AutoFyn Technical Report: Non-Parametric Expert Iteration for Long-Horizon Agents

Adib Hasan* Daniel Schaffield* Akashnil Dutta† Tarik Adnan Moon*

August 13, 2026

Abstract

We introduce AutoFyn, an agent harness inspired by the Expert Iteration algorithm, adapting a frozen model across many rounds by updating persistent state from verified reward signals rather than model weights. Each round begins from a fresh model session, and durable information is reintroduced only through explicit interfaces such as persistent memory files, reports, and repository state. Within a round, an orchestrator explores, plans and builds many alternative approaches with specialized agents, while a task-grounded verifier verifies the work and supplies an objective reward for measuring progress. This reward is distilled back into the persistent state, which updates the effective policy for the next round. In this technical report, we formalize this loop and describe its persistent state and verification interfaces. We then demonstrate its use in three domains, namely olympiad mathematics, data science, and cybersecurity. On the six fresh problems of the 2026 International Mathematical Olympiad, every model with room to improve scores higher under AutoFyn than in its provider's own coding agent. AutoFyn also built the top-ranked agent on the Spider 2.0 dbt benchmark, and has produced 16 maintainer-confirmed vulnerability advisories in Next.js, MetaMask, pnpm, Warp, LiteLLM, Langflow, and Open WebUI.
Code: https://github.com/SignalPilot-Labs/autofyn

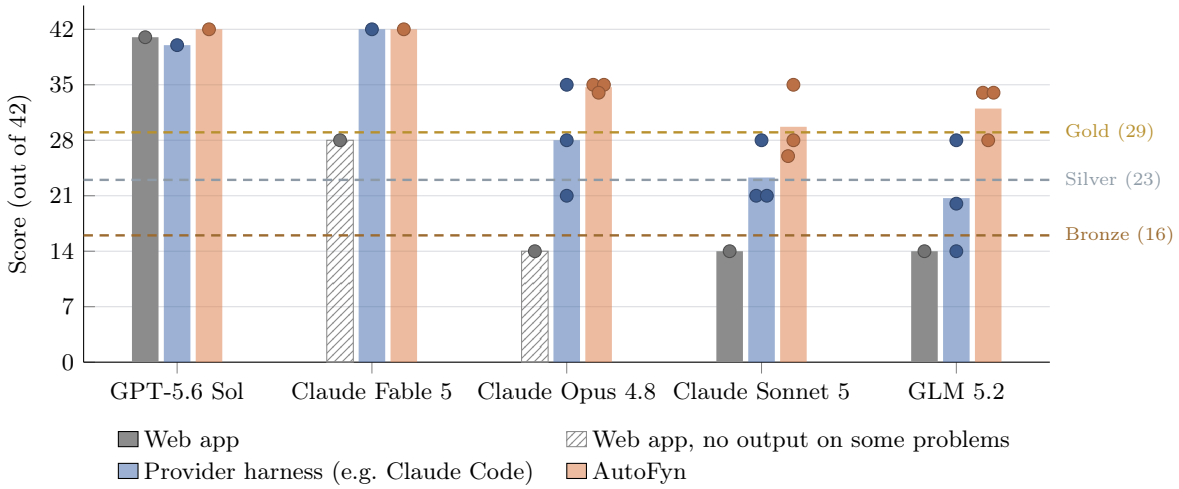
1 Introduction

In 2026, it has become normal for agentic systems to operate for many hours, and even days, to complete a complex task across hundreds of model calls and tool interactions. For example, coding agents resolve real world software issues that require coordinated edits across an entire repository [Jimenez et al., 2024, Yang et al., 2024], and iterative schemes such as reflection revise their own outputs over repeated attempts [Shinn et al., 2023, Madaan et al., 2023]. As the operational horizon grows, two failure modes recur in agentic systems. First, the context of a single agent accumulates until the model attends to it unevenly and its effective use degrades [Liu et al., 2024]. Second, when an agent evaluates its own work, an incorrect conclusion may pass that evaluation and condition later steps, since language models are unreliable at correcting their own reasoning without external feedback [Huang et al., 2024], so errors accumulate across the run.

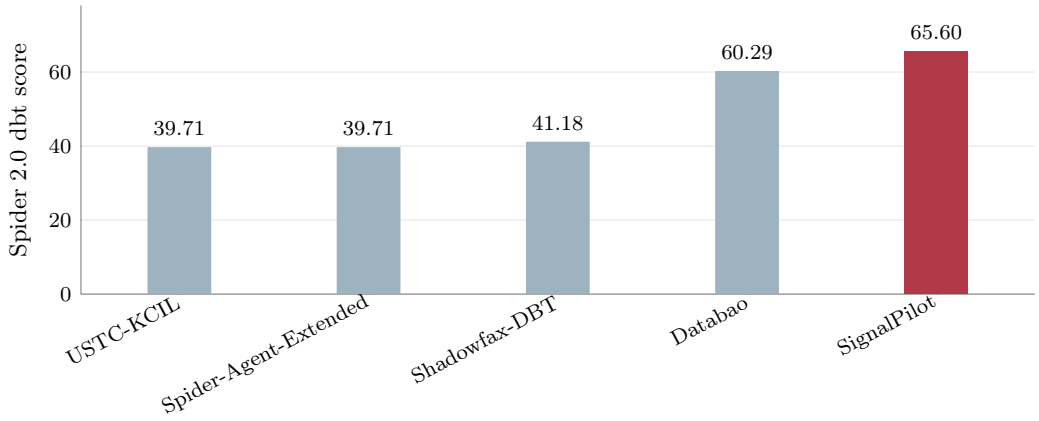
AutoFyn is designed around these two failure modes. To bound the context, AutoFyn uses an orchestrator worker model where a single orchestrator delegates work to subagents and records each round's progress to disk, then begins the next round in a fresh context initialized only from that record. To prevent errors from accumulating, the signal that closes each round is an external verification rather than the agent's own assessment. A round is treated as progress only when that signal confirms it. AutoFyn applies both together, so that what a round carries forward is bounded in size and confirmed outside the agent.

The resulting loop is analogous to Expert Iteration [Anthony et al., 2017], developed in Section 3. The correspondence, however, is structural rather than algorithmic, since AutoFyn is non-parametric and an adapted policy is not automatically a better one. This places AutoFyn alongside methods that pair a frozen model with an updated experience memory [Zhang et al., 2023].

*SignalPilot Labs
†Prentis AI



(a) IMO 2026, audited score out of 42 for each model under each harness. Bars are cell means, circles are individual runs, and the dashed lines mark the 2026 medal cutoffs.



(b) Spider 2.0 dbt, top five entries on the public leaderboard. The SignalPilot Agent, highlighted, was built and optimized autonomously by AutoFyn. Scores are as listed on 31 July 2026.

Figure 1: AutoFyn performance in olympiad mathematics and data science. The olympiad scores in (a) are the audited grades of our companion study [Hasan et al., 2026].

In this report, we describe AutoFyn and its performance across three domains, namely, olympiad mathematics, data science, and cyber security. On the six problems of the 2026 International Mathematical Olympiad, every model with room to improve scores higher under AutoFyn than in its provider's own coding agent (Figure 1a), and each run's process trail is fully archived. On the Spider 2.0 dbt benchmark for data science tasks, an agent AutoFyn built and optimized without human intervention tops the public leaderboard (Figure 1b). In security auditing, the loop has found over 150 individual vulnerabilities in widely used open source projects, of which we have submitted 43 advisories with 16 confirmed by their maintainers to date.

Our contributions are the following.

- AutoFyn as a system, a long horizon agent harness that composes context reset across rounds, search over candidate policies, an externally verified reward gate, and a distilled persistent state into a single loop.
- A formulation of this loop as a non parametric expert iteration, in which search proposes candidates, the external verifier is the expert criterion, and distillation into persistent state is the policy update.
- Results across three domains, an audited study on the 2026 International Mathematical Olympiad, a

top ranked data science agent on the Spider 2.0 dbt benchmark, and maintainer confirmed vulnerability advisories in widely used open source projects.

2 Related Work

Expert iteration. AutoFyn builds on expert iteration [Anthony et al., 2017], which alternates a search stronger than the current policy with an update that folds the search results back in. Self play systems realize this pattern with tree search as the expert [Silver et al., 2018], and recent work brings tree search to language model reasoning [Zhou et al., 2024]. AutoFyn instead plays the expert with a verifier grounded loop of review and revision, and its update adapts non parametric context state rather than model parameters or a search tree. Methods that pair a frozen model with an updated experience memory [Zhang et al., 2023] share this weightless adaptation, and reflection revises its own outputs without external grounding [Shinn et al., 2023, Madaan et al., 2023]. We are not aware of prior work that frames such a loop as expert iteration, and Section 3.4 states the correspondence and the point at which it stops holding.

Long context agents. It is known that long contexts are used unevenly [Liu et al., 2024] and researchers have studied ways to manage them. Proposed mechanisms include paged memory [Packer et al., 2023], hierarchical working memory [Hu et al., 2024], explicit context management for coding agents [Liu et al., 2025], and history folding for long running web and software tasks [Sun et al., 2025, Ye et al., 2025]. Closest to our own is the Ralph loop [Huntley, 2025], which reinvokes a coding agent on the same prompt indefinitely, so each iteration starts an empty context and state survives only through the repository, a plan file, and a specification directory. The shell loop itself reads no exit status, and continuation rests on human judgment of the repository. AutoFyn scores each round with an external verifier, and the orchestrator carries an artifact forward only when that score improves on the best so far (Equation 3). Fresh contexts, summarization, and bounded memory each predate our work, and AutoFyn contributes their composition with hard round boundaries, provenance bearing state, and grounded acceptance.

Verification. Verification and refinement pipelines have reported strong olympiad results by iterating a model against structured checks [Huang and Yang, 2025]. AutoFyn shares this verifier driven shape and differs in its explicit persistent state and its typed verification records. What a check establishes also depends on the harness around it, since the design of the agent computer interface shapes agent behavior in software engineering [Yang et al., 2024]. Recent work measures the harness and the model together rather than the model alone, benchmarking harness configurations across model backends [Yao et al., 2026] and transferring a single optimized harness across five held out models on olympiad level problems [Lee et al., 2026].

3 Method

3.1 Problem Setting

Let G be the goal, E the environment, and B the budget expressed as wall clock time and a number of rounds. Let $\mathcal{R}$ be the set of roles and $\Theta = \{\theta_r : r \in \mathcal{R}\}$ the assignment of a base language model to each role. The parameters in Θ stay frozen for the entire episode, so roles may resolve to different model tiers but no assignment changes during a run. Where one model serves every role, Θ reduces to a single frozen π_θ. Let M_t be the persistent state at round t, written to disk and reconstructed into a fresh context each round. Let V be the verifier the environment exposes, which maps a candidate artifact a to a typed record

$$y = V(a; E) = (s, e, \ell, v), \tag{1}$$

with status or score s, evidence bundle e, assurance class ℓ, and verifier identity v. The assurance class orders evidence by strength, from model self assessment through separate context review, executable checks, and independent human grading, up to a machine checked certificate. A later round reads the record rather than a bare accept or reject, so it acts on the reason a candidate failed.

A round spends part of B and produces one or more candidate artifacts together with a distilled update to the state. The episode ends under the control policy of Section 3.9, and the reported output is the incumbent artifact retained under the acceptance rule of Section 3.7.

3.2 Core Design Principles

Fresh sessions over explicit state. Every round begins a fresh model session. The previous round's conversation is not reloaded, and information returns only through a fixed set of durable interfaces, namely the persistent memory files, the repository and the artifacts it references, prior reports when requested, and the record of user activity. These carry provenance with them, so an evaluation outcome arrives with the evidence behind it and a failed approach arrives marked as failed. Context size therefore tracks the state rather than the elapsed run.

Search over candidates. Rather than commit to a single line of attack, the orchestrator dispatches role specialized agents to explore alternative approaches, produce competing plans or proof strategies, build one or more candidates, and review them, routing a rejected build back to the builder and a rejected plan back to the planner. This branching over approaches and revisions is the search half of expert iteration, structured by role and by phase rather than by an explicit search tree, with its breadth in a round set by the budget.

Grounded acceptance. A round is judged by evidence from the environment rather than by the model's assessment of its own output (Section 3.6). Self assessment may steer the search within a round, since the model must decide what to try next, but it cannot establish that the goal has been met. Language models are unreliable at judging their own reasoning without external feedback [Huang et al., 2024], so AutoFyn does not let an agent certify its own success.

3.3 The AutoFyn Round

An orchestrator drives each round. It reads the persistent state, decides the highest value step toward the goal, and routes that step through exploration, planning, an optional plan review, building, and build review, each served by a role specialized agent. It does not explore the codebase, design solutions, or write code itself beyond small fixes. Whether any two agents run concurrently depends on how the dispatch was issued, so generating and comparing alternative candidates is branching search even when the branches run in sequence.

Control of the loop lives outside the model, in a surrounding process. That process starts a fresh orchestrator session per round, reads persistent metadata, archived memory, user activity, and the previous round's report index to construct the round, and after the round records the round summary, commits and pushes the work, and decides whether to begin another round. The git save and push behavior is performed by the harness rather than by the orchestrator, and per round reports together with the memory directory are archived outside the sandbox and restored on resume. Algorithm 1 states the loop.

3.4 Non-Parametric Expert Iteration

Define the contextual policy induced at round t by

$$\pi_{\Theta}^{M_t}(x \mid h) = \Pi\big(\Theta, K(G, M_t, h)\big), \tag{2}$$

where K is the context construction procedure, h is the within round interaction history, and Π composes the role assigned models into the orchestrated search. The parameters in Θ are fixed, yet changing M_t changes the distribution over subsequent behavior, so $M_{t+1} \neq M_t$ implies $\pi_{\Theta}^{M_{t+1}} \neq \pi_{\Theta}^{M_t}$ in general. The policy thus changes across rounds with no parameter update, which is a contextual update rather than training in the usual sense.

Algorithm 1 The AutoFyn round loop

input: goal G, frozen role assignment Θ, orchestration procedure Π, verifier V, budget B
initialize persistent state M_0 from G; incumbent $I_0 \leftarrow \varnothing$
$t \leftarrow 0$
while control policy permits continuation **do**
    $C_t \leftarrow K(G, M_t, E_t)$ ▷ fresh session; durable state only, no inherited transcript
    $(\tau_t, \mathcal{A}_t) \leftarrow \mathcal{S}(\Theta, \Pi, C_t; B_t)$ ▷ orchestrated search; one or more candidates
    $y_{t,i} \leftarrow V(a_{t,i}; E_t)$ for each $a_{t,i} \in \mathcal{A}_t$ ▷ evidence bearing records
    $a_t \leftarrow A(\mathcal{A}_t, \{y_{t,i}\}_i, I_t)$ ▷ within round selection
    $I_{t+1} \leftarrow S(I_t, a_t, y_t)$ ▷ cross round incumbent update
    $M_{t+1} \leftarrow U(M_t, \tau_t, a_t, y_t)$ ▷ lossy distillation into typed state
    archive reports and repository state; discard the live model session
    $t \leftarrow t + 1$
end while
return incumbent I_t

Table 1: Structural correspondence between classical expert iteration and AutoFyn.

Classical expert iteration	AutoFyn
Apprentice policy π_{θ_t}	Contextual policy $\pi_{\Theta}^{M_t}$
Apprentice proposes candidates	Explore, plan, and build produce candidate artifacts
Expert procedure	Verifier grounded review and revision
Grounding signal within the expert	Grounded verifier V
Expert target	Selected accepted candidate and its findings
Parameter update θ_{t+1}	Persistent state update M_{t+1}
Updated apprentice	Newly conditioned policy $\pi_{\Theta}^{M_{t+1}}$

3.5 Comparison with Classical Expert Iteration

Classical expert iteration has three parts, an apprentice policy, an expert procedure that spends extra compute to produce actions stronger than the apprentice's, and an update that folds the expert's targets back into the apprentice. AutoFyn instantiates all three within a round, as Table 1 sets out. Its expert is the review and revision loop, which grades a candidate against the environment, diagnoses the failure, and routes the revision back to planning or building until a candidate is accepted. The verifier V grounds that loop rather than constituting the expert itself, and the expert spends more compute than a single apprentice sample, across several candidates and rounds of revision. The update is the distillation U, which folds the accepted candidate and its findings into M_t rather than into the weights.

The correspondence is structural rather than algorithmically identical. AutoFyn optimizes no parameters and constructs no explicit expert policy, updating instead the non parametric state that conditions later actions. It also lacks the guarantee that makes a game playing expert reliable, since a Monte Carlo tree search provably improves on its apprentice whereas AutoFyn's verifier can accept a wrong candidate or reject a right one. Improvement is therefore established by later verified performance rather than assumed from the update.

3.6 Verification

The environment produces the signal, not the model, so a check is reproducible from the artifact and the environment alone. Each one is also narrow. A passing test speaks only to its cases, a benchmark only to its evaluator, and a failed refutation only to the cases it searched, so the record names the property it checks and leaves the rest open.

In data science the check is the benchmark evaluator, which scores a submission against held out expected outputs and returns a per case breakdown. The breakdown rather than the bare score drives the next round,

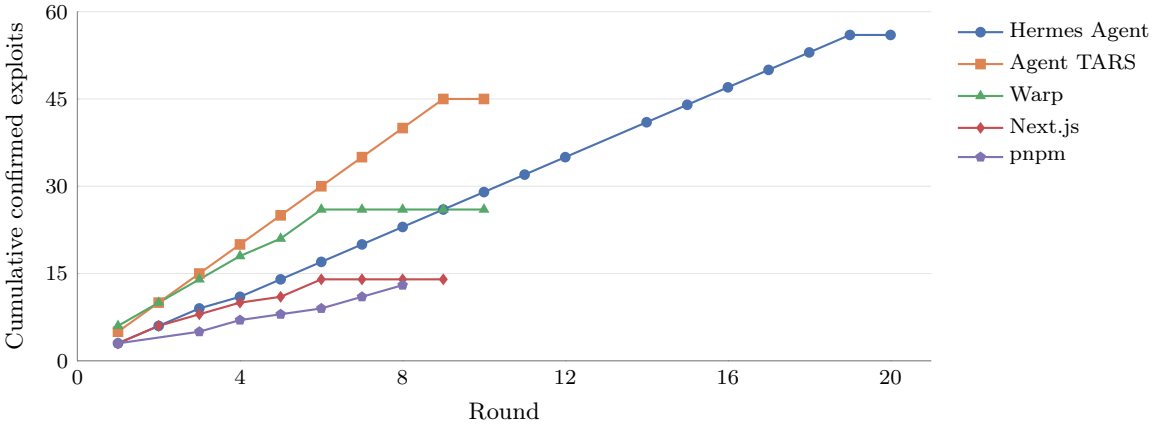


Figure 2: Cumulative confirmed exploits by round in five security audit runs, each an independent AutoFyn episode against a different target. A round adds to the count only when a new exploit fires against the target build, so the curves are monotone by construction and flatten once a run turns to consolidating what it has. The pnpm run is truncated at round 8, after which it spent a long stretch without a confirmed addition before closing a final chain at round 129.

since a failed case list tells it where to look. In security auditing the check is executing the candidate exploit against the target build, which demonstrates a reachable path rather than the full scope or severity of the vulnerability. Regression tests pass or fail throughout and are diagnostic either way.

In olympiad mathematics the check only refutes, since it cannot establish that the logic of a proof is sound. Where the proof claims a bound, AutoFyn searches small and degenerate cases for a counterexample. Where it claims an algebraic computation, a symbolic algebra package looks for a discrepancy. A counterexample refutes the step it targets and an unsuccessful search establishes nothing, so the signal is reproducible from the artifact rather than dependent on model judgment. This suffices at olympiad level, where the claims inside a proof are concrete enough to attack by search. Research level problems would need a proof assistant such as Lean.

3.7 Selection, Memory, and Incumbent Retention

Within a round, A selects among the candidates in $\mathcal{A}_t$ given their verification records, S updates the retained incumbent across rounds, and U distills the trajectory and outcomes into the persistent state. When candidates carry a comparable grounded score q, incumbent selection retains the better accepted artifact,

$$I_{t+1} = \begin{cases} a_t, & \text{accept}(y_t) = 1 \ \wedge \ q(a_t) > q(I_t), \\ I_t, & \text{otherwise}, \end{cases} \tag{3}$$

so that $q(I_{t+1}) \geq q(I_t)$ under deterministic comparable scoring. The orchestrator applies this rule when it writes the round's outcome into the persistent state, leaving I_t and the recorded goal untouched on a round that improves on neither, so the rule sits in the protocol layer of Table 2 rather than the harness layer. It bounds the incumbent, not the contextual policy, which carries no such guarantee since U is lossy.

Figure 2 shows the rule running in five security audits, where q counts the exploits confirmed against the target build. Each curve rises while the search finds new reachable paths and flattens once the run turns to consolidating what it has. The Hermes Agent run reaches 56 confirmed exploits over 19 rounds, while the pnpm run closes thirteen by round 8 and then spends a long stretch without a confirmed addition before a final chain at round 129. A flat stretch is a round that produced no accepted improvement and left the incumbent in place.

The persistent state is partitioned as $M_t = (G_t, H_t, R_t, W_t, I_t)$, comprising the goal and its user amendments, the evaluation history, the distilled rules and role specific lessons, the working status of hypotheses, failures, and next steps, and the incumbent artifact with its evidence. Each subagent role owns a memory

Table 2: Assurance level of representative system behaviors.

Behavior	Assurance
Fresh model session per round	Enforced by harness
Round transitions, archival, resume	Enforced by harness
Git commit and push after a round	Enforced by harness
Time lock on early termination	Enforced by harness
Frozen role to model assignment for a run	Enforced by harness
Orchestrator updates `run_state.md` each round	Required by protocol
Orchestrator delegates rather than codes directly	Required by protocol
Every build is reviewed before acceptance	Required by protocol
Verified facts kept distinct from hypotheses	Required by protocol
Concurrency of same type specialists	Observed per run

Table 3: How each application was run, with the verifiers as described in Section 3.6 and intervention counted over the episode alone.

Domain	Model	Budget	Verifier	Intervention	Evidence
Olympiad mathematics	several	20 h per run	refutation	none	audited grades
Data science	Opus 4.5/4.8	unlimited	per case evaluator	none	leaderboard
Security auditing	Opus 4.5/4.8	unlimited	exploit reproduction	stop a stalled run	advisory IDs

file into which it distills rules specific to that role, so a builder accumulates build knowledge and a reviewer accumulates review knowledge without either polluting the other. Rules are written as commands rather than observations, carry the round and reason they were learned, and are retired when the code they concern is gone. The working status keeps hypotheses, failures, and next steps in sections of their own, so a verified outcome stays distinguishable from speculation.

3.8 What the Architecture Enforces

Some of the behavior described above is enforced mechanically by the harness. The rest is requested through prompts and holds only insofar as an archived run confirms it. Table 2 sorts representative behaviors into these categories.

3.9 Stopping and Output Selection

Termination depends on whether the run is time locked. In an unlocked run the orchestrator may stop as soon as the goal is accepted. In a time locked run the harness denies an early end of session until the budget expires or the user intervenes, after which the loop may allow a single grace round for consolidation where configured. Episode duration is therefore not the time to reach the final candidate, which matters when reading the timings we report. The output is the incumbent retained under Equation 3, together with the evaluation history documenting how it was reached.

4 Applications

We report results in three domains, ordered by how strong a verifier the environment affords. In olympiad mathematics a proof cannot be executed and acceptance rests on refutation. In data science a benchmark evaluator settles it, and in security auditing a candidate exploit either fires against the target build or does not. Table 3 sets out how each was run.

Table 4: IMO 2026 totals out of 42 for each model in its provider's coding agent and in AutoFyn. The six sub-frontier cells are means over three independent runs, so those entries may be fractional. Scores are the audited grades reported in our companion study [Hasan et al., 2026], which describes the grading standard and the audit in full.

Model	Provider agent	Agent total	AutoFyn total	Gain
GPT-5.6 Sol	Codex	40.0	42.0	+2.0
Claude Fable 5	Claude Code	42.0	42.0	+0.0
Claude Opus 4.8	Claude Code	28.0	34.7	+6.7
Claude Sonnet 5	Claude Code	23.3	29.7	+6.3
GLM 5.2	Zcode	20.7	32.0	+11.3

4.1 Olympiad Mathematics: IMO 2026

The six problems of the 2026 International Mathematical Olympiad were released after every model's training cutoff. We ran five models on all six through three harnesses, the provider's web interface, the provider's own coding agent, and AutoFyn held fixed across every model, and graded the proofs under a completion-based standard that awards 7 only for a complete argument and 0 when any load-bearing step is left unproved. Each run was audited by a frontier model other than the one that wrote the proof, and a panel of three past IMO medalists reviewed those audits. All of this ran after the episodes had closed, so no grade reached the loop that produced the proof. The full matrix, the grading standard, and the analysis of where proofs fail are the subject of our companion study [Hasan et al., 2026], which archives every graded write-up, audit report, and process trail.

Table 4 sets each model's AutoFyn total against the same model in its provider's own coding agent. Every model with room to improve scores higher under AutoFyn than in its provider's agent, by 2.0 points for GPT-5.6 Sol and by 11.3 for GLM 5.2, which moves from 20.7 to 32.0 and crosses a medal band with no change to the model backend. Only Claude Fable 5 has no room, completing the contest in both harnesses. The gain is bounded above by the model, since Problem 3 goes unsolved in all eighteen sub-frontier runs across both agent harnesses. We do not isolate what produces the difference. The harnesses vary in episode length, tool access, retrieval, and multi-agent structure, so these totals compare deployments rather than the loop alone.

4.2 Data Science: Spider 2.0 dbt

The Spider 2.0 dbt benchmark scores an agent on realistic data transformation workflows against held out expected outputs. AutoFyn was given the goal of building and improving an agent for this task, and it constructed and optimized that agent autonomously across its rounds, using the benchmark's per case breakdown as the round signal. The resulting system, submitted as the SignalPilot Agent, holds the top position on the public leaderboard, and Figure 1b places its score against the next four entries.

4.3 Security Auditing

Here the verifier is the live exploit reproduction of Section 3.6, and a finding counts only once it fires against the target build. To date AutoFyn has found over 150 individual vulnerabilities across thirteen projects, among them Next.js, MetaMask, pnpm, Warp, LiteLLM, Langflow, Open WebUI, and RAGFlow. Related findings are bundled into a single advisory where they share a root cause, and we have submitted 43 advisories from them, of which 16 are confirmed by the maintainers of Next.js, MetaMask, pnpm, Warp, LiteLLM, Langflow, and Open WebUI, with the remaining 27 still open. Each confirmed advisory carries an identifier a reader can check, and the full per project record is maintained in the project repository.

Figure 2 plots five of these runs round by round.

5 Conclusion

We described AutoFyn, an agent harness that bounds each round's context to a distilled state and admits a round as progress only on an external verifier's signal. It is interpreted as expert iteration over a contextual policy and the loop adapts a frozen model by changing what constructs its context rather than what sets its weights.

We ran the same loop ran in three domains differing only by the available verifiers and found impressive results. We make AutoFyn available as infrastructure for long horizon agents, with an auditable trail of every run.

References

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Adib Hasan, Akashnil Dutta, and Tarik Adnan Moon. Beyond the model: How LLM performance varies with harness design on the 2026 international mathematical olympiad, 2026. Companion study. Full run archive at https://github.com/SignalPilot-Labs/IMO-2026.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. HiAgent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. *arXiv preprint arXiv:2408.09559*, 2024. URL https://arxiv.org/abs/2408.09559.

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*, 2024. URL https://arxiv.org/abs/2310.01798.

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. URL https://arxiv.org/abs/2507.15855.

Geoffrey Huntley. Ralph wiggum as a "software engineer". https://ghuntley.com/ralph/, 2025. Accessed 2026-07-31.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses, 2026.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638.

Shukai Liu, Jian Yang, Bo Jiang, Yizhi Li, Jinyang Guo, Xianglong Liu, and Bryan Dai. Context as a tool: Context management for long-horizon SWE-agents. *arXiv preprint arXiv:2512.22087*, 2025. URL https://arxiv.org/abs/2512.22087.

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*, volume 36, 2023.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404.

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

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024. URL https://arxiv.org/abs/2405.15793.

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, Xiangzheng Zhang, and Tong Yang. Harness-bench: Measuring harness effects across models in realistic agent workflows, 2026.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Siheng Chen, Jingren Zhou, and Yong Jiang. AgentFold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*, 2025. URL https://arxiv.org/abs/2510.24699.

Danyang Zhang, Lu Chen, Situo Zhang, Hongshen Xu, Zihan Zhao, and Kai Yu. Large language models are semi-parametric reinforcement learning agents. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/f6b22ac37beb5da61efd4882082c9ecd-Abstract-Conference.html.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.