FOUNDRY: Host-Owned Trust and Memory for Long-Horizon Agent Swarms

Monishwaran Maheswaran [*1,2] **Leon Lakhani** [*1] **Shu Liu** [1] **Yuqing Jian** [2] **Tianyi Zhang** [2]
Kurt Keutzer [1] **James Zou** [3,2] **Aditya Akella** [4] **Ben Athiwaratkun** [2] **Chenfeng Xu** [4,2]

Abstract

LLM-agent swarms can turn frontier models into discovery engines but fail when agents own the two states that determine progress: trusted evaluation and persistent memory. Without an external evaluator, agents reward-hack local reports and claim improvements that do not survive independent checking; without shared memory, agents waste budget re-discovering already-falsified hypotheses. We introduce FOUNDRY, a host-coordinated control plane whose principle is: agents propose, while the host verifies and remembers. The host owns an authoritative evaluator, an established-facts registry that compounds evidence across independent agents, and a hypervisor → orchestrator → solver hierarchy that routes work under token budgets. Across combinatorial mathematics, GPU-kernel engineering, and computational biology, FOUNDRY instantiates without per-domain changes: it tightens the Erdős minimum-overlap upper bound, achieves the fastest GPUMode TriMul kernel runtime on H100, and matches the state-of-the-art on the OpenProblems single-cell denoising benchmark. These results suggest that progress in agentic discovery depends not only on stronger agents, but on the system boundary that separates untrusted proposal generation from trusted verification and memory.

1. Introduction

Agentic AI is shifting from merely pursuing stronger base models toward asking how intelligence should be organized. Recent systems increasingly translate ideas from human organizations, biological collective behavior, and systems engineering into computational substrates for agents: roles, delegation, verification, memory, routing, and feedback. Swarm intelligence offers one such blueprint: classical swarm systems show how simple agents can produce adaptive collective behavior through local interaction, self-organization, and environmental feedback (Beni & Wang, 1993). The same organizing principle is now reappearing in LLM-agent discovery systems, where populations of agents search over programs, constructions, kernels, and experimental procedures, then use archives, validators, and feedback to turn individual attempts into collective progress. Systems such as FunSearch, AlphaEvolve, TTT-Discover, OpenEvolve, and ShinkaEvolve have already advanced mathematical bounds, GPU kernels, matrix-multiplication routines, computational-biology tasks, and combinatorial search problems (Romera-Paredes et al., 2024; Novikov et al., 2025; Yuksekgonul et al., 2026; Sharma, 2025; Lange et al., 2025).

Yet as these systems move from isolated attempts to long-horizon swarm runs, their system structure becomes the bottleneck. First, the trusted score is often operationally coupled to the agent: the same agent that proposes a candidate also runs local validation scripts, times the candidate, and reports the resulting number. Second, cross-attempt memory is usually organized as a flat archive of candidate *programs*, rather than a structured record of *hypotheses* that have been supported, contradicted, or falsified. It does not preserve the boundary of what has already been tried and ruled out. Actually, for long-horizon swarms, this negative knowledge is often the scarce resource.

These two design choices result in two compounding failure

Table 1. Headline results across three discovery benchmarks.

	Math C_5 (↓)	**Kernel** H100 μs (↓)	**Bio** Score (↑)
Best public	0.380927	1092	0.64
Best prior AI	0.380876	1014	0.73
FOUNDRY (ours)	**0.380863**	**989**	**0.73**

[1]UC Berkeley [2]Together AI [3]Stanford University [4]UT Austin. Correspondence to: Monishwaran Maheswaran <monishwaran@berkeley.edu>, Chenfeng Xu <xuchenfeng@utexas.edu>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

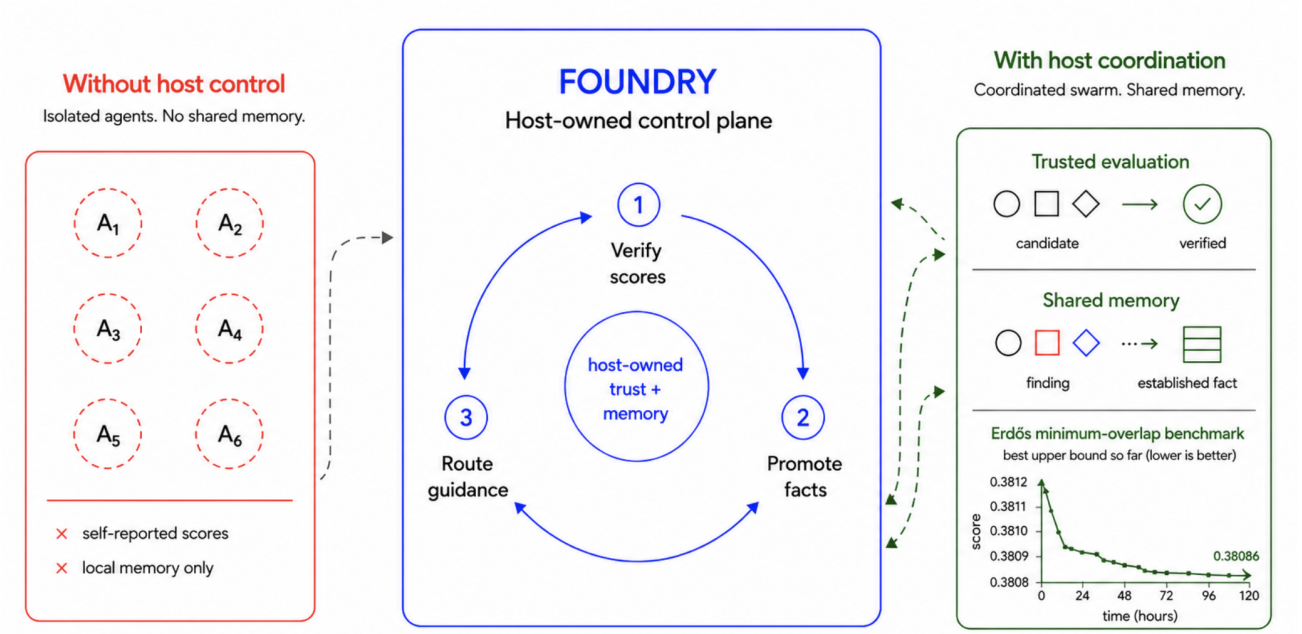

Figure 1. FOUNDRY's host-owned control plane. The coordinator loads state and launches the swarm; the problem adapter layer supplies schemas, prompt builders, a program bank, and the trusted-evaluate entry point. The host-owned control plane sits outside every solver container and exposes three components to the run: a trusted evaluator that recomputes scores, a facts registry that promotes findings from candidate to established, and an orchestrator that routes guidance. Containerized LLM coding-agent solvers receive prompts and guidance from the host and return artifacts and findings. The invariant that trust and shared memory remain host-owned is enforced by the dashed shared-state checkpoint boundary; the durability layer (process supervision and a run-level budget sentinel) keeps long swarm runs recoverable.

modes. **(1)** Collapse of the trust boundary: When the only evaluator the agent can see is its own local report, the agent can optimize the report rather than the underlying quantity. **(2)** Wasted re-derivation: When independent solvers work in parallel without host-owned shared memory, each agent privately rediscovers the same negative results, such as *"annealing past the local minimum does not help"* or *"step-like perturbations are unstable past resolution $n = 64$"*. A few thousand dollars of frontier-model agent time can therefore evaporate into redundant computation that the next agent will repeat. Existing multi-agent frameworks (Wu et al., 2023; Hong et al., 2024; Qian et al., 2024) pass messages between agents, but the message a discovery swarm most needs is not merely *"what should I work on next"*; it is *"what has every prior agent already shown does not work"*.

The field has spent the last year asking *"how do we train better discovery agents?"* The TTT-Discover, EvoTune, and ThetaEvolve lineages answer by adding training-time signal at inference time (Yuksekgonul et al., 2026; Surina et al., 2025; Wang et al., 2025b). We argue this is the second question. The first is structural: *"how do we organize frozen agents so they compound their own progress?"* Both failure modes above share a root cause: the agent owns the evaluator and the shared memory. Civilization's progress-compounding institutions take the opposite stance. Peer review uses parties with no interest in the result; open-source code review merges commits by reviewers who did not write them; replicated experiments are the only ones that count. The collapse of that separation into the agent itself is what produces both reward-hacking and wasted re-derivation. The training-free evolutionary lineage (Romera-Paredes et al., 2024; Novikov et al., 2025; Sharma, 2025; Lange et al., 2025) grounds candidates against an external scorer, which addresses the first half of this collapse, but its archive carries forward candidate programs, not falsified hypotheses, and the trusted scoring code typically lives inside the same process the agent runs in. The structural fix is to move both invariants out of the agent: the trusted score, and the cross-agent memory of what has already been ruled out.

We propose FOUNDRY, a host-coordinated metaharness for long-horizon agent swarms. FOUNDRY treats agents as frozen proposal generators: they may search, edit, test, and propose findings, but they do not own the trusted score or the persistent memory. Both are host-owned invariants. The host process owns three mechanisms that agents do not. First, it owns a trusted evaluator that re-derives every accepted score from first principles and never trusts the agent's reported scores or findings. The same evaluator is mounted read-only inside the agent container so that agents can self-check, but modifying local code cannot affect the

authoritative run. Second, the host manages the findings of the agent swarm, driving strategy diversity across the swarm by injecting select findings and instructions into subsequent agent prompts according to swarm-level signals. Third, the host owns a hypervisor → orchestrator → solver hierarchy that routes work through file-based mailboxes under a run-level token budget. Across both phased waves and long-lived swarms, the same invariant holds: *agents propose; the host verifies and remembers*.

In summary, we make the following contributions:

1. **From the trust perspective,** we identify a structural failure mode in long-horizon discovery swarms: agents are routinely allowed to own both the evaluator and the memory, collapsing the trust boundary and preventing progress from compounding across attempts. This motivates a strict separation in which the host re-derives every accepted score from first principles and never trusts the agent's reported number (Section 3).
2. **From the system perspective,** we introduce FOUNDRY, a host-coordinated metaharness for frozen frontier coding agents such as Claude Code and Codex, organized as a hypervisor → orchestrator → solver hierarchy that routes work through file-based mailboxes under a run-level token budget (Section 3). Its central design principle is that trust and memory live at the host, not in the agents.
3. **From the memory perspective,** we design a confidence-weighted established-facts registry with explicit promotion states (`candidate`/`supported`/`established`/`retired`) driven by independent support and contradiction, allowing future agents to inherit not only successful programs but also falsified hypotheses (Section 3).
4. Across three discovery benchmarks spanning combinatorics (Erdős minimum-overlap C_5), GPU kernel optimization (H100 TriMul), and scientific machine learning (OpenProblems Denoising), FOUNDRY delivers improvements without per-domain harness changes: a tighter upper bound on C_5, a faster TriMul kernel, and matches the current state of the art on OpenProblems Denoising (Section 4).

2. Related work

Context and memory management for long-horizon LLM agents. Recent LLM systems iteratively improve outputs by incorporating feedback from prior attempts. Editable-context and hierarchical-memory approaches (Zhang et al., 2026; Packer et al., 2024) organize past information across iterations; reflective methods (Shinn et al., 2023; Madaan et al., 2023) feed model-generated feedback into subsequent prompts; and test-time search and scaling (Yao et al., 2023; Snell et al., 2024) structure inference and the rollout-verifier mix. These approaches improve organization and reuse within a single session, but neither cross independent agent sessions nor separate the trusted scorer from the agent.

LLM-driven evolutionary search and discovery. A second line pairs LLMs with cross-attempt archives. FunSearch (Romera-Paredes et al., 2024) uses island-based program archives; AlphaEvolve (Novikov et al., 2025) adds MAP-Elites and multi-objective scoring across roughly fifty open mathematics problems; open-source replications and extensions (Sharma, 2025; Lange et al., 2025) build on AlphaEvolve with novelty-driven selection. AlphaTensor and AlphaDev (Fawzi et al., 2022; Mankowitz et al., 2023) train AlphaZero-style policies on specialized architectures for matrix multiplication and sorting. A learning-based thread adapts the generator via test-time RL or online DPO (Yuksekgonul et al., 2026; Surina et al., 2025; Wang et al., 2025b). K-Search (Cao et al., 2026) replaces the flat archive with a structured world-model search tree for GPU kernels. Across these threads, cross-attempt memory is an archive of candidate *programs* and the trusted scorer is typically a Python function inside the agent's own process.

Multi-agent orchestration frameworks. Multi-agent LLM frameworks (Wu et al., 2023; Li et al., 2023; Hong et al., 2024; Qian et al., 2024; Wang et al., 2025a) provide message-passing, role-assignment, and conversation-management primitives. They organize how agents talk to each other but neither contribute SOTA-advancing mechanisms nor enforce a host-resident trust boundary; they sit underneath FOUNDRY rather than competing with it.

FOUNDRY differs from these lines on two structural axes. The trusted evaluator is host-resident and unreadable from inside the container (§3); the agent's reported score is replaced by an independent host-side recomputation. The cross-attempt memory is a confidence-weighted hypothesis registry with promotion semantics (§3), not a flat archive: it carries forward falsified hypotheses, demotes them on cross-source contradiction, and re-injects survivors into every subsequent agent's prompt. The same coordinator drives three unrelated benchmarks under three `ProblemSpec`s with no per-domain code changes (§3), under both phased waves and long-lived swarms; the result is a training-free system that compounds progress across independent agents.

3. FOUNDRY

FOUNDRY implements three design choices that move trust and memory out of the agents. The trusted evaluator $\mathcal{E}$ is host-resident, with only a read-only mirror $\mathcal{E}_{\text{ro}}$ exposed

to each policy π_i. The cross-attempt memory $\mathcal{M}_t$ is host-curated: only the host writes to it, and only the injection ι exposes it to a future policy. The policies themselves are realized as two roles: a pool of M frozen LLM coding agents that act as *solvers*, and a single *orchestrator* that reads solver transcripts and curates $\mathcal{M}_t$. Algorithm 1 states the resulting discovery loop.

Trust boundary. The host owns the authoritative copy of $\mathcal{E}$ and re-runs it on every accepted candidate; the agent sees only $\mathcal{E}_{\mathrm{ro}}$, which lets a policy self-check its work but cannot affect what the host accepts. For each submission $a_i^{(t)} = (x_i^{(t)}, \hat{s}_i^{(t)}, F_i^{(t)})$, the host accepts the candidate iff

$$\Phi\big(x_i^{(t)}\big) = 1 \quad \text{and} \quad \big|\, \mathcal{E}(x_i^{(t)}) - \hat{s}_i^{(t)} \,\big| \le \varepsilon, \tag{1}$$

where Φ is a problem-specific schema validator. Mismatches are logged and discarded; findings $F_i^{(t)}$ are processed independently of acceptance, so a rejected candidate may still contribute hypotheses to $\mathcal{M}_t$.

Host-curated cross-attempt memory. The memory $\mathcal{M}_t$ preserves findings across agents and across rounds, so independent solvers do not re-derive the same negative results (e.g., *"warm starts upsampled by non-integer ratios inflate C_5"*). Because ι is the only inter-policy channel, no solver observes another's intermediate state; cross-agent influence flows entirely through findings the host has admitted to memory. FOUNDRY uses two realizations of the update operator U. In *swarm* mode, the orchestrator distills cross-solver evidence into broadcast messages that enter $\mathcal{M}_t$ when it promotes them. In *phased* mode, agents emit hypotheses tagged with per-agent endorsement weight $\sigma_+(h, v) \in [0.25, 3.0]$ and contradiction weight $\sigma_-(h, v) \in [0.25, 3.0]$, and the host applies the rule

$$\mathrm{label}(h) = \begin{cases} \text{retired} & \text{if previously deprecated,} \\ \text{candidate} & \text{else if } w_{\mathrm{challenge}}(h) \ge w_{\mathrm{support}}(h), \\ \text{established} & \text{else if } c_{\mathrm{score}}(h) \ge 3.0 \text{ and } |\mathcal{V}_+(h)| \ge 2, \\ \text{supported} & \text{else if } c_{\mathrm{score}}(h) \ge 1.5, \\ \text{candidate} & \text{otherwise,} \end{cases} \tag{2}$$

where $\mathcal{V}_+(h) = \{v : \sigma_+(h, v) > 0\}$ and $c_{\mathrm{score}} = w_{\mathrm{support}} - w_{\mathrm{challenge}}$. Both realizations share a structural invariant: a hypothesis enters the next policy's context only after independent corroboration by at least two distinct agents, which prevents any single confident agent from anchoring the run on a finding it cannot independently corroborate.

Per-problem specification. The same coordinator drives all three benchmarks; each problem supplies Φ, $\mathcal{E}$, prompt builders, and a metric direction, and no code path branches on the problem identity.

Algorithm 1 The FOUNDRY discovery loop.

Require: policies $\{\pi_i\}_{i=1}^{M}$, evaluator $\mathcal{E}$, schema validator Φ, baseline x_0, horizon T, tolerance ε

$\mathcal{M}_0 \leftarrow \emptyset$; $x^\star \leftarrow x_0$
for $t = 0, 1, \ldots, T-1$ **do**
  for policy i in parallel **do**
    $c_i^{(t)} \leftarrow \iota(\mathcal{M}_t, i)$; $a_i^{(t)} = (x_i^{(t)}, \hat{s}_i^{(t)}, F_i^{(t)}) \sim \pi_i(\cdot \mid c_i^{(t)})$
    accept $a_i^{(t)}$ iff $\Phi(x_i^{(t)}) = 1$ and $|\mathcal{E}(x_i^{(t)}) - \hat{s}_i^{(t)}| \le \varepsilon$ {Eq. 1}
  end for
  $\mathcal{M}_{t+1} \leftarrow U\big(\mathcal{M}_t, \{a_i^{(t)}, \mathcal{E}(x_i^{(t)})\}_i\big)$
  $x^\star \leftarrow \mathrm{argmin}\{\mathcal{E}(x) : x \in \{x^\star\} \cup \text{accepted}\}$
end for
return $x^\star$

4. Results

We evaluate FOUNDRY on three problems in three independent domains: discrete mathematics (Erdős minimum-overlap), GPU-kernel engineering (GPUMode TriMul on H100), and computational biology (OpenProblems single-cell denoising). For each problem, we report the best known human result and the best known AI result, and we compare FOUNDRY's host-verified score against both. Every accepted candidate in this section is independently re-evaluated outside the agent's container by the trusted evaluator (§3), so the column we report is directly comparable to the cited prior numbers when those were also produced under independent re-evaluation on the same hardware.

4.1. Erdős minimum-overlap

Task and setup. This is a classic problem in combinatorial number theory, posed by Erdős in 1955. Following AlphaEvolve and TTT-Discover, we work in the relaxed setting that asks for the smallest C_5 such that every step function $h : [0, 2] \to [0, 1]$ with $\int h\, dx = 1$ satisfies $\sup_k \int h(x)\,(1 - h(x + k))\, dx \ge C_5$. A candidate is a numerical array of length n representing such a step function, produced by an agent's thinking tokens followed by Python code that either constructs a new step function or modifies an existing one. The host parses and executes the code, renormalizes the result to satisfy $h(x) \in [0, 1]$ and $\sum_i h_i = n/2$, and recomputes the reward as $\max(\texttt{np.correlate}(h, 1-h, \texttt{mode=full}) \cdot 2/n)$, or returns ∞ if the artifact fails schema validation or contains no `generator_code`.

Result. Bounds before AlphaEvolve were $0.379005 < c < 0.380927$, with the upper bound due to Haugland (2016) and the lower bound due to White (2023); AlphaEvolve improved the upper bound to 0.380924 via verifier-based

Table 2. Erdős minimum-overlap upper bounds. FOUNDRY sets the new state-of-the-art at $C_5 = 0.380863$, advancing TTT-Discover's prior best by 1.3×10^{-5}.

Method	Model	C_5 (↓)	Notes
Best human			
Haugland 2016	—	0.380927	pre-AI human upper bound
Best known AI			
AlphaEvolve	Gemini 2.0 Pro+Flash	0.380924	training-free evolutionary search
TTT-Discover	gpt-oss-120b	0.380876	test-time RL on open model
FOUNDRY	Claude Code	**0.380863**	training-free, host-verified

evolutionary search, and TTT-Discover improved it further to 0.380876 via test-time RL on an open model. FOUNDRY improves the upper bound to **0.380863**, surpassing TTT-Discover's 0.380876 by an absolute margin of 1.3×10^{-5}. This advance is achieved with frozen frontier-model coding agents and no per-problem fine-tuning.

Discovered construction. The winning step function exhibits two structural features that FOUNDRY's registry promoted to `established` during the run: approximate antisymmetry $h(x) \approx 1 - h(2 - x)$, and low small-lag autocorrelation. Both were derived independently by multiple solver agents before promotion and were re-injected into the prompt of every subsequent solver. The discovery recipe alternates between sequential linear programming on a $\sigma = 0.10$ Gaussian-shaken warm-start and trust-constr polish at larger shake amplitudes; §5 walks through the trajectory in detail.

4.2. GPU-kernel engineering

Table 3. GPUMode TriMul runtimes on H100, geometric mean across the seven benchmark shapes; standard deviation across outer iterations of the same evaluator. FOUNDRY advances K-Search by $\sim 2.5\%$, the strongest public submission (`shiyegao`) by $\sim 9.4\%$, and TTT-Discover by $\sim 18.1\%$.

Method	Model	H100 (↓, μs)	Std (μs)
`zeyushen`	—	1365	±13
TTT-Discover	gpt-oss-120b	1208	±19
`shiyegao`	—	1092	±2
K-Search	GPT-5.2 + Gemini-3-Pro	1014	±3
FOUNDRY	Claude Code	**989**	**±4**

Task and setup. We apply FOUNDRY to the GPUMode TriMul benchmark, a single-H100 optimization of the triangular multiplicative-update kernel used in AlphaFold's Evoformer block (Jumper et al., 2021). The benchmark suite is the upstream `gpu-mode/reference-kernels` problem set with seven (sequence-length, batch, dim, hidden-dim) shapes; the score is the geometric mean of per-shape runtimes in microseconds. A candidate is GPU-kernel code in Triton or CUDA, produced by an agent's thinking tokens followed by kernel code that compiles and runs under the upstream evaluator on a single H100. The host reward is the negative geometric mean across the seven benchmark shapes, in microseconds; the kernel fails-closed if correctness checks fail or it times out.

Result. TTT-Discover targets the same kernel via test-time RL on gpt-oss-120b (Yuksekgonul et al., 2026), and K-Search uses a structured world-model search tree (Cao et al., 2026); `shiyegao` and `zeyushen` are public GPUMode H100 submissions. To make the margins directly comparable, we re-evaluate every comparator under a single H100 environment with the upstream TriMul evaluator; only kernels re-evaluated by us appear in Table 3. FOUNDRY's best kernel achieves a geometric-mean runtime of $\mathbf{989 \pm 4}\,\mu$s on H100, a $\sim 18.1\%$ reduction against TTT-Discover ($1208 \pm 19\,\mu$s), a $\sim 9.4\%$ reduction against the strongest re-evaluated public submission (`shiyegao`, $1092 \pm 2\,\mu$s), and a $\sim 2.5\%$ reduction against K-Search ($1014 \pm 3\,\mu$s). Because the agent has shell access and could in principle bias its own runtime measurement (warm caches, JIT artifacts), only reproducible runtimes under an evaluator the agent cannot read are directly comparable; FOUNDRY exposes that margin by design (§3).

Novelty. Three structural choices distinguish FOUNDRY's kernel from the comparators in Table 3.

Shape-aware fusion topology. The kernel selects a different fusion topology depending on the channel-to-hidden-dim ratio. When $C >$ `hidden_dim`, the dim$=384$ regime that dominates the geometric mean, it fuses the left projection, left gate, and out gate into a single three-output kernel that shares one x_{norm} load across all three, with a separate two-output kernel handling the right projection and right gate. When $C \leq$ `hidden_dim`, it instead uses two two-output projection-and-gate kernels and recomputes the out-gate dot product inline from x_{norm} in the final fused kernel, avoiding a third intermediate write. TTT-Discover and K-Search compile a single fixed proj-and-gate kernel that is reused on every shape; `shiyegao` ships a C++/CUDA extension that does not branch on shape; `zeyushen`'s leaderboard

kernel is a single Triton LayerNorm targeting one seqlen path. The conditional fusion saves one full x_{norm} read on the geomean-dominating shape ($\sim$ 770 MB, $\sim$ 256 μs of bandwidth) without regressing the smaller-C shapes.

End-to-end shape-keyed buffer cache. All intermediate fp16 tensors, including the output-projection buffer, are cached by a tuple-keyed shape signature, and the weights are cached fp32→fp16 through a weakref-keyed dictionary, so that repeated calls on the same shape skip every allocation and every type conversion. K-Search caches a subset of these buffers; TTT, `shiyegao`, and `zeyushen` allocate intermediate buffers fresh on each call. Adding the output-projection buffer to the cache (relative to the closest in-suite variant) accounts for the final $\sim$ 0.5–0.8% of geometric-mean reduction.

Pure Triton end to end. The pipeline is implemented entirely in Triton with cuBLAS only for the batched matmul. `shiyegao` relies on a C++/CUDA `torch.utils.cpp_extension` kernel ($\sim$ 2245 LOC); the FOUNDRY kernel is $\sim$ 391 LOC. Keeping the kernel in one mutable language preserved the option for solvers to iterate on the fusion topology and tile sizes locally, which is how the kernel arrived at the conditional split above.

4.3. Single-cell denoising

Task and setup. We apply FOUNDRY to the OpenProblems Denoising task from the OpenProblems benchmark suite (Luecken et al., 2025), which evaluates algorithms that recover true expression values from noisy UMI counts in single-cell RNA-sequencing data. Following Batson et al. (2019), the benchmark partitions observed molecules into training and test sets via binomial sampling and scores a denoised training matrix against the held-out test counts; this provides a proxy for accuracy against true expression values without requiring external ground truth. A candidate is a Python callable mapping a sparse count matrix to a dense denoised matrix of the same shape, produced by an agent's thinking tokens followed by code that the host parses and runs. The benchmark reports two complementary metrics: mean-squared error (MSE) in log-normalized space, which measures overall reconstruction accuracy, and Poisson negative log-likelihood, which assesses how well the denoised counts match the statistical properties expected of count data; each is normalized so that no denoising scores 0 and perfect denoising scores 1. The reward is the mean of the two normalized scores, or 0 if the algorithm exceeds the 400-second host time budget on either dataset or fails the Poisson constraint we add to focus search on MSE (normalized Poisson $\geq$ 0.97). The benchmark provides three datasets: PBMC, Pancreas, and Tabula Muris Senis Lung, in order of size; we use Pancreas for in-loop evaluation and report performance on the held-out PBMC and Tabula datasets.

Result. The strongest pre-AI human entries on the OpenProblems leaderboard are MAGIC (Van Dijk et al., 2018) with reversed normalization (and an approximate-solver variant) and ALRA (Linderman et al., 2022) with sqrt normalization; TTT-Discover (Yuksekgonul et al., 2026) produced the prior best AI result on this benchmark via test-time RL on `gpt-oss-120b` initialized from MAGIC code, and OpenEvolve provides additional non-RL coding-agent baselines reported in that work. FOUNDRY matches TTT-Discover within reporting precision: PBMC score 0.72 vs. 0.71, Tabula tied at 0.73, and per-dataset MSE tied at 0.15 on PBMC and 0.14 on Tabula, with the Poisson constraint passing on both held-out datasets. This is achieved with frozen Claude Code agents and no per-problem fine-tuning, against a baseline whose discovery recipe was test-time-trained directly on this benchmark.

5. Case Study: Coordinating a 17-agent Erdős swarm

We illustrate the coordination mechanism through one long-swarm run on the Erdős minimum-overlap problem, in which 17 solver agents and one orchestrator agent run in parallel for 119 wall-clock hours and submit 942 host-verified candidates. Solver agents are independent frontier-model coding sessions in separate containers with shell access; the orchestrator is a separate session that reads every solver's events and heartbeats, polls the host's verification feedback, and writes short natural-language guidance into a per-solver inbound channel. No agent is given an explicit search algorithm; each decides its own polish, shake, parameterization, and refinement strategy. Every step-down in the swarm-best trajectory (Figure 2) aligns with a single orchestrator broadcast, and each broadcast aggregates diagnostic signal that no individual agent could produce alone.

Phase 1, basin lock (hours 0–13). Diagnosed problem. Agents warm-start from public reference constructions, reaching $C_5 = 0.380870$ within minutes, then apply a wide repertoire of local optimization techniques (sequential linear programming, mass-transfer descent, projected gradient, smooth-max gradient, active-subgradient cutting plane). None descends below the warm-start floor; the basin appears locally locked. **Rationale.** If the basin is locally locked under continuous polish, the swarm needs structurally distinct perturbations to escape it; the orchestrator should canonicalize this diagnosis so agents stop redundantly polishing the same basin. **Mechanism.** The orchestrator aggregates cross-agent evidence (continuous polish is stationary up to 64K tangent-direction tests with a 21-D null space), publishes a basin-lock diagnosis to the swarm's shared scratch

Table 4. OpenProblems Denoising scores on the held-out PBMC and Tabula datasets. Each per-dataset score is the mean of the normalized MSE and Poisson metrics (higher is better). FOUNDRY matches the prior state-of-the-art set by TTT-Discover within reporting precision, tying it on per-dataset MSE (0.15 on PBMC, 0.14 on Tabula). Scores cited for prior work follow Yuksekgonul et al. (2026); (A) approximate solver, (R) reversed normalization, (S) sqrt normalization.

		PBMC			Tabula		
Method	**Model**	**Score (↑)**	**MSE (↓)**	**Poisson (↓)**	**Score (↑)**	**MSE (↓)**	**Poisson (↓)**
Best human							
MAGIC (A, R)	—	0.64	0.19	0.05	0.64	0.18	0.03
MAGIC (R)	—	0.64	0.19	0.05	0.64	0.18	0.03
ALRA (S, R)	—	0.50	0.26	0.05	0.47	0.27	0.03
Best known AI							
OpenEvolve	gpt-oss-120b	0.70	0.16	0.05	0.71	0.15	0.03
TTT-Discover	gpt-oss-120b	0.71	**0.15**	0.05	0.73	**0.14**	0.03
FOUNDRY	Claude Code	**0.72**	**0.15**	**0.05**	**0.73**	**0.14**	**0.03**

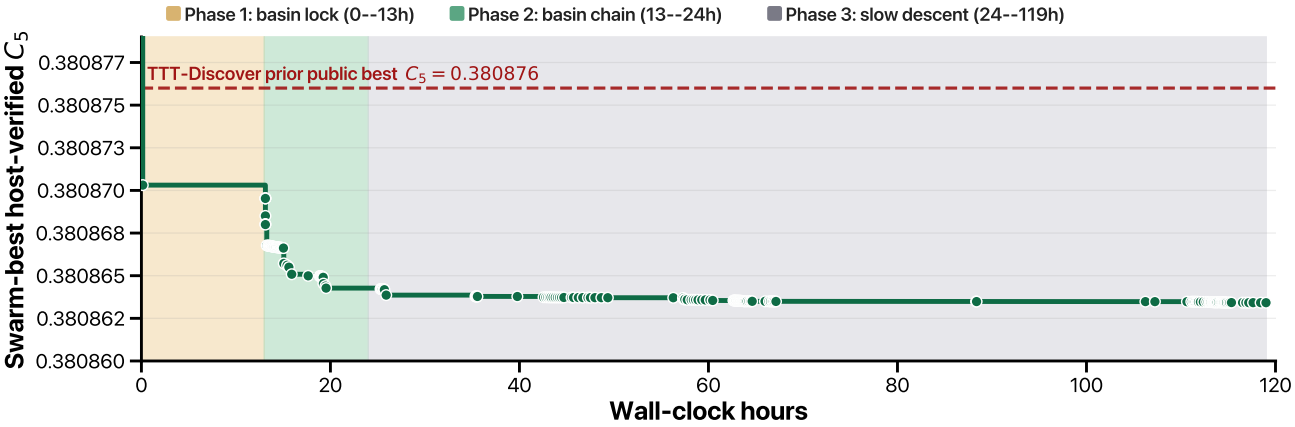

Figure 2. Swarm-best host-verified C_5 over the first 24 hours of an Erdős long-swarm run with 17 solver agents and one orchestrator agent. *Phase 1* (tan) is the basin lock: polishing public reference constructions plateaus at $C_5 \approx 0.380870$. *Phase 2* (green) is the basin chain triggered when one agent's $\sigma = 0.10$ Gaussian shake at hour 13 is broadcast by the orchestrator to all peers; six independent agents reproduce the recipe and the swarm-best descends in discrete steps to 0.380863. The chain saturates by hour ~ 20 and the floor holds for the remaining ~ 95 hours of the run (Phase 3, not shown). The horizontal dashed line marks the prior public best ($C_5 = 0.380876$, TTT-Discover (Yuksekgonul et al., 2026)); the run advances it by 1.3×10^{-5}. This case study illustrates the coordination mechanism behind the headline result in Table 2.

space, and recommends four alternatives: interpolate to higher n then polish; binarize at high n then run swap simulated annealing; random restart via a large-σ Gaussian shake; or a different parameterization (Fourier coefficients, dual on White's spectral lower bound). **Observed effect.** Agents partition across the four directions. Random restart at $n \geq 2400$ reaches $C_5 \in [0.41, 0.49]$; Fourier-CMA at $K = 40, n = 2400$ plateaus at ~ 0.4045. By hour 13 the swarm has eliminated restart-from-random and Fourier-CMA as productive directions.

Phase 2, basin chain (hours 13–24). Diagnosed problem. The remaining alternative is a large-σ Gaussian shake; only one agent has tried it, and only at $\sigma = 0.05$. **Rationale.** If the warm-start basin's attractor radius is small but the basin landscape is rich at larger perturbation scales, increasing the shake amplitude could identify new local minima; once any agent finds a new basin, the orchestrator can amplify the recipe to the swarm. **Mechanism.** At hour 13, one agent runs a Gaussian shake at $\sigma = 0.10$ on the warm-start vector, polishes with SLP, and discovers a sub-basin at $C_5 = 0.380867$ that is structurally distinct from the warm-start basin ($L_2 = 1.10$, $402/600$ coordinates differ, active-lag set differs in $252/437$ entries). The orchestrator broadcasts the recipe and recommends heavier shake at $\sigma \in [0.15, 0.30]$ from the new basin. **Observed effect.** Within the next hour, six independent agents reproduce the recipe. The chain descends in discrete steps from 0.380870 to 0.380863, each step driven by a different agent applying a larger-σ shake on the previous floor.

Phase 3, slow descent (hours 24–119). Diagnosed problem. The basin chain saturates at $C_5 \approx 0.380863$, 1.3×10^{-5} below the prior public best; additional shake iterations at σ up to 0.50 yield no new basins. **Rationale.** The shake-and-polish recipe has exhausted the local basin landscape; the swarm needs a qualitatively different construction family. **Mechanism.** The orchestrator partitions the swarm across (i) high-precision Fourier-CMA at $K \in [40, 60]$, (ii) a dual / Lagrangian formulation against White's spectral lower bound, (iii) deep local search with `trust-constr` from the basin floor, (iv) a genetic algorithm on the discrete h-vector. At hour ~ 28 the host evaluator goes silent for 7.5 hours due to an infrastructure issue; the supervisor keeps all 17 solver processes alive while the orchestrator marks candidates from that interval as unverified and re-issues verification once the host recovers. **Observed effect.** None of the four alternative parameterizations escapes the basin within the wall-clock budget. The run ends with swarm-best $C_5 = 0.380863$, advancing the prior public best by 1.3×10^{-5}.

What the trajectory shows. Every step-down on the trajectory follows a single orchestrator broadcast within minutes, and each broadcast canonicalizes a diagnosis (basin-locked, larger-σ escape, alternative parameterizations) that no individual agent derives from its own state. The supervisor sustains multi-day continuity through a 7.5-hour host-evaluator outage without derailing the run.

6. Limitations

FOUNDRY's guarantees rest on a host-owned evaluator that is faithful to the objective, together with a per-problem adapter that supplies the schema, trusted evaluator, and prompt builders for each new domain; problems without a fast and trustworthy evaluator therefore lie outside its present scope. The promotion rule and prompt design rely on empirical choices we have not formally characterized, and all reported results use frozen frontier coding agents at substantial compute cost. Whether host-owned trust and memory help smaller models close this gap is a question for future work.

7. Conclusion

We argued that the bottleneck in long-horizon agentic discovery is structural, not capability-bound: when agents are allowed to own the trusted score and the cross-attempt memory, the trust boundary collapses and progress fails to compound. FOUNDRY takes the opposite stance. Trust and memory are host-owned invariants: the host re-derives every accepted score from an evaluator the agent cannot modify, and curates a confidence-weighted registry that promotes a hypothesis only after independent corroboration. The same coordinator, registry, and hypervisor $\rightarrow$ orchestrator $\rightarrow$ solver hierarchy drive three unrelated benchmarks without per-domain code changes.

Under this single substrate, frozen frontier coding agents tighten the upper bound on the Erdős minimum-overlap constant to $C_5 = 0.380863$, achieve a $989\,\mu$s geometric-mean runtime on the GPUMode TriMul kernel on H100, and lower the mean-squared error on OpenProblems Denoising. Together, these results suggest that progress in agentic discovery depends not only on stronger proposal generators, but on the system boundary that separates untrusted proposal from trusted verification and memory.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Batson, J., Royer, L., and Webber, J. Molecular cross-validation for single-cell rna-seq. *BioRxiv*, pp. 786269, 2019.

Beni, G. and Wang, J. Swarm intelligence in cellular robotic systems. In *Robots and biological systems: towards a new bionics?*, pp. 703–712. Springer, 1993.

Cao, S., Mao, Z., Gonzalez, J. E., and Stoica, I. K-search: Llm kernel generation via co-evolving intrinsic world model, 2026. URL https://arxiv.org/abs/2602.19128.

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., R. Ruiz, F. J., Schrittwieser, J., Swirszcz, G., Silver, D., Hassabis, D., and Kohli, P. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022. doi: 10.1038/s41586-022-05172-4. URL https://doi.org/10.1038/s41586-022-05172-4.

Haugland, J. K. The minimum overlap problem revisited, 2016. URL https://arxiv.org/abs/1609.08000.

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., and Schmidhuber, J. Metagpt: Meta programming for a multi-agent collaborative framework, 2024. URL https://arxiv.org/abs/2308.00352.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021. doi: 10.1038/s41586-021-03819-2. URL https://doi.org/10.1038/s41586-021-03819-2.

Lange, R. T., Imajuku, Y., and Cetin, E. Shinkaevolve: Towards open-ended and sample-efficient program evolution, 2025. URL https://arxiv.org/abs/2509.19349.

Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., and Ghanem, B. Camel: Communicative agents for "mind" exploration of large language model society, 2023. URL https://arxiv.org/abs/2303.17760.

Linderman, G. C., Zhao, J., Roulis, M., Bielecki, P., Flavell, R. A., Nadler, B., and Kluger, Y. Zero-preserving imputation of single-cell rna-seq data. *Nature communications*, 13(1):192, 2022.

Luecken, M. D., Gigante, S., Burkhardt, D. B., Cannoodt, R., Strobl, D. C., Markov, N. S., Zappia, L., Palla, G., Lewis, W., Dimitrov, D., et al. Defining and benchmarking open problems in single-cell analysis. *Nature Biotechnology*, pp. 1–6, 2025.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., and Clark, P. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Mankowitz, D. J., Michi, A., Zhernov, A., Gelmi, M., Selvi, M., Paduraru, C., Leurent, E., Iqbal, S., Lespiau, J.-B., Ahern, A., Köppe, T., Millikin, K., Gaffney, S., Elster, S., Broshear, J., Gamble, C., Milan, K., Tung, R., Hwang, M., Cemgil, T., Barekatain, M., Li, Y., Mandhane, A., Hubert, T., Schrittwieser, J., Hassabis, D., Kohli, P., Riedmiller, M., Vinyals, O., and Silver, D. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023. doi: 10.1038/s41586-023-06004-9. URL https://doi.org/10.1038/s41586-023-06004-9.

Novikov, A., Vu, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J. R., Mehrabian, A., et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., and Gonzalez, J. E. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., and Sun, M. Chatdev: Communicative agents for software development, 2024. URL https://arxiv.org/abs/2307.07924.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J. R., Ellenberg, J. S., Wang, P., Fawzi, O., Kohli, P., and Fawzi, A. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6. URL https://doi.org/10.1038/s41586-023-06924-6.

Sharma, A. Openevolve: an open-source evolutionary coding agent. https://github.com/algorithmicsuperintelligence/openevolve, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Surina, A., Mansouri, A., Quaedvlieg, L., Seddas, A., Viazovska, M., Abbe, E., and Gulcehre, C. Algorithm discovery with llms: Evolutionary search meets reinforcement learning, 2025. URL https://arxiv.org/abs/2504.05108.

Van Dijk, D., Sharma, R., Nainys, J., Yim, K., Kathail, P., Carr, A. J., Burdziak, C., Moon, K. R., Chaffer, C. L., Pattabiraman, D., et al. Recovering gene interactions from single-cell data using data diffusion. *Cell*, 174(3):716–729, 2018.

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., Tran, H. H., Li, F., Ma, R., Zheng, M., Qian, B., Shao, Y., Muennighoff, N., Zhang, Y., Hui, B., Lin, J., Brennan, R., Peng, H., Ji, H., and Neubig, G. Openhands: An open platform for ai software developers as generalist agents, 2025a. URL https://arxiv.org/abs/2407.16741.

Wang, Y., Su, S.-R., Zeng, Z., Xu, E., Ren, L., Yang, X., Huang, Z., He, X., Ma, L., Peng, B., Cheng, H., He, P., Chen, W., Wang, S., Du, S. S., and Shen, Y. Thetaevolve: Test-time learning on open problems, 2025b. URL https://arxiv.org/abs/2511.23473.

White, E. P. A new bound for Erdős' minimum overlap problem. *Acta Arithmetica*, 208:235–255, 2023.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

Yuksekgonul, M., Koceja, D., Li, X., Bianchi, F., McCaleb, J., Wang, X., Kautz, J., Choi, Y., Zou, J., Guestrin, C., and Sun, Y. Learning to discover at test time, 2026. URL https://arxiv.org/abs/2601.16175.

Zhang, Q., Hu, C., Upasani, S., Ma, B., Hong, F., Kamanuru, V., Rainton, J., Wu, C., Ji, M., Li, H., Thakker, U., Zou, J., and Olukotun, K. Agentic context engineering: Evolving contexts for self-improving language models, 2026. URL https://arxiv.org/abs/2510.04618.