	–	–	95.83	94.70	51.19
Kimi K2.5 (Think)	95.83	58.33	8.85	49.72†	–	95.83	87.12	–
GLM 5.1	–	72.40	11.46	–	–	95.83	89.39	–
Ours (all GPT-5.5xh)	**100.00**	**94.27**	**93.75**	**100.00**	**88.69**	**100.00**	**100.00**	**99.40**

Evaluation Protocol. Final-answer benchmarks are scored by answer correctness (0/1). For Proof-based benchmarks, IMO and USAMO are scored on a 0–7 scale following the public MathArena judging standards; Putnam use a 0/0.5/1 rubric: 1 for a correct proof, 0.5 for correct core reasoning with non-trivial gaps, and 0 for a fundamental error. Each proof is reviewed independently by at least two evaluators drawn from past competition medalists and mathematics Ph.D. students, with disagreements resolved through discussion. When available, baseline results are taken from published MathArena reports. For every experiment we run ourselves, including all STAR-PólyaMath variants and baseline entries not available, we run four independent attempts and report the average. Note that our evaluation protocol is aligned with MathArena. We mark such entries with † in Table 1.

4.2 Main Results

Table 1 reports the benchmark comparison across all evaluations. STAR-PólyaMath sets the strongest reported result on every benchmark column: perfect on AIME 2025, Putnam 2025, AIME 2026, and HMMT February 2026; 94.27 on Apex Shortlist; 93.75 on Apex 2025; 88.69 on IMO 2025; and 99.40 on USAMO 2026. The largest margin appears on Apex 2025, where our system reaches 93.75 compared with 80.21 by the strongest baseline GPT-5.5. Additional process statistics are presented in Appendix D.

4.3 Ablation Studies

Backbone substitution. Table 2 isolates the effect of backbone choice. Compared with the headline configuration that uses GPT-5.5 (`xh`) for all three roles, every alternative weakens performance. Two asymmetric mixed configurations (Reasoner+Verifier on one model, Meta-Strategist on the other) bracket the effect: pairing GPT-5.5 (`xh`) as Reasoner+Verifier with Claude Opus 4.7 (`xh`) as Meta-Strategist preserves more of the lift on Apex 2025 and IMO 2025 than the reverse pairing, but neither matches the homogeneous GPT-5.5 setting. Single-backbone weaker variants (`All GPT-5.2 (h)` and `All Claude Opus 4.7 (xh)`) drop further on the hardest benchmarks. Across all configurations, the framework retains the bulk of its lift over the corresponding bare-backbone baseline in Table 1, indicating that STAR-PólyaMath's gains come from structured orchestration rather than from any one specific backbone or from model-level mixing.

Component ablation. We next remove one mechanism at a time and evaluate the resulting system on a representative subset of benchmarks. To keep total cost manageable, all component ablations are run with the cheaper **All GPT-5.2** (`h`) configuration from Table 2 as the baseline; absolute scores are therefore lower than the headline numbers in Table 1, but the relative effect of removing each component is preserved and is what the ablation is designed to measure. Each ablated configuration is evaluated with four independent attempts and we report the average. For the baseline and the *w/o*

Table 2: Backbone substitution. All entries use the standard workflow; only the LLM backbone(s) instantiating the three roles change. "Mixed (A + B)" denotes Reasoner + Verifier instantiated by A and Meta-Strategist by B.

Configuration	AIME 25	Apex 25	Putnam 25	IMO 25	AIME 26	HMMT Feb. 26	USAMO 26
All GPT-5.5 (`xh`)	100.00	93.75	100.00	88.69	100.00	100.00	99.40
Mixed (GPT-5.5 + Opus 4.7)	100.00	58.33	100.00	82.14	100.00	100.00	98.81
Mixed (Opus 4.7 + GPT-5.5)	100.00	56.25	83.33	71.42	100.00	100.00	97.62
All GPT-5.2 (`h`)	100.00	52.08	91.67	75.00	100.00	97.73	75.00
All Claude Opus 4.7 (`xh`)	100.00	43.75	87.50	72.02	100.00	97.00	76.19

Table 3: Component ablations on top of the All GPT-5.2 (`h`) backbone configuration. All accuracy entries are four-run averages; numbers in parentheses are mean per-problem wall-clock in minutes for the rows where reported.

Configuration	AIME 2025	Apex 2025	Putnam 2025	IMO 2025
All GPT-5.2 (`h`)	100.00 (10 min)	52.08 (82 min)	91.67 (32 min)	75.00 (157 min)
w/o exploration	100.00 (18 min)	50.00 (99 min)	93.75 (41 min)	76.78 (121 min)
w/o Meta-Strategist	100.00	41.67	91.67	33.33
w/o persistent Meta-Strategist	100.00	41.67	83.33	25.00
w/o persistent debate	100.00	50.00	83.33	41.67
w/o arguing back	96.67	50.00	75.00	41.67
w/o trace-back & re-plan	91.67	16.67	37.50	17.86

exploration ablation we additionally report mean per-problem wall-clock time in parentheses, which is the dimension along which exploration's effect is most visible.

We describe each ablation configuration:

- **w/o exploration**: The exploration phase is disabled and the system proceeds directly to planning. We report both accuracy and the average per-problem wall-clock time.
- **w/o Meta-Strategist**: Completely removes the Meta-Strategist agent. When timeouts or stalemates occur, the system has no strategic intervention and must simply retry or terminate.
- **w/o persistent Meta-Strategist**: The Meta-Strategist exists but loses memory between interventions. Each timeout or failure is handled in a new session without learning from prior attempts.
- **w/o persistent debate**: The Reasoner and Verifier lose session continuity during challenge loops. Each debate round starts fresh.
- **w/o arguing back**: The Verifier may object to a step, but the Reasoner cannot engage in a sustained back-and-forth defense. This weakens bidirectional debate while preserving the rest of the pipeline.
- **w/o trace-back & re-plan**: Both backtracking mechanisms are disabled. The Verifier's TRACE-BACK and PROPOSE-REPLAN verdicts, together with the Reasoner's `[plan-blocked]` tag, are downgraded to CHALLENGE; the orchestrator can no longer archive earlier steps or abandon the current plan, so error recovery is limited to in-step debate.

Several patterns emerge from Table 3. **Trace-back and re-plan are the single most important mechanisms**: disabling both collapses accuracy on every proof benchmark, confirming that bounded cross-step error recovery is essential to long-horizon reasoning. **Exploration's effect is primarily on cost rather than accuracy** at this backbone level: accuracy stays within noise, while per-problem wall-clock rises on benchmarks where many problems short-circuit during exploration. **Cross-attempt memory matters more than within-attempt continuity on proofs**: removing Meta-Strategist persistence hurts IMO 2025 more than removing persistent debate. **Bidirectional debate matters on proofs** as well: *w/o arguing back* drops Putnam 2025 and IMO 2025 substantially. Overall, no single mechanism carries the gains in isolation; trace-back/re-plan and persistent meta-supervision contribute the largest share on long-horizon proof benchmarks.

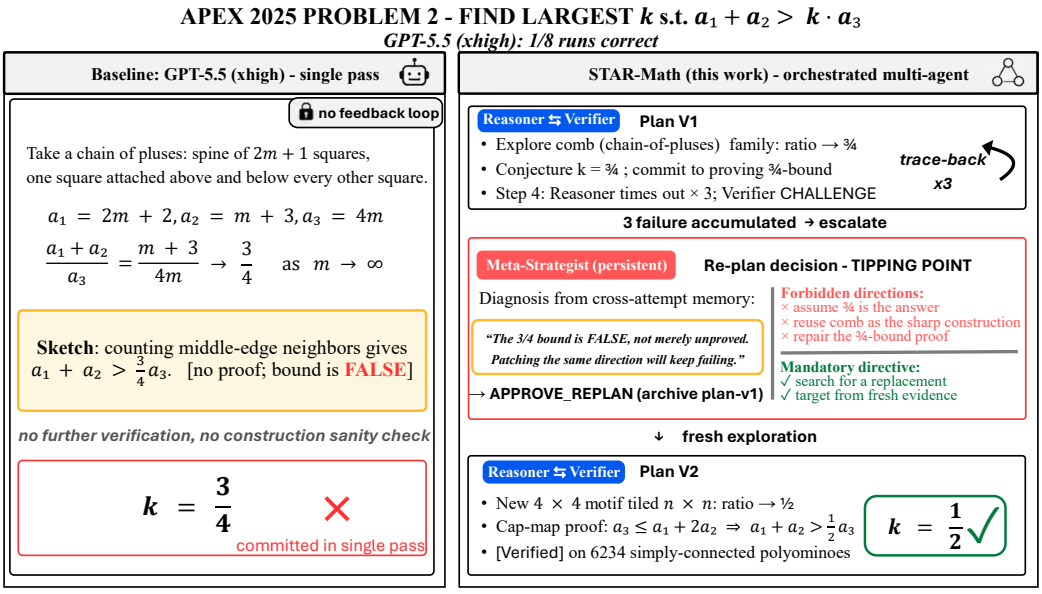

Figure 2: **Apex 2025 Problem 2 case study.** (Left) The example single-pass GPT-5.5 baseline commits to the chain-of-pluses construction and the false universal bound $k = 3/4$. (Right) STAR-PólyaMath's Plan v1 falls into the same attractor; after three timeouts and trace-backs, the Meta-Strategist's cross-attempt memory diagnoses "the $3/4$-bound is *false*, not unproved", issues an APPROVE-REPLAN verdict with explicit forbidden directions, and triggers a fresh exploration. Plan v2 then finds a denser 4×4 motif with ratio $\to 1/2$ and closes a code-verified cap-map proof, yielding the correct answer $k = 1/2$.

4.4 Case Study: Apex 2025 Problem 2 and the Role of Persistent Meta-Supervision

We briefly illustrate how persistent meta-supervision rescues a single-pass attractor failure on **Apex 2025 Problem 2**, on which the strongest baseline GPT-5.5 (`xh`) is correct on only 1/8 MathArena runs; the correct answer is $k = \frac{1}{2}$. The baseline locks onto a chain-of-pluses construction whose ratio tends to $3/4$ and confidently reports $k = 3/4$ on a universal bound that is in fact false. STAR-PólyaMath's Plan v1 falls into the same attractor; the Verifier challenges Step 4 and the Reasoner times out three times in a row, each triggering a local trace-back that fails to escape. Aggregating the three failures in its persistent session, the Meta-Strategist diagnoses that the bound is *false, not merely unproved*, returns APPROVE-REPLAN, and forbids future plans from anchoring on $3/4$ or reusing the comb. A fresh exploration then finds a denser 4×4 motif with ratio $\to 1/2$ and closes a code-grounded cap-map argument $a_3 \le a_1 + 2a_2$ (`[verified]` on 6,234 simply-connected polyominoes), yielding $k = 1/2$. It is the accumulation of repeated failures within a single Meta-Strategist session that converts "unproved" into "false-and-must-be-replaced", and turns the system away from a coherent-but-wrong story it would otherwise keep refining. This is precisely the failure mode that the persistent Meta-Strategist is designed to catch. Figure 2 summarizes the comparison; the full trajectory is given in Appendix E, with a second extended case study on IMO 2025 Problem 6 in Appendix F.

5 Limitations

We highlight three limitations that matter for interpreting the present results.

Compute cost and wall-clock time. STAR-PólyaMath dispatches many LLM calls per problem across the three roles, and per-call budgets bound individual invocations rather than the total trajectory; hard problems can substantially exceed human contest time and incur non-trivial token cost. This is acceptable when correctness dominates latency, but limits cost- or latency-sensitive deployment.

No formal verification layer. Verification is conducted in natural language, supplemented by Python execution for `[verified]` claims. A natural extension is a neuro-symbolic Verifier backed by a

formal system such as Lean, which would let `[hard-verify]` claims be discharged or surfaced as concrete unmet obligations rather than judged on prose alone.

Benchmark scope and saturation. STAR-PólyaMath and strong baselines saturate several of our benchmarks at or near 100%, so they are no longer informative for measuring further progress. Extending evaluation to research-level mathematics where problems are open-ended, may have no closed-form answer, and require sustained novel reasoning over days is the natural next frontier and one we leave to future work.

6 Conclusion

We introduced **STAR-PólyaMath**, a multi-agent framework that addresses three recurring failure modes of long-horizon mathematical reasoning: hallucination accumulation, memory fragmentation, and imbalanced reasoning–tool trade-offs through an orchestrated state machine with nested challenge–step–replan loops, supervised throughout by a persistent Meta-Strategist with cross-attempt memory. With a shared GPT-5.5 backbone, STAR-PólyaMath sets the strongest reported score on every benchmark in our comparison, with its largest margin on Apex 2025 (93.75 vs. 80.21). Backbone-substitution and component ablations show that the gains come from the framework's orchestration rather than from any specific model or from model-level mixing.

References

[1] OpenAI. Introducing GPT-5.2. https://openai.com/index/introducing-gpt-5-2/, 2025.

[2] Google DeepMind. Gemini 3 pro. https://deepmind.google/models/gemini/pro/, 2025.

[3] Jasper Dekoninck, Ivo Petrov, Kristian Minchev, Mislav Balunovic, Martin Vechev, Miroslav Marinov, Maria Drencheva, Lyuba Konova, Milen Shumanov, Kaloyan Tsvetkov, et al. The open proof corpus: A large-scale study of LLM-generated mathematical proofs. *arXiv preprint arXiv:2506.21621*, 2025.

[4] Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, et al. FrontierMath: A benchmark for evaluating advanced mathematical reasoning in AI. *arXiv preprint arXiv:2411.04872*, 2024.

[5] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24824–24837, 2022.

[6] DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[7] Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *International Conference on Automated Deduction (CADE)*, pages 625–635. Springer, 2021.

[8] Thomas Hubert, Remi Mehta, Laurent Sartran, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 2025. doi: 10.1038/s41586-025-09833-y.

[9] Jiangjie Chen, Wenxiang Chen, Jiacheng Du, Jinyi Hu, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Cheng Li, Zheng Yuan, et al. Seed-Prover 1.5: Mastering undergraduate-level theorem proving via learning from experience. *arXiv preprint arXiv:2512.17260*, 2025.

[10] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations (ICLR)*, 2024.

[11] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *International Conference on Machine Learning (ICML)*, pages 10764–10799, 2023.

[12] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024.

[13] Bin Lei, Yi Zhang, Shan Zuo, Ali Payani, and Caiwen Ding. MACM: Utilizing a multi-agent system for condition mining in solving complex mathematical problems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[14] Yichen Huang and Lin F. Yang. Winning gold at IMO 2025 with a model-agnostic verification-and-refinement pipeline. *arXiv preprint arXiv:2507.15855*, 2025.

[15] Akbir Khan, John Hughes, Dan Valentine, Laura Ruis, Kshitij Sachan, Ansh Radhakrishnan, Edward Grefenstette, Samuel R. Bowman, Trevor Darrell, and Ethan Perez. Debating with more persuasive LLMs leads to more truthful answers. In *International Conference on Machine Learning (ICML)*, 2024.

[16] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

[17] Sumanth Varambally, Thomas Voice, Yanchao Sun, Zhifeng Chen, Rose Yu, and Ke Ye. Hilbert: Recursively building formal proofs with informal reasoning. *arXiv preprint arXiv:2509.22819*, 2025.

[18] Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, et al. ToolOrchestra: Elevating intelligence via efficient model and tool orchestration. *arXiv preprint arXiv:2511.21689*, 2025.

[19] Zhihong Shao, Yuxiang Luo, Chengda Lu, Z. Z. Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. DeepSeekMath-V2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*, 2025.

[20] Hamed Mahdavi, Alireza Hashemi, Majid Daliri, Pegah Mohammadipour, Alireza Farhadi, Samira Malek, Yekta Yazdanifard, Amir Khasahmadi, and Vasant Honavar. Brains vs. bytes: Evaluating LLM proficiency in olympiad mathematics. *arXiv preprint arXiv:2504.01995*, 2025.

[21] Jiaheng Wang et al. To code or not to code? adaptive tool integration for math language models. *arXiv preprint*, 2025.

[22] Iddo Drori, Gaston Longhitano, Mao Mao, Seunghwan Hyun, Yuke Zhang, Sungjun Park, Zachary Meeks, Xin-Yu Zhang, and Ben Segev. Diverse inference and verification for advanced reasoning. *arXiv preprint arXiv:2502.09955*, 2025.

[23] George Pólya. *How to Solve It: A New Aspect of Mathematical Method*. Princeton University Press, 1945.

[24] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[25] Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification. In *International Conference on Learning Representations (ICLR)*, 2024.

[26] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *International Conference on Machine Learning (ICML)*, 2024.

[27] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

[28] Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip H. S. Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. MALT: Improving reasoning with multi-agent LLM training. In *International Conference on Machine Learning (ICML)*, 2025.

[29] Ziyu Wan, Yunxiang Li, Xiaoyu Wen, Yan Song, Hanjing Wang, Linyi Yang, Mark Schmidt, Jun Wang, Weinan Zhang, Shuyue Hu, and Ying Wen. ReMA: Learning to meta-think for LLMs with multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[30] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

[31] Ivo Petrov, Jasper Dekoninck, Lyuben Baltadzhiev, Maria Drencheva, Kristian Minchev, Mislav Balunović, Nikola Jovanović, and Martin Vechev. Proof or bluff? evaluating LLMs on 2025 USA math olympiad. *arXiv preprint arXiv:2503.21934*, 2025.

[32] Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625:476–482, 2024.

[33] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

[34] Jingyuan Wu et al. Agentic reasoning: A streamlined framework for enhancing LLM reasoning with agentic tools. In *Proceedings of ACL*, 2025.

[35] Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. MathArena: Evaluating LLMs on uncontaminated math competitions, 2025.

[36] Jasper Dekoninck, Nikola Jovanović, Ivo Petrov, and Martin Vechev. MathArena Apex: Unconquered final-answer problems, 2025. URL https://matharena.ai/apex/.

A System Configuration and Budgets

Runtime and Loop Bounds. Table 4 summarizes the hard limits governing the STAR-PólyaMath orchestrator. All timeouts apply to LLM inference and verification code execution.

Sampling Parameters. By default, all agents are dispatched through the GitHub Copilot CLI at `xhigh` reasoning effort, which activates extended inference for the Reasoner, Verifier, and Meta-Strategist. We do not modify temperature, top-p, or any other model-level decoding parameter from the platform default, and we do not tune any of these values per benchmark or per problem.

Agent Session Policy. Each agent role uses a deterministic, persistent session-naming scheme so that the orchestrator can decide between first-spawn (`-name`) and resume (`-resume`) without ambiguity (Table 5).

`<id>` is the problem identifier, `N` is the cumulative attempt index (incremented after every Reasoner trace-back or re-plan), and `NN` is the zero-padded step number. Persistent sessions are resumed via the Copilot CLI `-resume` flag; fresh sessions are created with `-name`. We did not tune any of the values in this section per benchmark; the same configuration is used for every problem reported in Tables 1, 2, and 3.

Table 4: System budgets and loop bounds. Identical values are used for every problem in every benchmark.

Parameter	Value
Per-call timeouts (seconds)	
Reasoner step reasoning	1800
Verifier step review	1200
Meta-Strategist intervention	600
Code execution (verification)	600
Loop iteration bounds	
`MAX_CHALLENGE_ROUNDS`	5
`MAX_REPLANS`	3
`MAX_EXPLORATION_ROUNDS`	2
`MAX_GENERATE_SOLUTION_RETRIES`	2
Plan structural bounds	
Step decomposition (recommended range)	6–10
Total steps allowed in a single plan	20
Stalemate detection	
Round budget for challenge stalemate	5

Table 5: Per-agent session persistence policy.

Agent	Session name	Scope
Reasoner	`reason-<id>-a{N}`	Cross-step within an attempt; `N` bumps on trace-back / re-plan
Verifier	`verify-<id>-step{NN}`	Fresh per step; persistent within the challenge loop
Meta-Strategist	`meta-<id>`	Persistent across the entire problem lifecycle

B Implementation Details

STAR-PólyaMath couples a procedural Python orchestrator with three specialized LLM agents (plus a non-coding Reasoner variant) coordinated through persistent state files and a hook-based context-injection mechanism. This appendix details the orchestrator state machine, the skill system, hook-based state injection, sub-agent dispatch, and the persistent state files that survive trace-backs and re-plans.

B.1 Orchestrator State Machine

The orchestrator is a deterministic, non-LLM control layer that advances each problem through five phases: *Setup*, *Exploration*, *Planning*, *Step Execution*, and *Solution Generation*. The pseudocode below shows the core loop; each phase transition is gated by an explicit verdict, and no progress occurs without acceptance.

Listing 1: Orchestrator five-phase loop.

```
Phase 0 Setup: load/create ProblemState, write problem.md, reset stats.

Phase 1 Exploration (if enabled):
  for round r = 1, ..., MAX_EXPLORATION_ROUNDS:
    spawn Reasoner exploration session -> exploration.md
    if assessment == SOLVED:
      Verifier whole-solution review (step_number = 0)
      if ACCEPT: finalize and terminate.
    elif assessment in {PARTIALLY_SOLVED, NEED_PLAN}:
      archive round; continue or fall through to Phase 2.

Phase 2 Planning:
```

```
  Reasoner emits plan.md (3-10 steps); orchestrator validates structure.

Phase 3 Step Execution Loop:
  for step N = current_step, ..., total_steps:
    Execute: Reasoner emits step-NN-report.md.
    Verify : Verifier (fresh session) ->
              ACCEPT | CHALLENGE | TRACE_BACK M | PROPOSE_REPLAN.
    if ACCEPT: advance to N+1.
    if CHALLENGE: enter Challenge Loop (<= MAX_CHALLENGE_ROUNDS rounds).
    if TRACE_BACK M: archive steps [M, N], restart at M.
    if PROPOSE_REPLAN: run Replan Decision sub-process.

Replan Decision (sub-process):
  Meta-Strategist returns CONTINUE | TRACE_BACK M | APPROVE_REPLAN | ABORT.
  on APPROVE_REPLAN: archive plan as plan-vK and return to Phase 2
                     if replan_count < MAX_REPLANS.

Phase 5 Solution Generation:
  Reasoner writes solution.md;
  Verifier whole-solution review;
  retry up to MAX_SOLUTION_RETRIES on rejection.
```

The orchestrator itself is stateless between calls: every control decision is recomputed from the persistent `ProblemState` object (Section B.5), which is serialized to `state.json` on every change.

B.2 Skill System

Skills are load-on-demand operational protocols stored in `.github/skills/`. Each skill is a self-contained Markdown file with a YAML preamble; agents invoke a skill by name when an applicable trigger is matched in their instructions, rather than carrying every protocol inline. This keeps each agent's static prompt short while making the protocols themselves auditable as standalone artifacts. Table 6 lists the seven skills shipped with STAR-PólyaMath, summarizing their purpose and invoking agent.

Table 6: STAR-PólyaMath skills. Invoking agents: R = Reasoner, V = Verifier, M = Meta-Strategist.

Skill	Purpose	Invoked by
`exploration-protocol`	Pre-plan exploration phase: deliverable structure, time budget, and the four assessment categories that drive routing.	R
`verifier-review-protocol`	Two-gate evaluation (Goal Gate + Logic Gate); ACCEPT / CHALLENGE / TRACE-BACK verdicts; required Verified Results block.	V
`verification-tag-protocol`	Definitions of `[verified]` / `[easy-verify]` / `[hard-verify]`; tag semantics and routing.	R, V
`meta-intervention-protocol`	Output formats and scenario-specific guidance for Meta-Strategist diagnoses, replan decisions, and timeout handling.	M
`math-solving-strategies`	Reference catalog of proof methods and tactics (induction, contradiction, extremal, invariants, ...) for strategy selection.	M
`construct-counterexamples`	Active falsification protocol: refute conjectures.	R, V
`code-issue-resolution`	Strategies for code timeouts, runtime errors, and wrong outputs.	R, M

B.3 Hook-Based State Injection

The orchestrator uses a Copilot CLI `sessionStart` hook to inject per-problem state into every agent session automatically. On every session start (including `-resume`), the hook reads the current `PROBLEM_STATE.md` for the active problem, extracts the slice relevant to the requesting agent role, and prepends it to the agent's incoming context. The hook is the single point at which shared state crosses into LLM context, so every other component (orchestrator, agents, skills) can treat `PROBLEM_STATE.md` as authoritative.

B.4 Sub-Agent Dispatch

Agents are spawned as Copilot CLI sub-processes by the orchestrator, with role descriptions, allowed tools, and hard constraints declared in per-role `.agent.md` files under `.github/agents/` (`reason`, `verify`, `meta-prompt`, `reason-noncoding`). Session persistence is controlled at dispatch time by the naming scheme already given in Table 5: the Verifier is spawned fresh per step and per whole-solution review, the Reasoner reuses one persistent session within a plan attempt (bumped on trace-back or re-plan), and the Meta-Strategist holds a single session for the entire problem lifecycle.

B.5 Persistent State Files

STAR-PólyaMath maintains a dual-layer per-problem state: a human-readable canonical layer (`PROBLEM_STATE.md`) and a machine-readable mirror (`state.json`). Both live under `scratch/<id>/`, alongside the plan, step reports, code, archives, and final solution.

Canonical layer: `PROBLEM_STATE.md`. This file is what agents see (via the injection hook of Section B.3). It contains the current phase, plan version and current step, the *Verified Results Ledger* (every claim accepted by the Verifier, tagged by category), the *Confirmed Failures* ledger (one record per trace-back / re-plan, with the abandoned plan summary, the explicit *forbidden directions*, and any rescued partial results), and exploration findings or hints if applicable.

Machine-readable mirror: `state.json`. The orchestrator serializes the `ProblemState` object (`src/state.py`) to JSON after every change. Schema fields include `current_phase`, `current_step`, `total_steps`, `steps` (per-step status, challenge rounds, trace-back count), `failed_records` (list of `FailureRecord`: reason, forbidden directions, rescued results), session names (`reasoner_session_name`, `meta_prompter_session_name`), and run statistics (per-agent call counts and wall-clock time).

Why a persistent ledger matters: trace-back and re-plan recovery. On every trace-back or re-plan, the orchestrator archives the abandoned plan version and step range to `archive/plan-vK/`. Crucially, the `failed_records` list is *never cleared*: it accumulates across replan attempts, and the next Reasoner spawn is injected with the full history of forbidden directions so it cannot silently re-anchor on a previously-tried strategy. Symmetrically, any verified intermediate results from earlier attempts (`rescued_results` in each `FailureRecord`) are re-injected into the new plan, so the system builds incrementally on partial progress instead of restarting from zero. This ledger is the single source by which trace-back and re-plan are non-destructive.

C Agent Prompts

Each agent is defined by three layers: (1) a system-level agent definition file in `.github/agents/`, (2) on-demand skills loaded from `.github/skills/` when an applicable trigger is matched, and (3) per-problem state injected at runtime by the `sessionStart` hook (Appendix B.3). This appendix shows the system-prompt cores and the verdict / decision schemas that the orchestrator parses; full prompts and skill files are provided in the supplementary code archive.

C.1 Reasoner

The Reasoner is the primary problem-solving agent: it explores, plans, executes steps, defends its arguments through the challenge loop, and writes the final solution.

System prompt (excerpt).

```
You are the **Reasoner** in a multi-agent mathematical reasoning system.

## CRITICAL: No Internet Access
You MUST NOT access the internet. Do not use ‘curl‘, ‘wget‘, ‘requests‘,
or any HTTP client. All reasoning must be self-contained.

## CRITICAL: Do Not Manage System Files
You MUST NOT write to ‘state.json‘, ‘progress.md‘, or ‘plan.md‘. These
```

```
files are managed by the orchestrator. You may read them for context.

## Your Responsibilities
1. Plan Creation: decompose the problem into 3-10 numbered steps.
2. Step Execution: execute one step at a time with rigorous reasoning.
3. Verification Tagging: tag every nontrivial claim with one of
   [verified] / [easy-verify] / [hard-verify].
4. Challenge Response: address the Verifier's concerns with evidence.
5. Code Execution: run code yourself to verify computational claims;
   do not propose code with [easy-verify] when you can run it.

## Verification Tags
Load the 'verification-tag-protocol' skill for full definitions.
- [verified] - You actually ran code; report the real output.
- [easy-verify] - Use only when you cannot run code yourself.
- [hard-verify] - Logical or proof claims not computationally verified.
[...]
```

Verification-tag protocol. Every nontrivial claim in a step report carries exactly one tag. The Verifier calibrates scrutiny per tag: `[verified]` claims are audited at the code-logic level (the Verifier does not re-run long computations but checks that the script and reported output are consistent); `[easy-verify]` claims are independently re-executed by the Verifier; `[hard-verify]` claims receive full proof review for hidden assumptions and logical gaps.

C.2 Verifier

The Verifier evaluates each step report through a two-gate framework: the Goal Gate (does the step achieve its stated objective?) and the Logic Gate (is the reasoning correct?).

System prompt (excerpt).

```
You are the **Verifier** in a multi-agent mathematical reasoning system.

Per-problem state (problem statement, plan, current step, prior verified
results, prior failures) is injected automatically as '[STAR-PolyaMath live
problem state]' -- treat it as authoritative.

## Hard rules
- No internet access. Verify locally only.
- Workspace boundary: read only inside the current problem's directory;
  do not read other 'scratch/<...>/' problems; do not write to
  'state.json', 'plan.md', or 'PROBLEM_STATE.md'.
- Save any verifier scripts to 'code/' as 'verify_step{NN}_*.py'.

## Method
For every review, follow the 'verifier-review-protocol' skill
(two-gate evaluation, output structure, verdict rules,
Verified Results block).
For verification tags, see 'verification-tag-protocol'.

## Disposition
- Be rigorous but fair.
- Be specific: quote exact text when identifying issues.
- A Goal-Gate failure is a CHALLENGE even if the Logic Gate passes.
[...]
```

Verdict output schema. The Verifier emits a structured report with the following sections:

- GOAL GATE — quotes the step objective from injected state, compares it to what the Reasoner actually delivered, declares YES / NO / PARTIAL.
- LOGIC GATE — reviews each tagged claim by tag type and notes any consistency issues with prior steps.
- VERDICT — exactly one of **ACCEPT**, **CHALLENGE**, **TRACE_BACK**, or **PROPOSE_REPLAN**.
- TRACE_BACK_TO — present iff verdict is TRACE_BACK; body is a single integer M with $1 \leq M \leq$ `current_step`.

- VERIFIED RESULTS — structured ledger entries tagged [Lemma], [Conjecture], [Computation], [Definition], or [Answer]; this is the snapshot the orchestrator carries forward through trace-back / re-plan.

C.3 Meta-Strategist

The Meta-Strategist is a persistent supervisor with cross-attempt memory. Unlike the Reasoner and Verifier, whose sessions can be reset, the Meta-Strategist retains a single continuous session for the entire problem lifecycle and intervenes at decision points with strategic guidance or mandatory directives.

System prompt (excerpt).

```
You are the **Meta-Prompter** in a multi-agent reasoning system. You
are a supervisor -- like a PhD adviser, not a co-solver. The Reasoner
does the math; the Verifier checks it. Your job is meta-level: detect
when joint effort goes wrong, and steer.

## Stance
- You may not know full mathematical detail. Avoid inventing lemmas
  the Reasoner has not produced.
- Do not become a third reasoner. If you find yourself drafting
  proofs or doing checks, stop.
- Read the situation, not the math. Look for: the Verifier raising
  the same concern repeatedly; the Reasoner cycling through variants
  of a broken idea.
- A long unconverged debate is itself a signal. Several rounds without
  convergence usually means the plan's conjecture is wrong, not that
  the Reasoner needs a nudge. Prefer APPROVE_REPLAN over mediation.
- When in doubt, escalate. Treat uncertainty as a reason to re-plan.

## Read-only
You MUST NOT solve the problem, write proofs, modify files, or
execute code.
[...]
```

Replan-decision output schema. When the orchestrator routes a re-plan proposal to the Meta-Strategist (from a PROPOSE-REPLAN verdict, a Reasoner [plan-blocked] tag, a stalemate at MAX_CHALLENGE_ROUNDS, or repeated timeouts), the response must contain:

- REPLAN_DECISION — exactly one of APPROVE_REPLAN, CONTINUE, TRACE_BACK_TO M, or ABORT.
- REASON_SUMMARY — single paragraph explaining the failure.
- PLAN_SUMMARY — (APPROVE_REPLAN only) one paragraph diagnosing what was wrong with the abandoned plan.
- FORBIDDEN_DIRECTIONS — (APPROVE_REPLAN only) 3–6 bullets the next plan must not take.
- REUSABLE_RESULTS — (optional) verified intermediate facts that survive the re-plan.

C.4 Orchestrator Output Parsing

The Python orchestrator extracts control signals from agent outputs by strict pattern matching on structured section headers, never on free-text prose. This is the mechanism by which control flow is isolated from any LLM hallucination in the reasoning text: even if a step report happens to contain words like "trace back" inside a sentence, only the contents of the dedicated ## Verdict and ## TRACE_BACK_TO sections influence the state machine.

Design principle. All control-flow parsing operates on structured headers (## Verdict, ## TRACE_BACK_TO, ### REPLAN_DECISION) rather than on free-text prose. A malformed verdict, including a trace-back without a parseable target step, is escalated by the orchestrator (e.g. as PROPOSE-REPLAN) rather than silently coerced, so every state transition is traceable to an explicit, parsable agent output.

D Runtime, Cost, and Verification Statistics

This appendix reports runtime, cost, control-flow, and verification-tag statistics that ground the cost discussion in Section 4.1 and the verification-tag protocol in Section 3.4. All numbers below come from the saved orchestrator state of the runs reported in Table 1. Every benchmark problem was attempted four times (four *batches*); per-problem statistics average across these four runs, and per-benchmark statistics average across the resulting per-problem values.

Cost reporting note. STAR-PólyaMath runs on the GitHub Copilot CLI, which bills by *premium requests* rather than by raw input/output tokens. We therefore report agent invocation counts as the unit GitHub Copilot meters against its premium-request quota, together with wall-clock time. As a first-order approximation, each Reasoner, Verifier, or Meta-Strategist invocation corresponds to one premium request; in-call code-execution sub-steps are excluded from the count.

D.1 Process statistics: exploration, trace-back, re-plan

Table 7 aggregates how often the orchestrator's three control-flow mechanisms are exercised on each benchmark. `N` is the number of (problem, batch) runs aggregated; "solved in exploration" counts runs that terminate during Phase 2 (Section 3.2) without entering planning; "mean trace-backs" and "mean re-plans" are averaged over runs.

Table 7: Control-flow statistics across benchmarks. "Solved in expl." is the share of runs that terminate via the Phase-2 SOLVED branch; "% w/ ≥ 1 TB" (resp. RP) is the fraction of runs that triggered at least one trace-back (resp. re-plan).

Benchmark	N	Solved in expl. (%)	Mean trace-backs	Mean re-plans	% w/ ≥ 1 TB	% w/ ≥ 1 RP
AIME 2025	120	100.0	0.00	0.01	0.0	0.83
AIME 2026	120	96.7	0.00	0.00	0.0	0.0
HMMT Feb 2026	132	93.9	0.05	0.05	2.3	2.3
Apex Shortlist	175	66.3	1.28	0.10	16.6	6.9
Apex 2025	44	54.5	1.39	0.30	36.4	18.2
Putnam 2025	48	72.9	1.02	0.10	18.8	10.4
IMO 2025	22	45.5	1.09	0.14	18.2	9.1
USAMO 2026	24	50.0	0.54	0.13	25.0	8.3

Three patterns emerge directly from the table.

(i) Exploration short-circuits easy benchmarks. On AIME 2025 every single run terminates in Phase 2 (100% solved in exploration); on AIME 2026 and HMMT 2026 the corresponding shares are 96.7% and 93.9%. These benchmarks rarely reach planning at all — the Reasoner produces a candidate solution during exploration and the Verifier's whole-solution review accepts it. By contrast, only 45–73% of runs on Apex 2025, Apex Shortlist, IMO 2025, USAMO 2026, and Putnam 2025 short-circuit this way; the remainder proceed through the full step loop. This is exactly the design intent of Section 3.2: cheap problems are dispatched cheaply, while harder problems incur the full cost of structured planning and verification.

(ii) Trace-back is a routine recovery mechanism on hard benchmarks; re-plan is reserved for plan-level errors. On Apex 2025 the mean number of trace-backs per run is 1.39, with 36.4% of runs triggering at least one. The mean number of re-plans is an order of magnitude smaller (0.30, with 18.2% triggering at least one). The same separation holds on Apex Shortlist (1.28 vs. 0.10), IMO 2025 (1.09 vs. 0.14), and Putnam 2025 (1.02 vs. 0.10): trace-backs are 5–$13\times$ more frequent than re-plans. This matches the design philosophy of Section 3.4: trace-back is the local error-recovery mechanism, while re-plan is a heavier intervention reserved for situations where the Meta-Strategist judges the entire plan unsound (cf. Section 4.4, Appendices E and F).

(iii) AIME-class benchmarks essentially never trace-back or re-plan. On AIME 2025/2026 the mean trace-back count is exactly 0, and the mean re-plan count is at most 0.01. Combined with the $\geq 96.7\%$ exploration-solve rate, this confirms that the orchestration overhead is paid almost entirely on hard benchmarks — the reverse of a system that uniformly spends compute regardless of difficulty.

D.2 Per-benchmark cost summary

Table 8 aggregates wall-clock time and agent-invocation counts across the eight benchmarks. *Mean* and *Median* are over per-problem averages; *Max* is the single longest-running problem; "Max problem" names that problem.

Table 8: Per-benchmark wall-clock and agent-invocation statistics. Wall-clock is in minutes; agent-call columns are mean per-problem invocations of the Reasoner, Verifier, and Meta-Strategist respectively.

Benchmark	Mean	Median	Max	Max problem	Reas.	Veri.	Meta
AIME 2025	8.20	7.88	39.57	`aimeII13`	6.83	4.71	0.02
AIME 2026	8.30	7.59	33.15	`aimeII13`	6.71	4.63	0.03
HMMT Feb 2026	7.04	2.61	59.95	`hmmt20`	3.91	2.47	0.13
Apex Shortlist	38.88	19.58	529.00	`apex_sl_7`	9.27	6.49	0.86
Apex 2025	86.69	35.75	498.00	`apex2025_12`	13.00	8.59	2.16
Putnam 2025	15.86	3.49	136.00	`putnamA6`	6.92	4.79	0.63
IMO 2025	54.53	18.63	498.00	`imo2025_6`	10.68	7.32	1.64
USAMO 2026	16.97	16.93	37.98	`usamo6`	8.83	6.25	0.46

Three observations align Table 8 with Table 7.

(i) Effort scales with difficulty. Easy benchmarks have median wall-clock $\leq$ 8 minutes and effectively zero Meta-Strategist invocations per problem (≤ 0.13). Apex 2025 and IMO 2025 invoke the Meta-Strategist 1.6–2.2 times per problem on average, consistent with the high trace-back / re-plan rates in Table 7.

(ii) Median is much smaller than mean on hard benchmarks. On Apex 2025, the mean is 86.7 minutes but the median is only 35.8; on IMO 2025, 54.5 vs. 18.6. A small number of tail problems consume disproportionate time — in particular, `apex2025_12` (which is the same problem as `imo2025_6`; see Appendix F) is the longest-running problem in the suite at 498 minutes (≈ 8.3 h).

(iii) Verifier-to-Reasoner call ratio is roughly stable. Across all benchmarks the Verifier is invoked at ≈ 0.6–$0.7\times$ the Reasoner's call count, reflecting the design in Section 3.4: each step incurs one Reasoner call followed by Verifier evaluation, plus additional Reasoner calls inside any challenge loop. The Meta-Strategist call count is the most diagnostic of difficulty, varying by two orders of magnitude across benchmarks (0.02 on AIME 2025 to 2.16 on Apex 2025).

D.3 Per-problem wall-clock distribution

Figure 3 shows the per-problem wall-clock distribution for each benchmark. Each marker is one problem, averaged over its four runs.

The figure makes the head–tail structure visible without relying on standard-deviation summaries (which are unstable at the four-run-per-problem sample size we use).

D.4 Verification-tag statistics

Table 9 reports the share of `[verified]`, `[easy-verify]`, and `[hard-verify]` tags across all step reports in the runs of Table 1, and Figure 4 visualizes the corresponding stacked breakdown.

The pattern is consistent with Section 3.4's claim that the verification-tag protocol implements a principled trade-off between reasoning and tools rather than a uniform bias toward either. `[verified]` claims dominate on AIME 2025/2026 (36–43%) and HMMT 2026 (36.5%), where final-answer arithmetic and small-case enumeration can be discharged by code; on the proof-based benchmarks (Putnam 2025, IMO 2025, USAMO 2026) the share of `[hard-verify]` claims rises to 83–86%, which is precisely where Section 3.4's Logic Gate carries the bulk of verification work. `[easy-verify]` is rare ($\leq 0.07\%$) across the board: when the Reasoner can cheaply check a claim, it almost always runs the check itself and upgrades the tag to `[verified]`, the behaviour explicitly mandated by the Reasoner system prompt (Appendix C.1). This turns the verification-tag protocol from a design choice into a measured property: the share of code-grounded claims tracks problem type, and the Verifier's calibration is exercised on a measurable mix of evidence.

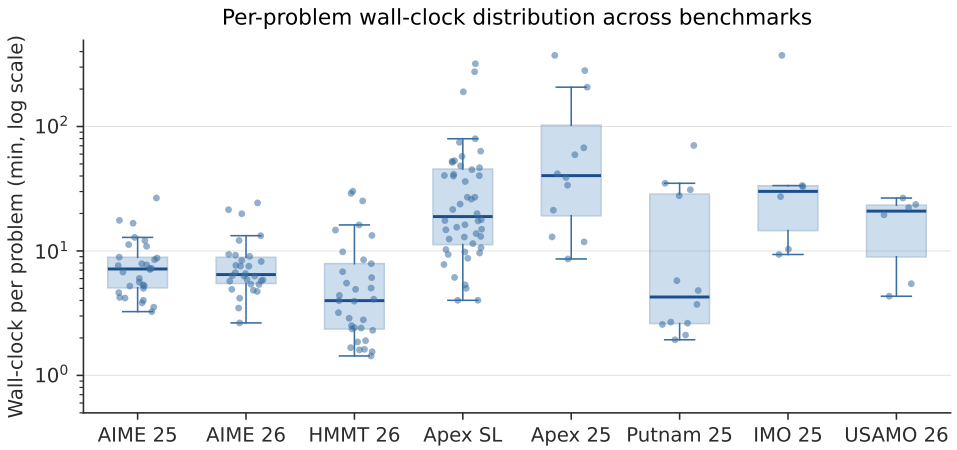

Figure 3: Per-problem wall-clock distribution across the eight benchmarks, on a log scale. Each marker is a single problem averaged over its four runs; box-and-whisker overlays show the median and inter-quartile range. AIME, HMMT, and USAMO problems concentrate at ≤ 30 min; Apex Shortlist, Apex 2025, and IMO 2025 have heavy upper tails dominated by a few hard problems, including the shared `apex2025_12`/`imo2025_6` tiling problem at the top.

Table 9: Verification-tag statistics aggregated over all step reports per benchmark. Counts are over individual tagged claims; percentages are within-benchmark.

Benchmark	**# step reports**	**# claims**	**% [verified]**	**% [easy-verify]**	**% [hard-verify]**
AIME 2025	437	4,572	43.0	0.07	56.9
AIME 2026	437	4,801	36.0	0.02	64.0
HMMT Feb 2026	160	1,988	36.5	0.05	63.4
Apex Shortlist	682	10,727	21.5	0.00	78.5
Apex 2025	220	3,329	24.2	0.00	75.8
Putnam 2025	117	1,930	14.1	0.00	85.9
IMO 2025	106	2,088	14.8	0.00	85.2
USAMO 2026	108	1,860	16.7	0.00	83.3

E Extended Case Study I: Apex 2025 Problem 2

This appendix complements Section 4.4 by laying out STAR-PólyaMath's full trajectory for Apex 2025 Problem 2. We trace the system's evolution across three phases: Plan v1 (a failed attempt anchored on an incorrect $3/4$ bound), the persistent Meta-Strategist's diagnoses across three trace-backs and the final re-plan verdict, and Plan v2 (the successful re-execution targeting the correct $1/2$ constant).

E.1 Plan v1: the failed $3/4$ attempt

Plan v1 conjectured $k = 3/4$ and committed to proving the universal bound $a_1 + a_2 > \frac{3}{4}\, a_3$. The evidence came from the *comb family* of polyominoes (a horizontal spine of $2m+1$ unit squares with one square attached above and below every other interior spine square). The verified comb counts, recorded in `PROBLEM_STATE.md` under "Rescued from earlier attempts (Plan v1, Step 3)", are

$$a_1(C_m) = 2m+2,\ a_2(C_m) = m+3,\ a_3(C_m) = 4m,\ \frac{a_1(C_m)+a_2(C_m)}{a_3(C_m)} = \frac{3}{4} + \frac{5}{4m} \xrightarrow{m\to\infty} \frac{3}{4}.$$

The Reasoner correctly established these counts (verified by a Step 3 enumeration script) and confirmed by exhaustive search that no hole-free fixed polyomino through 10 cells violates $4(a_1 + a_2) \leq 3a_3$. The candidate constant $3/4$ was therefore strongly supported by the evidence available at that point.

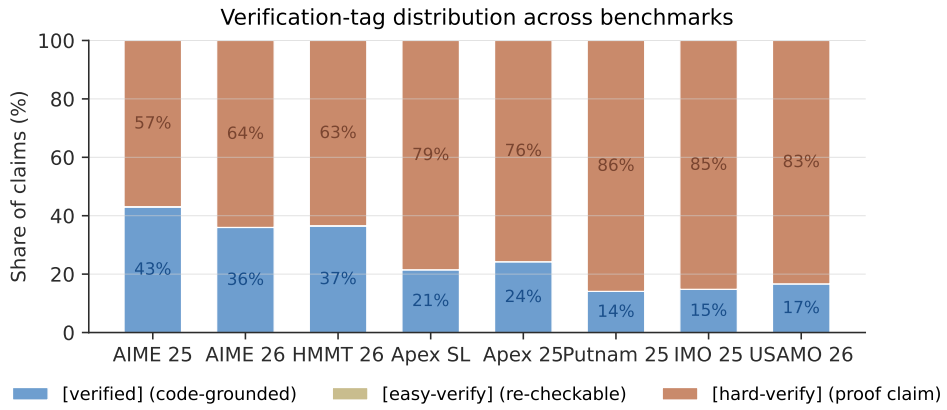

Figure 4: Distribution of verification tags across benchmarks, normalized to 100% of claims per benchmark. Final-answer benchmarks (left) carry a substantially larger share of code-grounded `[verified]` claims; proof-based benchmarks (right) shift toward `[hard-verify]` claims that require mathematical review.

Step 4's objective — to prove $a_1 + a_2 > \frac{3}{4}a_3$ universally — repeatedly timed out. Each Reasoner attempt set up a local discharging argument over boundary-adjacent square types but ran into unbounded propagation involving two-edge squares with opposite boundary edges. Three successive Step-4 attempts ended in timeouts, each triggering a Verifier-then-Meta-Strategist trace-back. The first trace-back diagnosis read:

> *Trace-back to Step 4: Prove the universal lower bound. The timeout is not a computation problem; the Reasoner's partial report shows the current* ***purely local discharging plan is structurally insufficient*** *because the double-supported a_2 obstruction can propagate. The next attempt should stop trying to close the same local table and instead model the obstruction chains globally.*

E.2 Three trace-backs and the re-plan verdict

All four interventions below originated from the *same* persistent Meta-Strategist session. This is the mechanism the paper highlights in Section 4.4: a non-persistent supervisor that sees only one timeout at a time would treat each as a local proof gap, but evidence accumulated within a single session reveals a structural pattern.

First trace-back (Step 4).

> *The timeout is not a computation problem; the current purely local discharging plan is structurally insufficient because the double-supported a_2 obstruction can propagate.* ACTION_TYPE: TRACE_BACK*;* TRACE_BACK_TO: *4;* USE_PURE_REASONING: *YES.*

Second trace-back (Step 4).

> *The failure is strategic, not computational: the local discharging table keeps running into the same unbounded strip-propagation obstruction. The Reasoner should stop extending the local table and convert the obstruction into a component-level accounting problem.* ACTION_TYPE: TRACE_BACK*;* TRACE_BACK_TO: *4;* USE_PURE_REASONING: *YES.*

Third trace-back (Step 4).

> *The failure is localized to Step 4's execution method: repeated timeouts indicate the current computational attack is infeasible within the step budget, but there is not enough evidence that the earlier plan structure or prior verified steps are unsound. Re-execute Step 4 from scratch with a revised, more structural or targeted approach rather than approving a full re-plan.*

After the third trace-back also yielded a timeout, the Meta-Strategist promoted the diagnosis from local-repair to plan-level: the bound itself was false, not unproved. The re-plan verdict reads:

> ***Plan summary.*** *Plan vK pursued a strategy anchored on proving a universal $3/4$ lower bound, likely treating that value as the candidate sharp threshold or final answer. This fails because the proposed universal bound is false, so further patching of the same inequality direction would continue to reinforce an invalid target.*
>
> ***Reason.*** *The current plan is unsound because its answer-bearing target was to establish a universal lower bound of $3/4$, and the Reasoner now reports that this bound is false rather than merely unproved at Step 4. This is not a local proof gap suitable for re-executing one step; the target itself must be replaced.*

The verdict carried explicit *forbidden directions* (verbatim):

```
- Do not assume 3/4 is the answer or the correct universal lower bound.
- Do not start the next plan by trying to repair the proof of the
  3/4 lower bound.
- Do not prove any inequality whose main target is a universal RHS of
  3/4 before independently re-establishing the correct candidate value.
- Do not reuse lemmas whose only role is to force or certify the false
  3/4 bound.
- Do not restrict construction search to cases consistent with the 3/4
  target; the next plan must actively look for a replacement target
  from fresh evidence.
```

These directives are injected into every subsequent Reasoner spawn through the `PROBLEM_STATE.md` hook (Appendix B.3), so v2 cannot silently re-anchor on $3/4$.

E.3 Plan v2: the successful $1/2$ trajectory

Plan v2's seven steps are:

1. Fix the exact combinatorial model.
2. Split the counts into inside and outside contributions.
3. Find the correct candidate constant from fresh evidence.
4. Prove the universal lower bound.
5. Make the discharging proof independent of shape pathologies.
6. Construct polygons approaching equality.
7. Conclude sharpness and the largest value.

Step 3 (new candidate). Re-running candidate search on a denser 4×4 motif (rows `#.#.`, `####`, `#.#.`, `....`) tiled $n \times n$ produced

$$a_1 = 9n, \quad a_2 = 4n^2 - n, \quad a_3 = 8n^2 - n,$$

and for $n = 50$ the ratio is $208/399 \approx 0.5213$ — already well below $3/4$ at $n = 5$.

Step 4 (cap-map proof). The verified lemmas establishing $a_3 \le a_1 + 2a_2$ are recorded in the Verified Results ledger:

> *[Lemma] Around any lattice vertex, the four adjacent squares cannot alternate inside-outside-inside-outside cyclically.*
>
> *[Lemma] If $e(Q) = 3$ and $f(Q)$ is the square across the boundary side opposite the unique same-colored neighbor of Q, then $e(f(Q)) \in \{1, 2\}$.*
>
> *[Lemma] For any square R with $e(R) \in \{1, 2\}$, at most $e(R)$ three-edge squares map to R under the cap-target map.*
>
> *[Lemma] Consequently, $a_3 \le a_1 + 2a_2$.*

The cap-map inequality was independently `[verified]` by a Step 4 verification script, which reported *zero failures over* $6{,}234$ *generated simply connected polyominoes up to size* 10, plus all listed motif cases.

Step 5 (pathology audit). A Step 5 audit script confirmed that the cap-map argument is independent of shape pathologies (interior ears, exterior notches, opposite-edge two-squares), reporting zero bad cap targets and zero capacity failures across the audit.

Step 6 (sharpness construction). A bridged 4×4-motif family $\{S_n\}$ gives admissible simple lattice polygons with

$$(a_1, a_2, a_3) = (11n-2,\ 4n^2+n-2,\ 8n^2-3n+2), \quad \frac{a_1+a_2}{a_3} = \frac{1}{2} + \frac{27n-10}{16n^2-6n+4} \xrightarrow{n\to\infty} \frac{1}{2}^{+}.$$

An independent verifier script confirmed the formula and validity for $n = 1, \ldots, 40$.

Step 7 (conclusion). Combining Step 4 (every admissible polygon satisfies $a_1 + a_2 > \frac{1}{2}a_3$) and Step 6 (a family realizing the ratio $\to 1/2$), no $k > 1/2$ works and $k = 1/2$ does. Plan v2 was accepted at Step 7 without further intervention.

E.4 Final solution

The accepted answer is $\boxed{k = \frac{1}{2}}$. The cap-map argument from the final solution reads:

> Color squares inside P black and squares outside P white; a boundary edge separates unlike colors. Around any lattice vertex the four squares cannot alternate black-white-black-white (else four boundary edges meet there).
>
> Take any square Q with $e(Q) = 3$; it has a unique same-colored neighbor. Let $f(Q)$ be the square across the boundary edge *opposite* that neighbor. By the alternation impossibility, $e(f(Q)) \in \{1, 2\}$. For any target R with $e(R) \in \{1, 2\}$, at most $e(R)$ three-edge squares cap onto R, so
>
> $$a_3 \le a_1 + 2a_2.$$
>
> The lowest horizontal boundary edge of P certifies $a_1 > 0$; hence
>
> $$a_3 \le a_1 + 2a_2 < 2(a_1 + a_2), \quad \text{i.e.,} \quad a_1 + a_2 > \tfrac{1}{2}\, a_3.$$
>
> Combined with the 4×4 motif construction approaching ratio $1/2$ from above, the largest k is $1/2$.

Full step reports, debate transcripts, and verification scripts are archived in the supplementary code release.

F Extended Case Study II: Apex 2025 Problem 12 (IMO 2025 Problem 6)

Apex 2025 Problem 12 is identical to IMO 2025 Problem 6, the 2025×2025 tiling problem. It is the hardest combinatorics problem in our benchmark suite. STAR-PólyaMath solved it end-to-end in a single run: the accepted solution arrived in plan version 3, with 8 steps, 12 cumulative trace-backs, 2 re-plans, and a total wall-clock time of 8h 18m.

F.1 Problem and answer

> Consider a 2025×2025 grid of unit squares. Matilda wishes to place on the grid some rectangular tiles, possibly of different sizes, such that each side of every tile lies on a grid line and every unit square is covered by at most one tile. Determine the minimum number of tiles Matilda needs to place so that each row and each column of the grid has exactly one unit square that is not covered by any tile.

The accepted answer is $\boxed{2112}$.

F.2 Plan and trajectory

The accepted plan v3 has eight steps:

1. Recast the grid condition exactly.

2. Replace tilings by a rectilinear-geometry invariant.
3. Specialize good chords to permutation geometry.
4. Translate chord selection into a bipartite matching problem.
5. Independently search for the extremal construction.
6. Prove the construction's upper bound geometrically.
7. Prove the matching lower bound for every permutation.
8. Assemble the final equality.

Reformulation phase (Steps 1–3). Step 1 establishes that any valid configuration has uncovered set $U_\pi = \{(i, \pi(i)) : 1 \le i \le 2025\}$ for some permutation $\pi \in S_{2025}$, and conversely; the problem reduces to $\min_\pi T(\pi)$, where $T(\pi)$ is the minimum number of rectangles partitioning the complement of U_π.

Step 2 entered a multi-round debate. The Verifier challenged an initial formula $T(\pi) = r(\pi)/2 + h(\pi) + 1 - \nu(\pi)$, citing a small-case counterexample (the $n = 3$ identity permutation gave a predicted 3 tiles vs. a true count of 4). The Reasoner revised the formula to

$$T(\pi) = r(\pi) + c(\pi) - h(\pi) - \nu(\pi),$$

where r counts resolved reflex corners, c counts covered connected components, h counts interior holes, and $\nu(\pi)$ is the maximum number of pairwise non-crossing *good chords*. The corrected formula was accepted after the debate.

Step 3 specializes good chords to permutation geometry: a horizontal chord exists between rows i and $i + 1$ exactly when $|\pi(i + 1) - \pi(i)| \ge 2$, with an analogous criterion for vertical chords on adjacent columns. Compatible (pairwise non-crossing) chords form an independent set in a bipartite incompatibility graph G_π.

Reduction phase (Steps 4–5). Step 4 applies König's theorem to the bipartite graph: $\nu(\pi) = |\mathcal{H}| + |\mathcal{V}| - \mu(G_\pi)$ where $\mu(G_\pi)$ is the maximum matching. Combining with Step 2 yields

$$T(\pi) = n + a(\pi) + \mu(G_\pi) - 1,$$

where $a(\pi)$ counts diagonally adjacent pairs of consecutive uncovered cells and $n = 2025$.

Step 5 entered a second multi-round debate. The Verifier rejected an initial bare existence claim of a 2112-tile construction and demanded an explicit permutation rule, computed value, and verification script outputs. The Reasoner responded with the construction: take $m = 45$ (so $2025 = m^2$) and define

$$\pi(am + i + 1) = im + (45 - a), \qquad 0 \le a, i < m.$$

This permutation, together with the four boundary rectangle families and $(m - 1)^2$ central rectangles, partitions the complement into exactly $4(m-1) + (m-1)^2 = 2112$ rectangles. A Step 5 verification script confirmed coverage and disjointness.

Lower-bound phase (Steps 6–7). Step 6 enumerates the 2112 rectangles explicitly, with a Step 6 verification script reporting `rectangles=2112`, `covered=4098600`, `expected_covered=4098600`, hence $\min_\pi T(\pi) \le 2112$.

Step 7 proves the matching lower bound for *every* permutation. Let $X \cup Y$ be a minimum vertex cover of G_π with $|X| = x$, $|Y| = y$; the x row cuts and y column cuts partition the grid into $(x + 1)(y + 1)$ zones. Each diagonal-adjacency component lies within a single zone (diagonal adjacencies admit no good chord), so $n - a(\pi) \le (x + 1)(y + 1)$, which algebraically forces

$$a(\pi) + x + y \ge 2\sqrt{n} - 2.$$

For $n = 2025 = 45^2$ this gives $a(\pi) + \mu(G_\pi) \ge 88$, hence

$$T(\pi) = n + a(\pi) + \mu(G_\pi) - 1 \ge 2025 + 88 - 1 = 2112.$$

Step 8 assembles the equality $\min_\pi T(\pi) = 2112$.

Re-plan history. The two re-plans recorded in `PROBLEM_STATE.md` both abandoned earlier targets that were directly falsified by computational evidence. The most consequential, recorded under "Confirmed Failures", reads:

> *Plan vK pursued a proof of a* 2700 *lower bound for the tiling quantity; this fails because Step 4 produced a verified* 2112*-tile construction, giving a concrete counterexample to the intended lower bound and invalidating the plan's main direction. The current plan is unsound because its planned* 2700 *lower bound has been directly falsified by a verified* 2112*-tile construction. This is not a step-local repair issue: the central target inequality of the plan is false, so tracing back within the same plan would keep the Reasoner anchored to an impossible bound.*

The accompanying forbidden direction ("Do not attempt to prove a 2700 lower bound or any lower bound exceeding the verified 2112-tile construction.") was injected into the next plan, ensuring the system did not re-anchor on the falsified target.

F.3 Key verified results

The Verified Results ledger records the following representative entries (verbatim from `PROBLEM_STATE.md`):

- **[Lemma]** (Step 1) Valid configurations are in bijection with permutations $\pi \in S_{2025}$ via $U_\pi = \{(i, \pi(i))\}$.
- **[Lemma]** (Step 2) $T(\pi) = r(\pi) + c(\pi) - h(\pi) - \nu(\pi)$.
- **[Computation]** (Step 3) A Step 3 verification script confirms the chord classification for all permutations of size $n = 2, \ldots, 6$.
- **[Computation]** (Step 4) A Step 4 verification script confirms the bipartite matching formula for $n = 2, \ldots, 7$.
- **[Computation]** (Step 5) The number of rectangles is $4(m-1) + (m-1)^2 = 2112$ for $m = 45$.
- **[Computation]** (Step 6) A Step 6 verification script confirms `rectangles=2112`, `covered=4098600`, `expected_covered=4098600`.
- **[Lemma]** (Step 7) For every $\pi \in S_{2025}$, $T(\pi) \geq 2112$, established by three independent Step 7 verification scripts (exact formula on small cases, vertex-cover slicing, matching bound).
- **[Answer]** (Step 8) The minimum number of rectangular tiles Matilda needs is 2112.

F.4 Final solution sketch

> **Upper bound (construction).** With $m = 45$ and $\pi(am + i + 1) = im + (45 - a)$, the complement of U_π is partitioned by four boundary rectangle families plus $(m-1)^2$ central rectangles, totaling $4(m-1) + (m-1)^2 = 2112$ rectangles. Hence $\min_\pi T(\pi) \leq 2112$.
>
> **Lower bound (matching argument).** For any $\pi \in S_{2025}$, $T(\pi) = n + a(\pi) + \mu(G_\pi) - 1$ where $a(\pi)$ counts diagonal adjacencies of consecutive uncovered cells and $\mu(G_\pi)$ is the maximum matching of the bipartite incompatibility graph of good chords. A vertex-cover slicing argument shows $a(\pi) + \mu(G_\pi) \geq 2\sqrt{n} - 2$. For $n = 2025 = 45^2$, this gives $a(\pi) + \mu(G_\pi) \geq 88$, so $T(\pi) \geq 2025 + 88 - 1 = 2112$ for every π.
>
> **Conclusion.** Upper and lower bounds coincide, so $\min_\pi T(\pi) = 2112$.

The complete proof, all step reports, debate transcripts, and verification scripts are archived in the supplementary code release.