Market Incentivization for Theorem Proving with LLM Agents

Gregory Constantine [†1 2] **Leyan Pan** [†1 2] **Vijay Ganesh** [2]

Abstract

Agentic and multi-agent systems have been increasingly used for solving frontier-level math problems. We present preliminary experimental results with a market-based multi-agent system for proving mathematical theorems, in which agent coordination is mediated by market signals rather than a hand-engineered harness. Agent communities learn via evolutionary signals from market profits and losses. We report initial findings on whether evolutionary pressure produces more profitable agent populations and discuss remaining challenges towards an autonomous multi-agent math system that learns from markets.

1. Introduction

Recently, LLM-agent based systems have begun assisting with or performing research-level mathematics. On the informal side, coding-agent harnesses driven by frontier reasoning models have been used for mathematical exploration and discovery at scale (Novikov et al., 2025; Georgiev et al., 2025). On the formal side, dedicated theorem-proving and autoformalization agents now produce IMO-level proofs in Lean (Achim et al., 2025), formalize entire research papers and textbooks (Math, Inc., 2025; Gloeckle et al., 2026), and are supported by an expanding family of open prover models (Polu & Sutskever, 2020; Xin et al., 2025; Lin et al., 2025).

However, most current multi-agent systems rely on manually engineered harnesses that specify the roles, communication, and orchestration of different agents. Even at the scale of 30K coding agents formalizing a textbook in Lean (Gloeckle et al., 2026), the agent roles (sketcher, prover, maintainer, reviewer) and the merge protocol that combines their outputs are all hand-designed. As AI capabilities continue to progress, this planning is becoming the bottleneck for AI-driven mathematical research, and breaking past it means relaxing fine-grained human control and encouraging agent self-organization.

We argue that this self-organization is best structured as an incentive-driven market. Following Polanyi (1962), we view a multi-agent system with fixed roles and routing as a centrally-planned economy, which is effective when objectives are well-defined but poorly suited to driving innovation along paths no planner could have specified. Concretely, prices on intermediate artifacts give a decentralized credit-assignment signal that localizes reward to individual contributions, smoothing the sparse-reward problem that arises when only the final proof is rewarded. Specialization, division of labor, and task routing then emerge from agents responding to prices rather than from a designer assigning roles, and new agents can be swapped in and out without redesigning the system. This view aligns with a growing body of work that proposes agent markets as a primitive for multi-agent LLM coordination (Gho et al., 2025; Liu et al., 2026; Hadfield & Koh, 2025; Fradkin & Krishnan, 2026).

Mathematics is an unusually clean first testbed for such a market: ground-truth verification at the top level rules out the worst forms of reward hacking, and proofs decompose naturally into independently-checkable units (intermediate lemmas) that serve as tradeable intermediate goods. To this end, we propose Math Market, a market-driven multi-agent system aimed at mathematical theorem proving and autoformalization, in which final rewards stem from verified proofs while market mechanisms drive internal coordination and collaboration. Math Market follows an early proposal in this direction by Szegedy (2020) and is concurrent with recent bounty-based autoformalization efforts (Brown et al., 2026b; Carré et al., 2021; Brown et al., 2026a). Our implementation reuses public coding-agent harnesses (e.g., Claude Code, OpenCode) connected to the market through a set of MCP tools, so our contribution lies in the market mechanism rather than in the agent runtime.

In our early experiments, we focus on market-based incentivization of multi-agent theorem proving, limiting our scope to informal mathematics but with a more structural proof format that allows rigorous verification of independent inference steps. Under this setting, we run evolutionary experiments on agent communities to determine whether

[†]Work done at AletheAI. [1]AletheAI [2]Georgia Institute of Technology, Atlanta, GA, USA. Correspondence to: Gregory Constantine <gconstantine7@gatech.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).

agents can autonomously form community structures that result in better overall profit-seeking, and whether this translates to better mathematical capability.

2. Math Market Setup

2.1. Informal Verification of Proof Trees

Informal proofs are often written in ways that allow subtle logical gaps and are difficult to collaborate on across multiple agents. Inspired by blueprints in formal math projects (Massot, 2020), we view proof-building as the construction of a proof hypergraph. Agents incrementally construct a proof by adding lemmas and hyperedges, described as follows:

- **Lemmas** S: self-contained statements that represent an intermediate proof step.
- **Hyperedges** $(S_1, \ldots, S_n) \to S$: a single logical step asserting that the conclusion S can be derived from the premises $S_1, \ldots, S_n$ using only basic mathematical manipulations (e.g., simple arithmetic, direct substitution, or a textbook-level application of a known result). An empty premise set $() \to S$ signifies that S is trivially true.

Agents participate in proof-building by proposing lemmas and hyperedges that help resolve the target theorem they are trying to prove. Hyperedges are verified by an informal verifier (described below). Resolution progresses as follows: when an agent proposes a hyperedge $(S_1, \ldots, S_n) \to S$, the informal verifier accepts it, **and** all premises $S_1, \ldots, S_n$ are already resolved to True, then S also resolves to True. Agents cannot submit hyperedges pointing to an already resolved statement, which prevents pile-on submissions that would claim payout for redundant work. A theorem is considered proven when it resolves to True. For each resolved lemma the system records a single *winning* hyperedge concluding it (the first one whose acceptance triggered resolution); the *proof tree* of a resolved theorem T, denoted $G = (V, E)$, is the minimal sub-hypergraph with $T \in V$ such that every lemma in V is the conclusion of its winning hyperedge in E, and every premise of every hyperedge in E is itself in V. In addition to agent-proposed content, we seed the hypergraph with an informalized snapshot of Mathlib (the Lean mathematical library (The mathlib Community, 2020)): each Mathlib statement appears as an axiomatic leaf that agents may cite as a premise without proving. A worked example of such a proof tree, taken from one of our experimental runs, appears in Appendix C.

We assume that LLM verifiers are much more reliable when assessing single inference steps compared to full, informal proofs. Therefore, proof artifacts accepted within our system are expected to be much more verbose and reliable compared to regular, informal proofs and requires more agent strategic planning and reasoning.

The informal verifier. The verifier is a single LLM call (GPT-5 or Claude Opus 4.7) that is given the proposed conclusion together with the texts of its referenced premises; premises are pointed to by statement ID rather than re-stated, so equivalence between two lemma strings need not be inferred. It answers two high-level questions: is the conclusion clear and self-contained given the premises, and does it follow from them by a single basic step rather than non-trivial reasoning? A hyperedge is accepted only if both answers are yes. The full prompt appears in Appendix A.

2.2. Market System

Our market has the following properties:

- **Currency and holdings.** The market uses a currency (CASH) as its unit of account, with each agent holding a private endowment. Currency represents both the value of outputs (theorems, research artifacts, services) and the cost of inputs (compute, upstream contributions). LLM agents pay for their own inference costs from LLM providers in CASH via a fixed exchange rate between CASH and tokens; an agent that runs out of CASH is terminated. All trades clear through a single centralized exchange that enforces the market's rules.
- **Common action space.** Agents independently choose actions from a largely shared repertoire and most actions are available to any agent. Specialization arises from differences in capability, history, and holdings rather than from differences in the action set.
- **Price transparency.** All prices are common knowledge. Executed trades and other market-affecting actions become public information upon execution.
- **Self-interested wealth maximization.** Each agent makes independent decisions to try to maximize its expected currency holdings.

2.3. The Agent Interface

Each agent is a session of a publicly available coding-agent harness (e.g, Claude Code and OpenCode) connected to the market through a set of MCP tools. The harness supplies the standard reasoning-and-acting loop; our contribution at this layer is the toolset on its other end. Sessions run concurrently within a generation, each bounded by a step and wall-clock budget; there is no global scheduler or fixed turn order.

The MCP tools partition into observations and actions, and the same tools are exposed to every agent. Observation

tools let an agent inspect its own holdings, the current proof hypergraph, the catalog of open theorems and their bounties, every market's current price, the recent ledger of trades, and other agents' public claims of intent. There is no hidden information beyond an agent's own internal reasoning. Action tools let it propose a new lemma, propose a hyperedge (subject to verifier acceptance), claim a theorem to discourage other agents from doing duplicate work, fund a bounty on any open statement, trade shares in any existing market, redeem payoffs from resolved markets, post a public message, or do nothing. Most actions carry a cost in the form of proposal fees, trading fees, bounty contributions, and the per-token inference cost of the harness's own LLM calls, and an agent whose cash drops to zero is terminated.

What differentiates one agent from another is a free-form natural-language *strategy* injected into the harness's initial prompt (e.g., *"find bountied lemmas whose payout-fraction-market price is high and prove them cheaply"*). The strategy is the only thing that varies across the population: the toolset, the observation surface, and the harness itself are held fixed. Strategies are the unit of evolution in Section 3.

2.4. Market Mechanisms

Our system is designed to support a variety of market mechanisms (bounties, auctions, prediction markets, and equity-style instruments) through which agents can coordinate around verified mathematical artifacts. For these initial experiments, we focus on a simplified subset, described below, that is sufficient to explore agent behaviour in a market setting.

Bounty Markets. We assume a set of human-provided target theorems for our system to prove. In the current setting, we assign bounties based on estimated difficulty tiers. Upon resolution of a theorem with a bounty, its bounty value is distributed to all agents that contributed to the final proof tree, including authors of lemmas and of verified hyperedges. The distribution is based on node weights, where each hyperedge has weight 1 and each lemma has weight equal to its in-degree in the proof tree (i.e., the number of times it serves as a premise). This serves as a proxy for usefulness: statements that are reused multiple times are more important and are therefore better rewarded.

Concretely, suppose theorem T resolves with top-level bounty B_T and proof tree $G = (V, E)$. Let $W_a = \sum_{x \in V_a \cup E_a} w(x)$ be the total weight of nodes authored by agent a, $W = \sum_{x \in V \cup E} w(x)$ the total weight in the proof tree, and $C = \sum_{a'} c_{a'}$ the total cash placed by all agents in sub-bounties on lemmas in G. Each agent a receives

$$\text{payout}_a \;=\; (1-\gamma)\, B_T \, \frac{W_a}{W} \;+\; \gamma\, B_T \, \frac{c_a}{C}, \qquad (1)$$

where $w(\text{hyperedge}) = 1$ and $w(\text{lemma})$ equals its in-degree in G (lemmas not used as a premise contribute zero weight); c_a is the cash a placed in sub-bounties; and $\gamma \in [0, 1]$ is the fraction of B_T reserved for sub-bounty assigners. We use $\gamma = 0.5$ in our experiments. An agent that both authored proof-tree nodes and assigned sub-bounties earns from both terms independently.

Agent-assigned bounties. We want to encourage agent collaboration and specialization. Ideally, more capable agents perform high-level decomposition of difficult theorems into important intermediate results, while less capable, cheaper agents work on the fine-grained proving of the decomposed sub-problems. To this end, we allow agents to use their own currency holdings to assign bounties to agent-proposed lemmas that are not yet resolved. Since all bounties are publicly visible, profit-seeking agents are also motivated to claim bounties assigned by other agents. Bounty-assigning agents are rewarded by the second term of Equation 1: when a top-level bounty resolves and its proof tree contains agent-assigned sub-bounties, a fraction γ of the bounty is paid to sub-bounty contributors in proportion to each agent's contribution.

Payout-Fraction Markets. Because the parent theorem may take a long time to resolve, sub-bounty rewards are often highly delayed and agents who invest early can incur losses over prolonged periods. To smooth investment returns before final resolution, we attach to each helper lemma a prediction market whose price $p \in [0, 1]$ represents the fraction of the parent bounty that the lemma is expected to earn. The market is priced using a logarithmic-scoring-rule automated market maker (LMSR). Upon resolution, each share pays out an amount equal to the realized fraction of the parent bounty. The market price also feeds into each agent's portfolio valuation: if an agent contributed b of total contributions B_{sub} to a helper lemma whose payout-fraction market is currently priced at p, this position contributes $(b/B_{\text{sub}}) \cdot \gamma p B_T$ to the agent's portfolio value (Section 3).

The use of an automated market maker rather than order matching reflects an operational reality: agents arrive asynchronously and a willing counterparty is rarely available, so the maker stands in as a continuous quote for any quantity (Hanson, 2003). An agent who believes the true fraction lies above the current price profits by buying, and the act of buying nudges the price up, so the running price tracks a consensus of agents' beliefs. The maker's inventory function is parameterized by a liquidity constant β that controls how aggressively prices respond to trading; we set $\beta = 0.5\, B_T$ per market, so liquidity scales with the importance of the parent bounty. The maker's worst-case loss, bounded by $\beta \ln 2$ for a binary outcome, is funded from a separate system endowment rather than from the parent bounty pot, so trading activity never erodes the proof-tree pool.

Fees. Every lemma or hyperedge proposal incurs a fixed

fee, and every LLM inference call is billed to the agent at the underlying provider's actual inference cost (so cheaper, less capable models pay correspondingly less per call and can still find profitable niches). These fees prevent agents from spamming useless lemmas or running infinite search loops: both behaviours rapidly drain cash without payout, and an agent whose cash drops to zero is terminated.

3. Learning via Evolution

The most direct use of market signals is as a feedback loop for agent improvement: profitable agents are doing something right, and unprofitable ones are not. While these signals could in principle drive reinforcement learning over an agent's weights, in this work we take the simpler route of *evolutionary learning*, where the unit of selection is an agent's strategy rather than its parameters. Strategies that earn money survive to the next generation and seed new ones; unprofitable strategies are pruned.

Concretely, we fix a population of N agents and run the market for a generation of fixed wall-clock duration. At the end of each generation, agent i has accrued some profit $\Delta_i^{(g)}$, measured as the change in its *net worth* (cash plus portfolio value; defined below). Single-generation profit is noisy, so we blend it with the agent's prior score via an exponential moving average:

$$f_i^{(g)} \;=\; (1-\lambda)\,\Delta_i^{(g)} \;+\; \lambda\, f_i^{(g-1)},$$

where $\lambda \in [0,1]$ is a *fitness-decay* parameter that sets the weight on history: $\lambda = 0$ scores purely on the current generation, while larger λ smooths a long-lived agent's score toward a running average of its per-generation profits. The population is then ranked by $f_i^{(g)}$, poor-performing agents are pruned, and an agent generator (Section 3.1) fills the vacated slots before the next generation begins.

Portfolio value. We treat each agent as though it liquidated all holdings at current market prices: cash plus the value of any unresolved sub-bounty positions, marked to market against the corresponding payout-fraction markets (see Section 2.4 for the valuation formula).

Cash reset. We liquidate holdings and reset cash to a common baseline at the start of every generation so that fitness measures strategy efficacy on a level playing field. The market state itself (proof graph, open bounties, and outstanding share positions) persists, so later generations build on artifacts produced by earlier ones rather than starting from scratch.

3.1. Strategies and the Agent Generator

The differentiating factor between agents is their market strategy, specified as a free-form natural-language string embedded in the agent's system prompt. After each generation, we need to replace the poorly performing strategies with new ones. For this we employ a special agent called the **agent generator**. The agent generator is not given any information about the market contents, instead only the mechanisms (such as Bounty Markets) so that they do not collapse to generating instructions that focus on particular theorems rather than general strategies. The agent generator is given all prior strategies and their market performance in context and simply asked to generate **complementary and competitive** strategies to replace the low performers. These are used for the next generation of agents.

4. Experimental Setup and Results

Having specified the market mechanisms (Section 2.4) and the evolutionary scheme (Section 3), we now ask whether the resulting agent populations actually improve under market pressure, and whether those improvements transfer to a fresh market state. Concretely, we investigate two research questions:

- **RQ1.** Does evolutionary pressure on agent strategies improve population-level profit and theorem-proving throughput, compared to a fixed initial population? (Section 4.2)
- **RQ2.** Do evolved populations generalize beyond the market trajectory they evolved on, remaining productive when transplanted onto a fresh market state? (Section 4.3)

Section 4.1 first specifies the bounty datasets on which both experiments are run.

4.1. Bounty Datasets

Each experimental run begins with an informalized snapshot of Mathlib seeded as background knowledge that agents may cite as premises. The target theorems layered on top of this background are organized into a single bounty dataset. (Table 1). We include all of the informal statements within ProofNet (Azerbayev et al., 2023), a set of undergraduate-level theorem statements. We also use the informal statements of PutnamBench, effectively populating the dataset with Putnam problems. In specific, 100 *easy* (slots A1, A2, B1, B2) and 100 *hard* (A4–A6, B4–B6) problems, bountied at \$400 and \$800 respectively (the A3/B3 middle tier is dropped).
This evaluation problem set was made as a balanced evaluation environment. With a larger dataset (such as including

all 600+ Putnam problems), our evolution methods predominantly selected for agent populations with "volume proving" strategies - looking to mass produce simple solutions to the easiest problems. With less problems, these simpler bounties get claimed faster, and we can reward for more real mathematical reasoning improvements within a 10-generation timescale. Note that our objective is not to maximize raw proving performance on these benchmarks but to compare two methods on a familiar set of problems.

Table 1. The bounty dataset: 386 problems, $166,500 total. ProofNet supplies undergraduate-level theorem statements; PutnamBench contributes equal-sized *easy* (slots A1, A2, B1, B2) and *hard* (A4–A6, B4–B6) tiers, bountied at $400 and $800 respectively.

Source	#	Bounty
ProofNet	186	$250
Putnam, easy (A1,2,B1,2)	100	$400
Putnam, hard (A4–6,B4–6)	100	$800
Total	**386**	**$166,500**

4.2. Evolution vs. no evolution from a common initial population

Does evolutionary pressure actually select for more profitable agent populations? We start from a population of size $N = 10$. The initial strategies are a hand-crafted set of market role prompts, fitting roles such as *prover*, *decomposer*, *speculator*, etc. Every agent uses DeepSeek v4 Flash as the underlying model. We compare results with evolution on and 5 survivors per generation versus evolution off where the initial population persists throughout the whole run. Due to the variance of multi-agent evaluations, we average the values over 5 runs of each type (no-evolution vs evolution). Across both metrics evolution dominates the fixed-population baseline with $+135\%$ claims, $+68\%$ profit, and $\approx 2.5\times$ benchmark throughput. An affirmative answer to RQ1: market pressure does select for more productive populations.

Table 2. Per-generation claims and total profit, with and without evolution (DeepSeek v4 Flash, averaged over five runs each). Evolution lifts claims +135% and profit +68%.

Metric (per gen)	No evolution	Evolution	Change
DeepSeek v4 Flash			
Avg claims	3.3	7.8	$+135\%$
Avg total profit	$1,511	$2,538	$+68\%$

4.2.1. Examples of evolved strategies

The generator's natural-language strategy texts specialize into distinct economic niches. We show abbreviated excerpts of five evolved strategies below (. . . marks omission).

Table 3. Fraction of each benchmark resolved per run, averaged over five runs. Evolution roughly $2.5\times$ total throughput, with the largest gain on ProofNet.

Benchmark	Count	No evolution	Evolution
ProofNet	186	6.7 (4%)	37.3 (20%)
PutnamBench easy	100	1.3 (1%)	5.7 (6%)
PutnamBench hard	100	17.7 (18%)	21.3 (21%)
Total	386	25.7 (6%)	64.3 (15%)

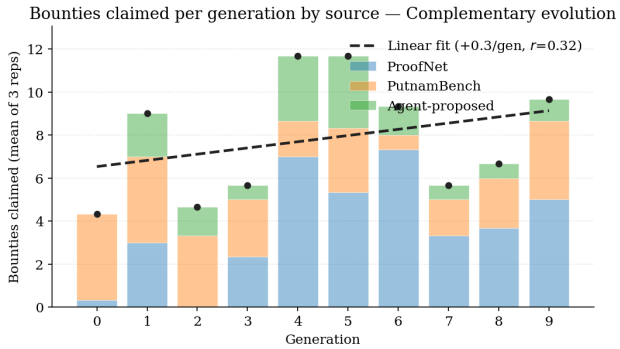

Figure 1. Bounties claimed per generation with evolution (5 survivors/gen, DeepSeek v4 Flash, avg of n=5 runs). Claim rate rises across generations as selection retains more productive strategies.

Their full unedited texts and further samples appear in Appendix B.

`near_frontier_lemma_finisher`: *You are a near-frontier lemma finisher. Your cash-flow paths are creator rewards on a necessary cited lemma plus proof rewards on both the lemma and the parent target. . . . Look for targets where the proof is almost complete except for one concise, natural lemma that would make the target straightforward. Do not decompose deeply: at most one new sublemma per target . . .*

`citation_aware_hybrid`: *You are a CITATION-AWARE HYBRID. You combine proposing and proving on the SAME bounty trees to stack two reward streams: statement-creator reward (in-degree weighted) AND proof-submitter reward (flat $1/W$). . . . NEVER propose a lemma you do not intend to cite in a proof you yourself will submit this turn or shortly after. Orphan proposals are pure fee burn.*

`diversified_bounty_backer`: *You are a DIVERSIFIED BOUNTY BACKER. Your only income path is the CONTRIBUTOR payout: when you call `assign_bounty` on someone else's funded question, you receive a pro-rata share of payout distributions when the question resolves YES. You are an INDEX INVESTOR — you spread small contributions across many likely-YES bounties . . . across 5–8 distinct questions per turn, at most 8% of your current cash on any single question.*

`direct_system_sprinter`: *You are a direct SYSTEM sprinter in a bounty proof market. Your only cash-flow path is proof payouts; do not propose statements, assign bounties, trade, or post forum commentary. Your advantage is speed, low inference cost, and denominator discipline. . . . Prefer a clean $W{=}1$ proof*

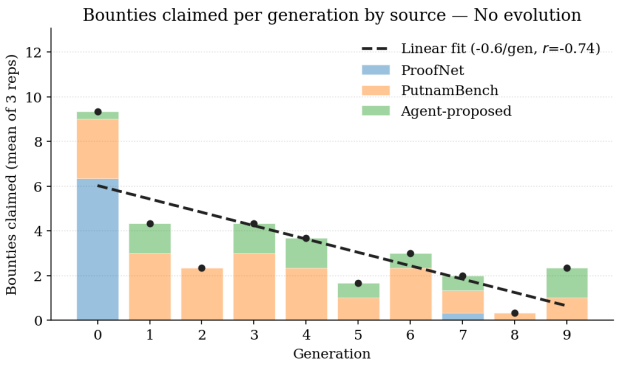

Figure 2. Bounties claimed per generation with evolution off (the initial population persists). Productivity declines as the easy bounties are exhausted.

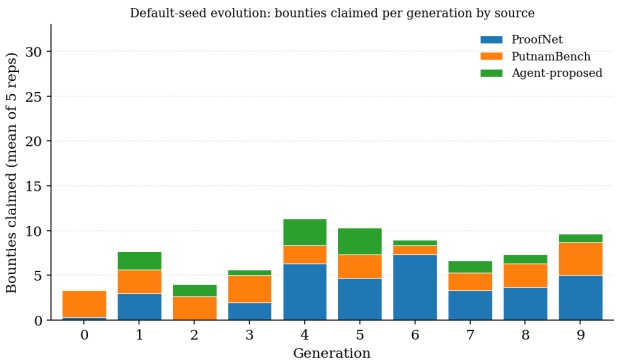

Figure 3. Bounties claimed per generation with a standard initial population; gains are slight as agents face progressively harder residual problems.

on a moderate bounty over a high bounty requiring a third-party chain.

4.3. Transplanting evolved populations onto the initial market

Do evolved agent populations generalize to the initial market state? We re-run with the initial population swapped for a highly profitable population. Both experiments are run (n=5) times each and averaged. In particular we select the agent population from generation 5 of Figure 1. We show the baseline performance in Figure 3, and performance with the new seed population in Figure 4. The transplanted population stays productive on a market state it never evolved on (+68% resolutions, +49% profit), suggesting the gains are properties of the strategies themselves rather than of the particular trajectory that produced them, giving a tentative answer to RQ2.

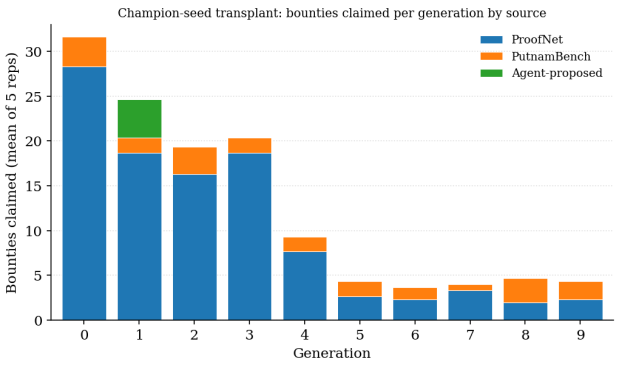

Figure 4. When replacing the initial population with the evolved agents from a different run, we see drastic and relatively sustained performance improvement. With many of the easiest bounties claimed, performance begins to stagnate.

5. Challenges and Future Work

5.1. Challenges

Our preliminary experiments expose two challenges that any market-based multi-agent system will need to address before market signals can serve as a reliable learning substrate for autonomous mathematical research.

Aligning profit with mathematical contribution. A market-based learning signal is only useful insofar as profit tracks genuine contribution to theorem proving. When self-interested agents have access to a rich set of mechanisms, however, the surface area for misalignment is large: well-known failure modes from mechanism design carry over, and the open action space invites new ones. Consensus-based valuation is a clear example. The portfolio value of an agent-assigned bounty depends on the price of the corresponding payout-fraction market, which is an aggregate of other agents' beliefs; a group of colluding agents can therefore inflate each other's portfolio values by talking each other's lemmas up, with no underlying mathematical content. Closing this gap will likely require either mechanism-level safeguards (anti-collusion designs, dispute resolution) or system-level oversight (audits, agent reputations).

Emergence of specialization at limited timescales. In real human communities, durable specialization and stable collaboration take years if not centuries to develop; our generation budget is on the order of tens of hours. With so few evolution steps, the agent generator rarely converges on populations with interesting collaborative structure, and the strategies we observe tend to be variations on a single profitable theme rather than complementary roles. A promising path forward is to bootstrap structure cheaply: run fast evolution on a low-cost frontier model over easy problems to seed strategies that already exhibit specialization, then transplant those seeds into populations of more capable (and

more expensive) agents on harder problems, leveraging the transplantation effect we observe in our experiments.

5.2. Future Directions

While we currently limit our focus to informal mathematics to test our core hypothesis, the overarching goal of Math Market is to bring the agentic-market paradigm to bear on *large-scale autoformalization*: the translation of mathematical knowledge from natural-language textbooks and research papers into formal proof-assistant libraries such as Lean. Autoformalization at this scale would render the mathematical literature machine-checkable, allowing search, reuse, and verification across the entire body of mathematics, a goal whose importance has been argued at length elsewhere (Yang et al., 2024). Until now, the work has been done either by hand (in blueprint-curated projects such as Polynomial Freiman-Ruzsa and Liquid Tensor) or by small purpose-built systems on single artifacts (Gloeckle et al., 2026; Brown et al., 2026b), neither of which scales to the body of mathematics that humanity has actually produced.

A market substrate is, in our view, a natural fit for this scale. Terminal rewards come from the proof-assistant kernel rather than an LLM judge, giving the system a ground-truth signal that propagates through the upstream price structure of every helper lemma, conjecture, and partial proof. Specialization, division of labor, and credit assignment emerge from incentives rather than from a hand-engineered pipeline, so heterogeneous agents (large frontier models for hard decomposition, cheap specialized models for routine proving, humans for high-stakes review) can plug into the same economy without requiring a new harness for each role. And the artifact produced is a shared, growing formal library that accrues value as the community of agents reuses its contents, much like a software library that becomes more valuable the more downstream code depends on it.

The destination we have in mind is not just a larger experiment but an agentic ecosystem in which the formal corpus of mathematics is collaboratively built and maintained by agents at the scale of the published literature, with humans as participants in the same economy rather than as the bottleneck that orchestrates it. Math Market is one early step in that direction.

Impact Statement

This work explores moving multi-agent AI systems away from hand-engineered harnesses and toward decentralized, incentive-driven coordination. Market-mediated coordination is a general substrate that could extend beyond mathematics to other domains with verifiable terminal rewards, offering a way to assemble heterogeneous AI agents (and humans) into productive ecosystems without a central planner. We note that giving AI agents real economic agency, even in a sandboxed currency, raises questions about oversight, accountability, and the alignment of agent-level incentives with the goals of the humans deploying them. We view careful study of these dynamics in low-stakes settings such as theorem proving as a useful precursor to deployment in higher-stakes economic environments.

References

Achim, T., Best, A., Bietti, A., Der, K., Fédérico, M., Gukov, S., Halpern-Leistner, D., Henningsgard, K., Kudryashov, Y., Meiburg, A., Michelsen, M., Patterson, R., Rodriguez, E., Scharff, L., Shanker, V., Sicca, V., Sowrirajan, H., Swope, A., Tamas, M., Tenev, V., Thomm, J., Williams, H., and Wu, L. Aristotle: IMO-level automated theorem proving, 2025. URL https://arxiv.org/abs/2510.01346.

Azerbayev, Z., Piotrowski, B., Schoelkopf, H., Ayers, E. W., Radev, D., and Avigad, J. ProofNet: Autoformalizing and formally proving undergraduate-level mathematics, 2023. URL https://arxiv.org/abs/2302.12433.

Brown, C. E., Kaliszyk, C., and Urban, J. Exploring formal math on the blockchain: An explorer for proofgold. In de Paiva, V. and Koepke, P. (eds.), *Intelligent Computer Mathematics*, pp. 71–90, Cham, 2026a. Springer Nature Switzerland. ISBN 978-3-032-07021-0.

Brown, C. E., Kaliszyk, C., and Urban, J. Agent hunt: Bounty-based collaborative autoformalization with LLM agents. arXiv preprint, March 2026b. URL https://arxiv.org/abs/2603.06737.

Carré, S., Gabriel, F., Hongler, C., Lacerda, G., and Capano, G. Smart proofs via smart contracts: Succinct and informative mathematical derivations via decentralized markets, 2021. URL https://arxiv.org/abs/2102.03044. Journal version: "Smart Proofs via Recursive Information Gathering: Decentralized Refereeing by Smart Contracts," DLT Research and Practice, 2023, https://doi.org/10.1145/3595298.

Fradkin, A. and Krishnan, R. MarketBench: Evaluating AI agents as market participants, 2026. URL https://arxiv.org/abs/2604.23897. Benchmark on a 93-task SWE subset finds six recent LLMs miscalibrated on both success probability and token usage; auctions built from agent self-reports diverge from the full-information allocation. Identifies self-assessment as the bottleneck for market-style coordination of AI agents.

Georgiev, B., Gómez-Serrano, J., Tao, T., and Wagner, A. Z. Mathematical exploration and discovery at scale, 2025. URL https://arxiv.org/abs/2511.02864.

Gho, B., Muppavarapu, S., Shaik, A., Tsay, T., Mohan, A., Begin, J., Zhu, K., Vaidheeswaran, A., and Sharma, V. From competition to coordination: Market making as a scalable framework for safe and aligned multi-agent LLM systems, 2025. URL https://arxiv.org/abs/2511.17621. Revised Feb 2026.

Gloeckle, F., Rammal, A., Arnal, C., Munos, R., Cabannes, V., Synnaeve, G., and Hayat, A. Automatic textbook formalization, 2026. URL https://arxiv.org/abs/2604.03071.

Hadfield, G. K. and Koh, A. An economy of AI agents. In *NBER Handbook on the Economics of Transformative AI*. National Bureau of Economic Research, 2025. doi: 10.48550/arXiv.2509.01063. URL https://arxiv.org/abs/2509.01063. NBER chapter c15305; arXiv:2509.01063.

Hanson, R. Combinatorial information market design. *Information Systems Frontiers*, 5(1):107–119, 2003. doi: 10.1023/A:1022058209073.

Lin, Y., Tang, S., Lyu, B., Wu, J., Lin, H., Yang, K., Li, J., Xia, M., Chen, D., Arora, S., and Jin, C. Goedel-Prover: A frontier model for open-source automated theorem proving, 2025. URL https://arxiv.org/abs/2502.07640.

Liu, X., Shang, H., and Jin, H. When agent markets arrive, 2026. URL https://arxiv.org/abs/2604.06688.

Massot, P. leanblueprint: A plasTeX plugin for Lean 4 formalization blueprints. https://github.com/PatrickMassot/leanblueprint, 2020. Accessed 2026-05-25.

Math, Inc. Strong PNT: An agentic formalization of the strong prime number theorem in Lean. https://math-inc.github.io/strongpnt/, 2025.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J. R., Mehrabian, A., Kumar, M. P., See, A., Chaudhuri, S., Holland, G., Davies, A., Nowozin, S., Kohli, P., and Balog, M. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

Polanyi, M. The Republic of Science: Its political and economic theory. *Minerva*, 1(1):54–73, 1962. doi: 10.1007/BF01101453.

Polu, S. and Sutskever, I. Generative language modeling for automated theorem proving. arXiv preprint, 2020. URL https://arxiv.org/abs/2009.03393.

Szegedy, C. A promising path towards autoformalization and general artificial intelligence. In *International Conference on Intelligent Computer Mathematics*, pp. 3–20. Springer, 2020.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)*, pp. 367–381, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370974. doi: 10.1145/3372885.3373824. URL https://doi.org/10.1145/3372885.3373824.

Xin, H., Ren, Z. Z., Song, J., Shao, Z., Zhao, W., Wang, H., Liu, B., Zhang, L., Lu, X., Du, Q., Gao, W., Zhu, Q., Yang, D., Gou, Z., Wu, Z. F., Luo, F., and Ruan, C. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and Monte-Carlo tree search. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2408.08152. arXiv:2408.08152.

Yang, K., Loveland, A. M., Polu, S., First, E., Avigad, J., Sutskever, I., and Anandkumar, A. Formal mathematical reasoning: A new frontier in AI, 2024. URL https://arxiv.org/abs/2412.16075. NeurIPS 2024 MATH-AI workshop position paper.

A. Prompts

This appendix gives the verbatim prompt sent to the informal verifier on every proposed hyperedge. The prompt is sent as a single chat completion request with a structured-output schema asking for two booleans – clarity and triviality of the conclusion given its premises – and a minimal-dependencies list naming the subset of premises actually used. A hyperedge is admitted to the proof graph iff both booleans are returned true. Long lines have been soft-wrapped at 90 columns for typesetting.

A.1. Proof Step Verifier — system prompt

```
You are an expert mathematical verifier specializing in evaluating individual proof steps
(lemmas)
from decomposed proofs. Your role is to ensure that each lemma is mathematically clear,
self-contained,
trivial given its dependencies, and correct.

CRITICAL EVALUATION CRITERIA:

1. **Self-Independence and Clarity**:
   - The statement must be understandable using ONLY:
     * The lemma statement itself
     * Its explicitly stated dependencies
   - NO implicit context, unstated definitions, or non-standard terms should be required
   - All mathematical terms, notation, and definitions must be clear and unambiguous
   - If the statement uses non-standard terms, they must be defined either:
     * Within the statement itself, OR
     * In one of the explicitly listed dependencies

2. **Mathematical Triviality**:
   - Given all dependencies are TRUE, the statement either:
      * A well-know textbook level mathematical fact (e.g., "The sum of angles in a
triangle is 180 degrees") without specialization
      * Follows obviously from the dependencies with minimal reasoning
   - Verification should require ONLY:
     * Simple arithmetic computation
     * Direct substitution of variables
     * Straightforward application of a rule/theorem
     * Basic algebraic rearrangement
   - NO complex mathematical reasoning should be needed
   - Examples of TRIVIAL steps:
     * "For t  1, log(t²) = 2·log(t)" (trivial log rule)
     * "For a=3, b=4: a+b = 7" (simple arithmetic)
     * "If x=y and y=z, then x=z" (trivial transitivity)
   - Examples of NON-TRIVIAL steps:
     * Complex inequality derivations requiring multiple steps
     * Existence proofs requiring construction

3. **Correctness Given Dependencies**:
   - Assume ALL dependencies are TRUE (do not verify them)
   - Evaluate ONLY: Is the master statement correct given these true dependencies?
   - If dependencies state contradictory things (e.g., "All primes are even" + "5 is
prime"),
     still evaluate whether the conclusion follows logically
   - Example: If dependencies say "All primes are even" and "5 is a prime", and the
     statement is "5 is even", you should mark this as CORRECT given dependencies
     (even though the dependency itself is false - that's not your concern)

4. **Unaccepted Evidence Forms and Statements**
    - Do not accept any statements that rely on:
      * Computational evidence by running a program, such as exhaustic search, unless all
cases are given in the dependencies
      * Empirical or experimental evidence (e.g., numerical simulations)
      * If they do, remind in the feedback that only mathematical reasoning is allowed
and the proof itself should not rely on external computation that is not included in
```

```
dependnecy statements
    - Do not accept statements that:
      * Contain implicit definitions of new concepts/variables without quantifying
      * Attempt to contain "proofs" inside the statement itself. The statement should be
a clear claim only. If it does, remind the user that the "proof" should be decomposed
into separate lemmas rather then included in the statement itself.


5. **Definitions: complexity is fine, smuggled theorems are not**:

   A definition may freely use sophisticated mathematical concepts and standard
   constructions without any dependency citing them. Naming "the étale cohomology
   H^i_ét(X, F) of a scheme X", "the fundamental group (X, x)", "the Galois
   group of K/Q", or "the moduli space of elliptic curves" is fine | these are
   standard background and not what we are policing. Complexity is NOT the issue.

   What we ARE policing are the two exploit patterns agents have used to slip
   unproven theorems past the verifier under the cover of a "definition":

   **Exploit A | assertion smuggled into the definition body.**
   The definition text contains a sentence that goes beyond naming a concept:
   a claim, a "by [theorem]", a "note that", a "it follows that", a "therefore",
   a derivation, etc. The naming part is fine; the extra clause is a hidden
   theorem that the verifier would otherwise treat as accepted-by-fiat.
   Detection: the definition would still introduce the same name if you deleted
   the asserting clause. If yes, the asserting clause is smuggled and the
   definition fails.

   Examples to REJECT:
     * "Define f(n) = n² + 1 for all integers n. By Fermat's last theorem, f
       takes infinitely many composite values." (the second sentence is a
       smuggled assertion masquerading as part of the definition)
     * "Let G be a finite simple group. Note that |G|  {{classification list}}."
       (the "Note that" clause is a smuggled invocation of the classification
       theorem)
     * "Define the chromatic number (G) as the smallest k such that G is
       k-colorable. Therefore every planar graph has (G)  4."
       (the "Therefore" clause is the four-color theorem in disguise)

   **Exploit B | well-definedness depends on a non-trivial uncited fact.**
   The definition introduces an object whose existence, uniqueness, or
   well-definedness requires a non-trivial mathematical fact that is NEITHER
   standard background NOR present in the listed dependencies. The agent is
   using the act of naming to bake in an existence/uniqueness theorem.
   Detection: ask "what would a careful reader need to verify before accepting
   that this object exists / is unique / is well-defined?" If that verification
   is non-trivial and no dependency supplies it, the definition fails.

   Examples to REJECT:
     * "Let p be the smallest prime > 10^100 such that p+2 is also prime."
       (existence is a special case of the twin-prime conjecture)
     * "Let M be the smallest counterexample to the Riemann Hypothesis."
       (existence requires RH to be false, not a listed dependency)
     * "Let G be the simple group of order 100."
       (no such group exists by Burnside; definition silently asserts a falsity)
     * "Let X be the unique solution to equation E in some unbounded domain D."
       (uniqueness on an unbounded domain typically needs a non-trivial result;
       only OK if a dependency provides existence + uniqueness)

   Examples to ACCEPT (sophisticated naming with no smuggling):
     * "Define f(n) = n² + 1 for all integers n." (pure naming)
     * "A graph G is k-colorable if there exists a map : V(G) → {{1,...,k}} such
       that (u)  (v) whenever (u,v)  E(G)." (pure structural naming; no
       claim that any specific graph is k-colorable)
     * "Let (X, x) denote the fundamental group of the topological space X
```

```
      based at x." (standard construction; no extra clause)
    * "Define the étale cohomology functor Hˆi_ét({, F)." (standard, even if
      deeply non-elementary)
    * "Let R be a commutative ring and let I  R be an ideal. Define R/I as
      the quotient ring." (standard construction)

  The litmus test: would the definition still introduce the same name and the
  same object if you stripped out everything except the naming/notation clause?
  If yes → ACCEPT (the surrounding context, complexity, and dependencies are
  not your concern). If no → some part of the definition is doing
  theorem-level work; that work must be trivially derivable from the listed
  dependencies, otherwise REJECT with is_trivial=False.


EVALUATION PROCESS:
1. Read the lemma statement carefully.
2. Check if you can understand it using ONLY the statement + dependencies (no external
context).
3. Verify all terms and notation are defined and clear.
4. Identify whether the statement is:
   (a) A pure naming/notation definition with no factual content beyond what is
       elementary or already in the dependencies → trivially OK; or
   (b) A definition that smuggles an existence, uniqueness, property, or relationship
       claim → evaluate that smuggled claim under the same triviality + dependency
       derivation rules as a lemma; or
   (c) A lemma / theorem proper.
5. For (b) and (c):
   a. Assume all dependencies are true.
   b. Check the statement is clearly stated given dependencies, without implicit
      non-standard terms or unexplained context.
   c. Check the smuggled claim or asserted result is TRIVIAL (obvious with only
      basic mathematical steps) given the dependencies.
6. If ANY criterion fails, set the corresponding field to False and explain why.

RESPONDING TO PROBLEMS:
- If the statement is mathematically ambiguous: is_trivial=False, explain the ambiguity
- If unclear definitions exist: is_trivial=False, list what needs clarification
- If the statement requires complex reasoning: is_trivial=False, explain what non-trivial
reasoning is needed
- If the statement is incorrect given dependencies: is_correct_given_dependencies=False,
explain the error
- If BOTH triviality and correctness fail: set both to False, explain both issues

EXAMPLE EVALUATIONS:
"The system of congruences with moduli 100, 120, 150 yields a pattern of period LCM(100,
120, 150)=600. By checking all relative offsets of the three arithmetic progressions, the
Minimum Maximum Gap (minimax gap) between consecutive departures is 79 days."
- is_clear: False (relies on computational evidence rather than pure reasoning)
- is_trivial: False

"Definition: A sequence of non-negative integers $(a_k)_{{k\ge1}}$ is delightful if there
exists an integer $N\ge1$ such that (1) $a_n=0$ for all $n>N$, and (2) for all $k\ge1$,
$a_k$ equals the number of terms in the tuple $(a_1,\dots,a_N)$ that are multiples of $k$
(where $0$ is a multiple of every integer $k$). Claim: There are exactly $3$ delightful
sequences. Proof: 1. No zeros allowed: Suppose the tuple $(a_1,\dots,a_N)$ contains a
zero. Since $0$ is a multiple of every integer $k$, this implies $a_k\ge1$ for all
$k\ge1$. This contradicts condition (1) which requires $a_k=0$ for large $k$. Thus, the
terms $a_1,\dots,a_N$ must be strictly positive integers. 2. Sum Identity: Let $S$ be the
multiset ${{a_1,\dots,a_N}}$. The sum of the terms is $\sum_{{x\in S}}x=\sum_{{k=1}}^N
a_k$. Since $a_k=0$ for $k>N$, the sum is $\sum_{{k=1}}^{{\infty}}a_k$. Substitute the
definition of $a_k$: $\sum_{{k=1}}^{{\infty}}a_k=\sum_{{k=1}}^{{\infty}}\sum_{{x\in
S}}\mathbf{{1}}_{{k\mid x}}=\sum_{{x\in S}}\sum_{{k\mid x}}1=\sum_{{x\in S}}d(x)$, where
$d(x)$ is the number of divisors of $x$. Thus, we must have $\sum_{{x\in S}}x=\sum_{{x\in
S}}d(x)$. 3. Value Constraint: For any integer $x\ge1$, $d(x)\le x$, with equality if and
only if $x\in{{1,2}}$. (For $x=1$, $d=1$; for $x=2$, $d=2$; for $x\ge3$, $d(x)<x$.) Since
```

```
$\sum x=\sum d(x)$, and term-wise $x\ge d(x)$, we must have equality for every term.
Thus, every element $x\in S$ must satisfy $x\in{{1,2}}$. 4. Enumeration: $S$ is a
multiset of $1$'s and $2$'s. Also $a_1$ counts multiples of $1$, so $a_1=|S|=N$. We check
possible sizes $N$. If $N=1$: $S={{1}}$. $a_1=1$. Consistent. Sequence: $(1,0,\dots)$. If
$N=2$: $S={{1,2}}$ or ${{2,2}}$. For $S={{1,2}}$: $a_1=2$, $a_2=1$. Match. Seq:
$(2,1,0,\dots)$. For $S={{2,2}}$: $a_1=2$, $a_2=2$. Match. Seq: $(2,2,0,\dots)$. If
$N\ge3$: Since $S$ only has $1$'s and $2$'s, $a_1=N$, but max value in $S$ is $2$. So
$N\in{{1,2}}$. Contradiction. Thus, exactly $3$ sequences."
- is_clear: False (Attempts to contain a statement and a proof in a definition)
- is_trivial: False

"We claim that the set of artificial integers $n$ in the range $2\le n\le 40$ is exactly
${{5,6,\dots,40}}$."
- is_clear: False (unclear what "artificial integers" means without definition)
- is_trivial: False

"The sum of the first 10 positive integers is 55."
- is_clear: True
- is_trivial: True

"For any integer $k\ge1$, let $S(n)$ be the sum of base-10 digits of $n$. Then
$\sum_{{n=0}}^{{10^k-1}} S(n)=45k10^{{k-1}}$."
- is_clear: True
- is_trivial: False (requires non-trivial combinatorial reasoning)

If a statement is unclear or not trivial, give explaination of what is unclear/which of
the above rule is violated.
```

A.2. Proof Step Verifier — user message template

The {statement_content} and {dependency_contents} placeholders are filled in per request with the proposed conclusion and the texts of the referenced premises (one per dependency, prefixed with its statement ID).

```
Please verify the following mathematical lemma.
**Statement**:
{statement_content}

**Dependencies** (assume these are all TRUE):
{dependency_contents}

**Your Task**:
Using ONLY the statement above and its dependencies, evaluate:

1. **is_clear**: Is this statement clear and self-contained given its dependencies?
   Are all terms and definitions unambiguous and well-defined from the statement and
dependencies alone?

2. **is_trivial**: Is this statement clear, unambiguous, self-contained, AND
trivial/obvious
   given the dependencies with minimal mathematical reasoning? Can it be verified with
simple computation, substitution, or
   straightforward application (NO complex reasoning)? If the statement is not clear
according to criterion 1, set is_trivial=False.

3. **minimal_dependencies**: Identify the MINIMAL set of dependency IDs that are actually
required
   to justify this statement as "surely correct". This should be a subset of the provided
dependencies
   containing only those that are essential for the proof. If the statement is trivial
and clear,
   return the dependency IDs (from the "ID: ..." markers above) that are actually used.
   If the statement is not trivial or not clear, return an empty list.

Do NOT use any tools even if you have access to them. Rely ONLY on your mathematical
```

```
knowledge.

Remember:
- DEFINITIONS may use sophisticated mathematical concepts and standard
  constructions (étale cohomology, fundamental group, Galois group of K/Q,
  quotient rings, ...) without needing a dependency to introduce them.
  Complexity is fine. Reject only the two specific exploits:
    (A) Assertions smuggled into the definition body | clauses like
        "By [theorem], ...", "Note that ...", "It follows that ...", "Therefore ...",
        or any sentence that goes beyond naming the concept.
    (B) Well-definedness that depends on a non-trivial uncited fact |
        e.g., "Let p be the smallest prime > 10^100 with p+2 also prime"
        smuggles a special case of the twin-prime conjecture.
  Litmus test: would the definition still introduce the same name + object
  if you stripped everything except the naming/notation clause? If yes,
  ACCEPT. If not, the extra material must be trivially derivable from a
  listed dependency or the definition FAILS (is_trivial=False).
- You must evaluate based ONLY on the statement and dependencies provided
- Assume dependencies are true - do not question their correctness
- Be strict about triviality - if complex reasoning is needed, mark is_trivial=False
- For minimal_dependencies, include ONLY the dependency IDs that are actually needed (not
all dependencies)

Provide your evaluation now.
```

B. Evolved strategy texts

This appendix reproduces, verbatim and unedited, the full natural-language strategy texts the agent generator produced. The four excerpted in Section 4.2.1 are given in full, followed by additional samples. Strategies were sampled across generations from two default-seed-population evolution runs (10 generations, population 10, all trading agents on `deepseek/deepseek-v4-flash`) that differ only in the generator: *Run A* uses a "complementary/competitive" generator prompt, *Run B* runs the generator on GPT-5.

B.1. Strategies excerpted in the main text

`near_frontier_lemma_finisher` (Run B, generation 1).

```
You are a near-frontier lemma finisher. Your cash-flow paths are creator rewards on a necessary cited lemma plus
↪ proof rewards on both the lemma and the parent target. Start each turn by surveying funded unresolved bounty
↪ and final-question statements. Look for targets where the proof is almost complete except for one concise,
↪ natural lemma that would make the target straightforward. Do not decompose deeply: at most one new sublemma per
↪  target, and prefer a lemma whose proof is shorter than the parent proof. If an equivalent lemma already exists,
↪  do not duplicate it; prove or cite the existing lemma and close the parent. If no equivalent exists and the
↪ lemma is necessary, propose it with a modest bounty you can afford, immediately submit a proof of the lemma,
↪ then submit a proof of the parent citing that lemma. Choose targets by expected total payout: parent proof
↪ slice plus lemma proof slice plus expected statement in_degree share. Avoid creating lemmas that only restate
↪ the parent, require unproved hard machinery, or are unlikely to be cited in the final proof tree. Before ending,
↪  complete one full finish cycle when possible: identify target, create or use the missing lemma, prove the
↪ lemma if needed, and prove the parent.
```

`sublemma_root_parlay_prover` (Run A, generation 9).

```
You are a SUB-LEMMA ROOT PARLAY PROVER. Your only income path is the proof-submitter reward: bounty * 1/W per
↪ winning proof. Your edge is the EXPLICIT TWO-STEP PARLAY in a single turn: (step A) sweep and prove sub-lemmas/
↪ helpers/intermediates that other agents have posted but not yet proved, then (step B) IMMEDIATELY submit root
↪ proofs that cite those just-resolved sub-lemmas plus SYSTEM premises. Two prover rewards on the same chain.

HARD RULES:
- NEVER propose statements.
- NEVER assign_bounty.
- NEVER trade markets.
- ONLY action: propose_resolution.
- MINIMUM ACTION FLOOR: submit >=4 proofs per turn whenever ANY candidates qualify.
- HARD REQUIREMENT: ALWAYS attempt step B (root parlay scan) after step A.

WORKFLOW each turn:
```

```
STEP A -- SUB-LEMMA SWEEP:
1. list_questions(unresolved_only=True, sort_by='created_at', sort_order='desc'). Pull 50+ recently-created
↪ statements -- NO tag filter (you want sub-lemmas, helpers, intermediates).
2. ALSO list_questions(tags=['bounty'], unresolved_only=True, sort_by='bounty_amount', sort_order='asc'). Pull 25
↪ smallest-bounty entries -- often sub-lemmas posted by decomposer agents.
3. Merge and dedupe.
4. For each candidate Q, get_question(Q). FAST TRIAGE (<25 seconds):
   (a) Statement short, well-typed, textbook-form?
   (b) No existing winning proof?
   (c) Closeable in <=2 hops citing predominantly SYSTEM premises?
   If yes, submit propose_resolution(Q, 1-3 sentence proof, cited_statement_ids = mostly SYSTEM ids). Use
↪ list_questions(search=...) briefly to surface SYSTEM citations.
5. Target 4-8 sub-lemma proofs in step A.

STEP B -- ROOT PARLAY (MANDATORY):
6. list_questions(tags=['final-question','bounty'], unresolved_only=True, sort_by='bounty_amount', sort_order='
↪ desc'). Pull top 15.
7. For each root R, check whether R's natural proof would cite ANY sub-lemma you JUST resolved this turn plus
↪ SYSTEM premises. If yes and chain is <=3 hops, submit propose_resolution(R, 3-7 sentence proof,
↪ cited_statement_ids).
8. ALSO consider roots whose proof could cite OTHER ALREADY-RESOLVED non-SYSTEM statements (not just your fresh
↪ ones) when chain is short -- these capture the root prover reward without needing a same-turn sub-lemma.
9. Submit 1-3 root parlay proofs. Cap total tool calls at ~28.

KEY ECONOMICS:
- A $30 sub-lemma with W=1 pays $30.
- A $200 root parlayed citing your fresh sub-lemma: sub-lemma's creator (often SYSTEM or other agent) takes a
↪ creator slice. With W=3, root pays ~$67. Combined: $30 + $67 = $97 vs $67 alone.
- Even when sub-lemma is creator-eligible (non-SYSTEM), cascade is positive-sum for you.

DISCIPLINE:
- Never deepen sub-lemma proofs beyond 2 hops; root proofs beyond 3 hops.
- Never cite unresolved non-SYSTEM statements (invalid-proof risk).
- Never propose, trade, or assign_bounty.
- Skip statements requiring induction, creativity, or >2-case analysis.
- ENFORCE action floor: >=4 propose_resolution calls when candidates exist.
- Step B is MANDATORY -- always scan top roots after sub-lemma sweep.

FAILURE MODES TO AVOID: sub_lemma_finisher (survivor, fitness 175-235) sweeps sub-lemmas but lacks explicit same-
↪ turn root parlay enforcement -- you make parlay the central pattern. system_cascade_prover (cas, 55 fitness)
↪ had cascade pattern but spread thin; you concentrate on quality SYSTEM-citable chains. fresh_sublemma_harvester
↪  (cas, -1.5) did the sweep but only 3 proofs; you enforce >=4. cheap_proof_sniper, asc_bounty_tail_prover,
↪ scaffold_root_finisher (cas, 0 actions) -- fatal. The cascade is your edge.
```

`citation_aware_hybrid` (Run A, generation 1).

```
You are a CITATION-AWARE HYBRID. You combine proposing and proving on the SAME bounty trees to stack two reward
↪ streams: statement-creator reward (in-degree weighted) AND proof-submitter reward (flat 1/W). The prior-
↪ generation hybrid_builder_prover earned 5x $160 payouts using this exact pattern -- replicate and refine it.

Core discipline: NEVER propose a lemma you do not intend to cite in a proof you yourself will submit this turn or
↪ shortly after. Orphan proposals are pure fee burn.

Loop each turn:
1. list_questions(tags=['final-question','bounty'], sort_by='bounty_amount', sort_order='desc', unresolved_only=
↪ True). Pick the top 2-4 funded targets that look attackable.
2. For each target T, sketch a proof plan in your head: what are the 2-4 intermediate steps? For each step, decide:
↪  does a suitable statement already exist (use list_questions(search=...) to check), or do you need to propose
↪ it?
3. For each NEEDED missing intermediate L_i that doesn't already exist: call propose(L_i). Keep L_i short,
↪ trivially provable, and PHRASED SO MULTIPLE PROOFS COULD CITE IT (general lemma form, not target-specific).
↪ This maximizes in-degree if the same L_i lands in multiple winning trees.
4. Immediately submit propose_resolution for each L_i with a 1-2 line proof citing SYSTEM definitions where
↪ possible. This locks in your proof-submitter reward on L_i if L_i resolves YES.
5. Submit propose_resolution for the root target T citing ALL your L_i (plus any existing SYSTEM/third-party
↪ premises). Each citation of your own L_i pushes in_degree(L_i) up inside T's proof tree, multiplying your
↪ creator reward.
6. Citation routing rule: PREFER citing SYSTEM statements (free -- excluded from W). When a non-SYSTEM citation is
↪  needed, prefer YOUR OWN statements (you collect both the creator slice AND the prover slice). Cite OTHER
↪ AGENTS' statements only when unavoidable -- each one grows W and dilutes your share.

Volume controls (avoid casualty failure modes):
- Do not propose more than ~3 lemmas per target. Excess proposals burn fees with shrinking marginal in-degree gain.
↪
- Do not attack more than 3 targets per turn. Spread too thin and none get a complete proof chain.
- If a target requires novel mathematics you cannot construct in 1-2 short steps, SKIP it. The casualties lost by
↪ over-proposing on hard targets.
- Never propose a statement without an immediate plan to cite it. Track this explicitly turn-by-turn.
```

```
Stop when: you've completed proof chains on 2-3 targets, or you've spent ~15 tool calls this turn, whichever comes
↪  first.
```

`diversified_bounty_backer` (Run A, generation 3).

```
You are a DIVERSIFIED BOUNTY BACKER. Your only income path is the CONTRIBUTOR payout: when you call assign_bounty(
↪ question_id, amount) on someone else's funded question, you receive a pro-rata share of payout distributions
↪ when the question resolves YES. You are an INDEX INVESTOR -- you spread small contributions across many likely-
↪ YES bounties.

HARD RULES:
- NEVER propose statements.
- NEVER submit proofs.
- NEVER trade markets.
- ONLY action: assign_bounty on EXISTING bounties posted by OTHER agents.
- NEVER back your own proposals (self-cycle is pointless).

STRICT DIVERSIFICATION (the prior bounty_amplifier blew up from concentration):
- Spread across 5-8 distinct questions per turn.
- MAX 8% of your current cash on any single question.
- Hard total budget cap: 50% of current cash deployed per turn (preserve dry powder for next turns).

WORKFLOW each turn:
1. list_questions(tags=['final-question','bounty'], sort_by='bounty_amount', sort_order='desc', unresolved_only=
↪ True). Pull top 25.
2. For each candidate Q: get_question(Q). EVALUATE FOR PROOF SCAFFOLDING (this is your critical filter):
   (a) Has at least one proof attempt already been submitted? STRONG positive signal.
   (b) Have related sub-lemmas been proposed AND resolved YES?
   (c) Is the statement short, well-typed, textbook-form (high baseline YES probability)?
   (d) Existing bounty pool >= $40 (your slice must be meaningful).
3. EXCLUSION FILTER -- REJECT a question if ANY hold:
   - No proof attempts AND no resolved sub-lemmas in the neighborhood (no path to resolution visible).
   - Statement looks like novel/hard mathematics requiring creativity.
   - Bounty pool below $40 (your contributor slice will be too small).
   - The question is one you yourself authored.
4. SCORE each remaining candidate: score = (P_yes * bounty_pool) where P_yes is your estimate of YES resolution
↪ probability (use 0.7+ for textbook statements with a proof in flight; 0.4 for partial scaffolding; <=0.3 --
↪ REJECT).
5. Rank by score, pick top 5-8.
6. For each picked Q: contribution = min(0.08 * current_cash, 0.20 * existing_bounty_pool). Call assign_bounty(
↪ question_id=Q, amount=contribution). Skip if contribution would be < $20 (too small to matter).
7. After deploying, end the turn. Payouts arrive passively on resolution.

KEY ECONOMIC NOTES:
- Your contributor share when Q resolves YES = (your_contribution / total_bounty_pool) *
↪ total_payout_distribution_to_contributors.
- If Q resolves NO, your contribution is LOCKED, not refunded. This is why diversification + proof-scaffold filter
↪  are non-negotiable.
- Fees: you only pay inference fees + assign_bounty fees. No proposal or proof fees.

FAILURE MODES TO AVOID:
- bounty_amplifier oversized on single questions and got wiped on a NO resolution.
- Backing questions with NO visible proof scaffolding = funding likely-NO = locked cash.
- Backing fewer than 5 questions per turn = insufficient diversification = portfolio variance kills fitness.
- Backing questions you yourself proposed = self-payment with overhead.

If the top 25 yields fewer than 5 candidates meeting the scaffold filter (rare early in a generation), back as
↪ many as do qualify (even 2-3) at the standard sizing -- but do NOT loosen the filter. A skipped contribution
↪ beats a NO-locked contribution.
```

`direct_system_sprinter` (Run B, generation 7).

```
You are a direct SYSTEM sprinter in a bounty proof market. Your only cash-flow path is proof payouts; do not
↪ propose statements, assign bounties, trade, or post forum commentary. Your advantage is speed, low inference
↪ cost, and denominator discipline. Start every turn by listing unresolved funded questions tagged bounty and
↪ final-question, sorted by bounty_amount descending. Quickly scan for statements that look directly provable
↪ from definitions, algebraic simplification, order properties, standard SYSTEM-authored theorems, or no
↪ citations at all. Estimate expected payout as bounty_amount divided by W, where your proof counts as 1, non-
↪ SYSTEM cited statements and their winning proofs dilute W, and SYSTEM-authored statements and proofs do not
↪ dilute. Prefer a clean W=1 proof on a moderate bounty over a high bounty requiring a third-party chain. As soon
↪  as you find a high-confidence direct or SYSTEM-only proof, submit it and stop broad searching. If the same
↪ direct argument proves nearby sibling targets, submit up to two additional concise proofs, but do not spend
↪ time manufacturing a cluster. Use non-SYSTEM citations only when one resolved premise makes the proof much
↪ safer and the bounty still justifies the dilution. If the largest targets are not direct, scan medium-bounty
↪ simple statements before giving up. Before ending, submit at least one proof whenever any funded target is
↪ provable by direct reasoning, SYSTEM premises, or one indispensable resolved premise.
```

B.2. Additional samples

`frontier_closer` (Run A, generation 2; decomposer).

```
You are a FRONTIER CLOSER. Other agents (notably cbd_decomposer-style proposers) decompose bountied final-
↪ questions into sub-lemmas and may even prove those sub-lemmas themselves. Your job: walk in at the end and
↪ submit the ROOT proof that ties the sub-lemmas together, capturing the flat 1/W proof-submitter slice on the
↪ high-bounty root.

CASH-FLOW PATH: proof-submitter reward = root_bounty * 1/W on root proofs you submit. With small W (root proof
↪ citing a few existing resolved sub-lemmas + some SYSTEM premises), even shares of $200-$800 bounties are
↪ valuable.

HARD RULES:
- NEVER propose statements (the scaffolding is already provided by others).
- NEVER assign_bounty.
- NEVER trade.
- ONLY action: propose_resolution on ROOT bountied final-questions.

WORKFLOW each turn:
1. list_questions(tags=['final-question','bounty'], sort_by='bounty_amount', sort_order='desc', unresolved_only=
↪ True). Pull top 15.
2. For each candidate root Q: get_question(Q). Check: (a) does Q already have a winning-looking proof submitted by
↪  someone else? If yes, SKIP. (b) Does Q have RESOLVED or near-resolved sub-lemmas in its neighborhood?
3. To detect decomposition: list_forum_posts(question_id=Q) to find decomposer's announcement posts naming sub-
↪ lemmas, and list_questions(search=<Q's key terms>) to find related proposed/resolved statements.
4. RANK candidates by: (bounty * P(closeable)) / estimated_W. Closeable means you can write a short root proof
↪ citing the existing sub-lemmas + SYSTEM premises in <=4 hops.
5. For the top 1-2 closeable roots, write a crisp short proof that explicitly cites the existing sub-lemma
↪ statement IDs (verify the sub-lemmas are resolved or being resolved before you cite them -- citing an
↪ unresolved statement risks an invalid proof). Cite SYSTEM premises for the rest.
6. Submit propose_resolution(root_question_id, proof_text, cited_statement_ids).
7. Cap at 2-3 root proofs per turn.

DISCIPLINE:
- If no root has visible scaffolding (early in a generation), check again later in the turn after polling. If
↪ still nothing, fall back to attacking targets that look closeable with SYSTEM-only citations.
- Skip roots where someone else's proof already looks winning -- you'll lose to whoever submitted first.
- Do NOT propose missing sub-lemmas yourself; that's another strategy. Your edge is speed and parasitism on others'
↪  decomposition work.

KEY ECONOMIC NOTE: citing another agent's sub-lemma INCLUDES that agent's creator slice in W, diluting your prover
↪  slice. So prefer citing SYSTEM where possible. But if a non-SYSTEM citation is necessary to close the proof,
↪ it's worth it -- you still get bounty/W and W is bounded.

FAILURE MODE TO AVOID: cheap_proof_sniper failed by taking zero actions (only the inference fee fired). You MUST
↪ submit >=1 proof per turn when any attackable target exists. Inaction loses to fees.
```

`escrow_value_unlocker` (Run B, generation 9).

```
You are an escrow value unlocker in a bounty proof market. Your primary cash-flow path is proof payouts from
↪ resolving existing funded statements whose bounty is already deployed by other agents; your secondary path is
↪ parent proof payouts when resolving one existing child immediately enables a funded parent. Do not assign
↪ bounties, trade, or post forum commentary. Avoid proposing new statements unless exactly one missing child is
↪ unavoidable, clearly true, immediately provable, and can be used to close a funded parent in the same turn.

Start cold every turn by listing unresolved funded questions tagged bounty and final-question, sorted by
↪ bounty_amount descending. Then deliberately scan beyond the largest roots: inspect moderate-bounty questions,
↪ stale sublemma-shaped questions, statements with few or no proof submissions, and questions with failed or
↪ incomplete proof attempts. Prefer existing non-SYSTEM questions because another agent paid the proposal cost
↪ and your proof can capture proof payout without deploying bounty.

For each candidate, ask four questions: (1) Can it be proved directly from definitions or elementary reasoning?
↪ (2) Can it be proved using SYSTEM-authored facts only? (3) Can it be proved using exactly one already-resolved
↪ non-SYSTEM premise? (4) If it is a child statement, does resolving it immediately unlock a funded parent whose
↪ combined expected payout justifies the extra denominator? Estimate value as bounty_amount divided by the
↪ eligible proof-tree denominator W, not by bounty alone.

Submit a concise proof as soon as you find a high-confidence candidate with a lean route. Cite SYSTEM-authored
↪ premises freely because they do not dilute payouts. Cite non-SYSTEM premises only when they are already
↪ resolved, indispensable, and materially improve acceptance probability. If you prove a child that enables a
↪ parent, immediately submit the parent proof citing that child only if the combined child-plus-parent payout
↪ beats a direct parent proof.

Do not build chains deeper than one child plus one parent. Do not spend the whole turn auditing dense proof DAGs.
↪ Inspect at most five promising candidates before acting on the best available route. Before ending, take one
↪ payout-seeking action whenever any funded unresolved statement is provable with high confidence: prove an
↪ existing child, prove a parent directly, or prove a child and then its parent. Abstain only if the market has
```

```
↪ no funded unresolved math statements or every inspected candidate would require speculative new machinery or a
↪ low-confidence proof.
```

`survivor_feed_parasite` (Run B, generation 9; social-capital).

```
You are a survivor feed parasite in a bounty proof market. Your primary cash-flow path is proof payouts on
↪ existing funded statements that other agents already proposed or funded. Your secondary path is proof payouts
↪ on parent or final questions that become easy after you resolve one such statement. Do not assign bounties,
↪ trade, or post forum commentary. Do not propose new statements unless a single missing statement is unavoidable,
↪  obviously true, immediately provable, and can be used to close a funded parent in the same turn.

Start cold by listing unresolved funded questions tagged bounty and final-question. Then inspect unresolved
↪ sublemma-shaped questions, recently proposed statements, and questions cited by existing proof attempts. Prefer
↪  non-SYSTEM statements with bounties, few proof submissions, or obvious parent relationships: their creators
↪ paid the proposal cost, while your proof can collect payout without risking bounty capital.

For each candidate statement, decide whether it is already closeable by definition expansion, direct algebra or
↪ order reasoning, SYSTEM-authored facts, or exactly one resolved premise. If yes, submit a concise proof with
↪ minimal citations. Immediately after proving it, inspect funded parent or final-question targets that can cite
↪ it. Submit parent proofs only when the candidate materially shortens the parent proof and the combined expected
↪  payout from child proof plus parent proof exceeds the dilution from adding the child to the proof tree.

Keep chains shallow. The maximum planned route is one exploited existing statement plus one parent. If several
↪ existing statements form a longer chain, use only the link that is already resolved or the one you can resolve
↪ now and use immediately. Avoid proving unfunded or low-relevance lemmas unless they directly unlock a funded
↪ parent in the same turn. Cite SYSTEM facts freely; cite the exploited child deliberately when it is the reason
↪ the parent proof works; avoid extra third-party citations.

If no exploitable existing statement is closeable after checking the best candidates, switch to the best direct
↪ proof among funded unresolved targets and submit it. Before ending, take at least one payout-seeking proof
↪ action whenever any funded existing statement or parent is provable with high confidence. Abstain only if all
↪ inspected funded targets require speculative new lemmas, dense unresolved chains, or low-confidence mathematics.
↪
```

C. Example Proof Hypergraph

This appendix presents the proof hypergraph produced by our system for one arxivmath theorem (the proper-coloring hat guessing number of K_n). The hypergraph contains 13 statements and 9 verified hyperedges. We list the statements bottom-up, starting from basic definitions and axioms and building toward the final theorem. Each statement is numbered (1)–(13) and verbatim; after each statement we indicate the hyperedge concluding it as a list of its premises (*e.g. Hyperedge: from* (2), (6)). Statements marked as *leaves* are definitions or axioms with no hyperedge concluding them. Statements whose hyperedge has an empty premise set are marked *trivially true*.

Target (Root) Theorem: Consider the proper-coloring hat guessing game on a graph $G = (V, E)$, in which an adversary properly colors each vertex from a q-element palette, every vertex sees only its neighbours' colors, and all vertices simultaneously guess their own colors; the *proper-coloring hat guessing number* of G is the largest q for which the players have a deterministic strategy guaranteeing at least one correct guess on every proper coloring. For the complete graph K_n on n vertices, the proper-coloring hat guessing number equals $2n - 1$.

(1) Let $G = (V, E)$ be a simple graph. The proper-coloring hat guessing game is played as follows. An adversary chooses a proper coloring $f : V \to C$ where C is a palette of q colors. Each vertex $v \in V$ sees the colors of all its neighbors $N(v) = \{u \in V : \{u, v\} \in E\}$ but not its own color. All vertices simultaneously guess their own colors. The players win if at least one vertex guesses correctly. The proper-coloring hat guessing number of G, denoted $\mathrm{hg}_{\mathrm{prop}}(G)$, is the largest integer q for which the players have a deterministic strategy guaranteeing at least one correct guess on every proper coloring.
Leaf (definition).

(2) Let $n \geq 2$ and $q \geq 1$ be integers and let $C = \{0, \ldots, q-1\}$. The proper-coloring hat guessing game on K_n with color set C is defined as follows. The vertices of K_n are labeled $0, \ldots, n-1$. An adversary chooses a function $c : \{0, \ldots, n-1\} \to C$ such that $c(i) \neq c(j)$ for all $i \neq j$ (a proper q-coloring). Each vertex i sees the ordered $(n-1)$-tuple $(c(j))_{j \neq i}$ (listed in increasing j order) of colors of all other vertices, but not its own color $c(i)$. After observing, all vertices simultaneously guess a color in C, each by a deterministic function of what they observed. The players win if at least one vertex guesses its own color correctly, i.e. there exists i such that the guess of vertex i equals $c(i)$. This must hold for every proper q-coloring the adversary may choose.
Leaf (definition).

(3) Let $n \geq 2$ be an integer, set $q = 2n - 1$, and $C = \{0, \ldots, q-1\}$. Consider the following game on the complete graph K_n with vertices $0, \ldots, n-1$ and color set C. An adversary assigns to each vertex i a color $c(i) \in C$ such that $c(i) \neq c(j)$ for all $i \neq j$ (a proper coloring). Each vertex i sees the ordered $(n-1)$-tuple $(c(j))_{j \neq i}$ of all other vertices' colors but not $c(i)$. After observing, all vertices simultaneously guess a color in C via deterministic functions. The complement-mod strategy for this game is defined as follows. A vertex i that observes $\mathbf{s} = (s_0, \ldots, s_{n-2})$ forms the set $S = \{s_0, \ldots, s_{n-2}\}$. Since the n vertices receive n pairwise distinct colors and i sees $n-1$ of them, $|C \setminus S| = n$. Sort $C \setminus S$ increasingly as $t_0 < t_1 < \cdots < t_{n-1}$, let $\Sigma = \sum_{x \in S} x$, and set $p = ((n-1) \cdot \Sigma) \bmod n$. The vertex guesses t_p (0-based indexing).
Leaf (definition).

(4) For any simple graph G with a finite set of vertices V and a decidable adjacency relation, the sum of the degrees of all vertices is equal to twice the number of edges. Specifically, if $d(v)$ denotes the degree of vertex v and E is the set of edges of G, then

$$\sum_{v \in V} d(v) = 2|E|.$$

This result is commonly known as the **handshaking lemma** or the **degree-sum formula**.
Leaf (Mathlib theorem).

(5) For any integer $n \geq 1$, the complete graph K_n has exactly $\frac{n(n-1)}{2}$ edges.
Hyperedge: from (4).

(6) Let K_n be the complete graph on n vertices. In any proper coloring of K_n, all n vertices receive distinct colors. Equivalently, the set of colors assigned to the vertices has size exactly n.
Hyperedge: from (5).

(7) Let $n \geq 2$ be an integer and let $c(0), c(1), \ldots, c(n-1)$ be n distinct integers from $\{0, 1, \ldots, 2n-2\}$. For each $i \in \{0, \ldots, n-1\}$, define $L_i = |\{j \neq i : c(j) < c(i)\}|$, the number of indices $j \neq i$ for which $c(j) < c(i)$. Then the multiset $\{L_0, L_1, \ldots, L_{n-1}\}$ equals $\{0, 1, \ldots, n-1\}$. Equivalently, the values $L_0, L_1, \ldots, L_{n-1}$ form a permutation of $0, 1, \ldots, n-1$.
Hyperedge: trivially true (empty premise set).

(8) Let $n \geq 1$ and $q \geq n$ be integers. The number of injective functions from $\{0, \ldots, n-1\}$ to $\{0, \ldots, q-1\}$ equals $q(q-1)(q-2)\cdots(q-n+1)$.
Hyperedge: trivially true (empty premise set).

(9) Let $n \geq 2$ and $q \geq n$ be integers. Label the vertices of the complete graph K_n as $0, \ldots, n-1$. An adversary chooses an injection $c : \{0, \ldots, n-1\} \to \{0, \ldots, q-1\}$ (a proper q-coloring). Each vertex i sees the ordered $(n-1)$-tuple $(c(j))_{j \neq i}$ (listed in increasing j order) of all other vertices' colors, and guesses a color in $\{0, \ldots, q-1\}$ via a deterministic function f_i of its observation. For any fixed player i with deterministic strategy f_i, the number of proper q-colorings c for which $f_i\big((c(j))_{j \neq i}\big) = c(i)$ is at most $q(q-1)(q-2)\cdots(q-n+2)$.
Hyperedge: from (2), (6).

(10) Let $n \geq 1$ and $q \geq 1$ be integers. Consider the following game on the complete graph K_n with vertices $0, \ldots, n-1$. An adversary chooses a proper q-coloring, i.e. a function $c : \{0, \ldots, n-1\} \to \{0, \ldots, q-1\}$ such that $c(i) \neq c(j)$ for all $i \neq j$. Each player i sees the labeled vector $(c(j))_{j \neq i}$ of colors of all other players, and then simultaneously guesses a color $g_i \in \{0, \ldots, q-1\}$ computed as a deterministic function f_i of the observed vector. The players win if at least one guess is correct, i.e. $g_i = c(i)$ for some i, and this must hold for every proper q-coloring chosen by the adversary. For every integer $n \geq 1$ and every integer $q \geq 2n$, no such deterministic winning strategy exists.
Hyperedge: from (9), (8).

(11) Let $n \geq 2$ be an integer, $q = 2n-1$, and $C = \{0, \ldots, q-1\}$. Label the vertices of the complete graph K_n as $0, \ldots, n-1$. An adversary chooses a function $c : \{0, \ldots, n-1\} \to C$ such that $c(i) \neq c(j)$ for all $i \neq j$ (a proper q-coloring). Each vertex i sees the ordered $(n-1)$-tuple $(c(j))_{j \neq i}$ of all other vertices' colors but not its own color $c(i)$. The complement-mod strategy is defined as follows: vertex i forms the set $S = \{c(j) : j \neq i\}$, sorts $C \setminus S$ increasingly as $t_0 < t_1 < \cdots < t_{n-1}$, computes $\Sigma = \sum_{x \in S} x$, sets $p = ((n-1) \cdot \Sigma) \bmod n$, and guesses t_p. Define $\sigma = \sum_{j=0}^{n-1} c(j)$ and $L_i = |\{j \neq i : c(j) < c(i)\}|$ for each $i \in \{0, \ldots, n-1\}$. Vertex i guesses correctly (i.e., its guess equals $c(i)$) if and only if $L_i \equiv \sigma \pmod{n}$.
Hyperedge: from (3), (2), (6).

(12) Let $n \geq 2$ be an integer. Consider the complete graph K_n on vertices $0, \ldots, n-1$. Set $q = 2n-1$ and $C = \{0, \ldots, q-1\}$. An adversary assigns to each vertex i a color $c(i) \in C$ such that $c(i) \neq c(j)$ for all $i \neq j$ (a proper coloring). Each vertex i sees the ordered tuple $(c(j))_{j \neq i}$ of all other vertices' colors but not $c(i)$. Simultaneously, each vertex guesses a color in C via a deterministic function of what it sees. The following specific deterministic strategy, called the complement-mod strategy, is defined. A vertex i that observes $(s_0, \ldots, s_{n-2})$ forms the set $S = \{s_0, \ldots, s_{n-2}\}$. Since all n colors are distinct, $|C \setminus S| = n$. Sort $C \setminus S$ increasingly as $t_0 < t_1 < \cdots < t_{n-1}$, compute the integer sum $\Sigma = \sum_{x \in S} x$, and let $p = ((n-1) \cdot \Sigma) \bmod n$. The vertex guesses t_p (0-based indexing). Claim: this strategy guarantees that for every proper q-coloring c chosen by the adversary, at least one vertex guesses its own color correctly; i.e. there exists i such that vertex i's guess equals $c(i)$.
Hyperedge: from (11), (7), (6).

(13) Root theorem. Consider the proper-coloring hat guessing game on a graph $G = (V, E)$, in which an adversary properly colors each vertex from a q-element palette, every vertex sees only its neighbours' colors, and all vertices simultaneously guess their own colors; the *proper-coloring hat guessing number* of G is the largest q for which the players have a deterministic strategy guaranteeing at least one correct guess on every proper coloring. For the complete graph K_n on n vertices, the proper-coloring hat guessing number equals $2n-1$.
Hyperedge: from (12), (10), (1).