Scaling Discovery through Test-Time Communication

Anonymous authors
Paper under double-blind review

Abstract

Science advances not in isolation but through collaboration, yet existing agentic systems capture little of this. Whether communicating agents help remains an open question with mixed prior results. We show that *test-time communication can substantially outperform independent parallel attempts on challenging tasks, where sharing a breakthrough can push the whole group forward*. We first study the effect of scaling multi-agent test-time communication, where agents have no predefined roles and communicate via a shared directory, on ARC-AGI-3, a benchmark requiring novel problem solving. We find that a team of k communicating agents, team@k, matches the success rate of $4k$ independent agents, and this advantage grows with k, suggesting gains compound with scale. The effect is not merely efficiency: a task that no single agent can solve, a team of agents can solve reliably. Furthermore, these gains transfer to research-oriented tasks, given sufficient compute. On polyomino packing, communicating agents outperform best@k and exceed the prior best-known score. On MNIST classifier compression, a team of four communicating agents produced a 1,957-byte classifier achieving 99.4% test accuracy, smaller than both the best-known human solution and the best single-agent result. These gains are not unconditional. Independent agents may outperform communication when compute is limited or when a clear measure of progress is absent. However, under sufficient compute and clear verifier feedback, multi-agent communication consistently yields stronger results.

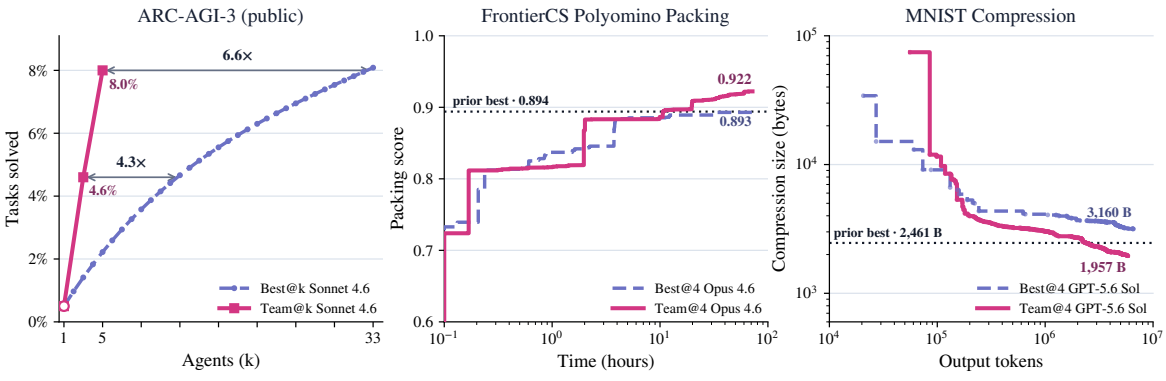


Figure 1: **Communication outperforms independent agents in open-ended tasks.** With communication, a team of k communicating agents not only outperforms best@k but scales success rate on ARC-AGI-3 faster. On algorithmic and ML optimization tasks, team@k beats single-agent runs and best-known human solutions, yielding new state-of-the-art results.

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

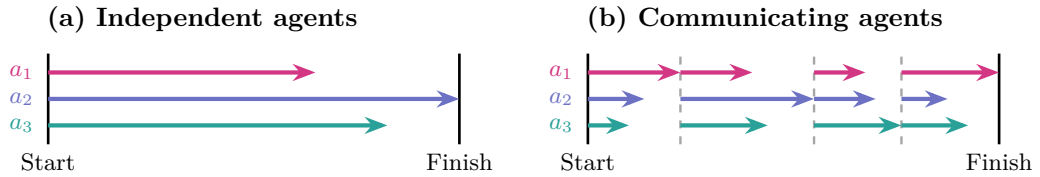


Figure 2: **Verified progress sharing allows a team to advance faster. (a)** Under independent parallel sampling, each agent advances only along its own trajectory, so success requires at least one agent to complete the entire trajectory independently. **(b)** By sharing verified progress, the task is divided into successive phases, and at each checkpoint, all agents continue from the improved state, where a different agent may supply the next breakthrough.

1 Introduction

For test-time compute, it is standard to sample k independent outputs from a large language model (LLM) and aggregate them by majority vote or best@k (Wang et al., 2023; Brown et al., 2024; Li et al., 2024; Huang et al., 2025; Snell et al., 2025). But independent attempts cannot share discoveries mid-search, a loss that is especially costly for LLM agents, whose hours-long trajectories of tool use and code execution produce intermediate results, failed experiments, and reusable artifacts that could guide one another. Communication is the natural remedy, yet its benefits are unclear: debate often fails to beat chain-of-thought despite more compute (Choi et al., 2025; Smit et al., 2024; Zhang et al., 2025d) and agents can conform to incorrect majorities or dilute. expertise (Wu et al., 2025; Pappu et al., 2026). Anthropic (2026b) report that even successful coordination can cost proportionally more tokens. On the other hand, Multi-agent systems reliably help when tasks decompose cleanly into parallel subtasks (Kim et al., 2025). Can communication offer more than parallelism?

Science suggests that it can. Scientists coordinate by adjusting their research to the results achieved by others (Polanyi, 1962; Donoho, 2024), while accumulated empirical evidence helps distinguish competing explanations and turn varied efforts into collective progress (Boudreau & Lakhani, 2015; Strevens, 2020). This process is iterative rather than merely parallel. Collaboration should therefore do more than divide and conquer. One agent's observation redirects another's search, and partial discoveries combine into a result that no individual reaches alone. We ask whether agent communication can produce the same effect.

To this end, we study a *minimal form of test-time communication* in which identical agents receive the same open-ended objective and communicate through a shared workspace, without predefined roles or a central orchestrator. Autonomously and asynchronously, the agents can exchange intermediate results, failures, and artifacts while their search is still unfolding. Our central comparison pits a team of k communicating agents, **team@k**, against the best result from k independent agents, **best@k**, under the same per-agent resources. This isolates the value of communication from the gains produced by additional parallel attempts.

We first study `ARC-AGI-3` (ARC Prize Foundation, 2026) using Claude Sonnet 4.6. In terms of full-game success rate, team@3 matches best@13 and team@5 matches best@33, making them as effective as 4.3× and 6.6× as many independent agents (Figure 1). Communicating teams can solve games that independent agents rarely or never solve. These benefits transfer to longer-horizon research problems. On Frontier-CS polyomino packing (Mang et al., 2026), where agents pack polyominoes into a minimum-area rectangle, team@4 scores 0.922 with Claude Opus 4.6 and 0.910 with Sonnet 4.6, compared with 0.893 and 0.891, respectively, for the strongest solo run of each model and 0.894 for the prior best-known result. On MNIST classifier compression under an accuracy constraint, we evaluate team@4 with GPT-5.6 Sol over 96 hours. The best team produces a state-of-the-art 1,957-byte classifier at 99.4% accuracy, beating the 2,461-byte best-known human solution and single-agent baselines.

These results suggest that test-time communication is most useful when agents have enough compute to build on one another's discoveries and can objectively assess whether those discoveries improve on earlier results. Sufficient compute allows teams to overcome initial coordination costs, while an accessible verifier, such as level success or a solution scorer, helps agents decide which discoveries to adopt. We call this mechanism *verified progress sharing*. As Figure 2 illustrates, independent agents succeed only if one agent completes the entire trajectory alone, whereas a communicating team can take each successive breakthrough from a different agent. By building on successive breakthroughs from different members, a communicating team can turn parallel exploration into cumulative progress.

When agents cannot verify progress, however, communication may offer less. On Terminal-Bench 2.0, where available feedback may not reliably rank varying approaches, we find that team@2 improves over single agents but does not outperform independent best@2. This distinction offers an explanation for the prior negative results, where agents lack a verifier to distinguish better solutions from worse ones and therefore abandon stronger solutions in favor of weaker ones (Choi et al., 2025; Pappu et al., 2026). Together, our findings show that communication can offer more than divide and conquer parallelism and help identify the conditions under which test-time communication excels. Our contributions are as follows:

1. **Communicating agents show stronger scaling than independent agents.** On the `ARC-AGI-3` suite of games, matching the solve rate of team@5 requires 33 independent agents. The multiplier grows with team size, from 4.3× for team@3 to 6.6× for team@5, suggesting that the gains from communication compound with scale, as seen in Figure 1.
2. **Test-time communication unlocks tasks that single agents cannot.** Team@3 lifts the solve rate on the game `FT09` on `ARC-AGI-3` from 9.4% for a single agent to 90%. Strikingly, team@5 solves the game `LP85` in 65% of trials, while none of 64 single agents do. Team@5 also improves the average level reached over best@5 even on unsolved games.
3. **Communication achieves state-of-the-art results on algorithmic and ML optimization tasks.** On Frontier-CS polyomino packing and ML compression, a team of agents outperforms independent agents and surpasses the prior best-known human solution. Agents reach these solutions by refining one another's discoveries, including ideas from peers' failed approaches.
4. **Communication benefits from verifiable progress and sufficient compute.** At low budgets, team@k incurs a coordination tax that outweighs its benefits. We identify *verified progress sharing* as a mechanism through which agents build on successive discoveries to overcome the coordination tax given enough compute.

2 Agentic Communication at Test-time

We study a minimal form of test-time communication instantiated through a shared workspace among identical agents. In each team@k trial, the harness launches k CLI agents concurrently in the same task container. They use the same model, tools, task instruction, and communication prompt. Each agent has a separate model context and designated scratch directory, while all agents share the task filesystem and communication artifacts.

Agents communicate directly through an append-only *communication log*, which acts as an asynchronous broadcast channel. Additional shared records contain adopted approaches, disconfirming evidence, and a score log of the approaches so far. Without assigned roles or a central orchestrator, the only protocol-level allocation is slot ownership. Agents race to create numbered slot directories with an atomic filesystem operation, so no two agents claim the same slot. After claiming a slot, each agent is asked to declare a distinct approach by considering the already-claimed slots. During execution, agents publish concise findings

Table 1: **ARC-AGI-3 per-level solve rate.** Proportion of trials reaching within d levels of a full solve over 25 games. Communication pays off most where the task is hardest.

Levels from target	team@3	best@3	best@13	team@5	best@5	best@33
5	52.8	45.4	63.9	64.1	52.8	69.5
4	28.2	25.8	46.5	33.6	32.9	55.4
3	16.6	9.7	20.8	22.9	12.8	31.3
2	7.2	5.0	9.3	16.2	6.1	15.8
1	4.8	2.6	6.3	10.0	3.7	10.2
(Solved) 0	**4.6**	1.4	4.7	**8.0**	2.2	8.1

with timestamps and reproducible evidence whenever they make notable progress, allowing peers to reproduce or build upon their results. Results placed on the leaderboard should include measured outcomes and reproduction instructions. An agent is to adopt a peer's approach only after observing a clearly better result, and even after adoption, it should preserve meaningful variations. The protocol thus combines diverse exploration, preserved by slot ownership and and a prompt (Appendix B.3) that encourages diversification, where verifier scores let agents filter out poorly performing approaches. Much prior work explore a range of multi-agent systems as well. We discuss related work in Appendix A.

Setup. We study test-time communication on three tasks covering a different aspect of open-ended research. **ARC-AGI-3** (ARC Prize Foundation, 2026) tests novel problem solving and interactive discovery by asking agents to solve unfamiliar grid-world games without instructions, inferring the rules, the goal, and each control from play alone. Each game has 6-10 levels, and this leveled structure makes breakthroughs cleanly measurable. We evaluate on all 25 public games with Sonnet 4.6 under its native per-level action budget. **Frontier-CS polyomino packing** (Mang et al., 2026) tests algorithmic optimization on an NP-hard problem. Agents must write a C++17 program that packs reflected and rotated polyominoes into a minimum-area rectangle, scored on 70 hidden test cases. We test Sonnet 4.6 and Opus 4.6 with teams of three or four agents, against the best published score (Qu et al., 2026). **MNIST Classifier Compression** tests empirical ML research. Agents must train and compress a self-contained MNIST classifier (Le Cun et al., 1998) that reaches 99.4% test accuracy or better without inspecting the test images or labels. Among qualifying artifacts, only the compressed size of the submission (inference code + model) counts. We test a team of four GPT-5.6 Sol agents over a 96-hour period, against the best prior result of 2,461 bytes from Dhairyashil R. G. (2024).

3 Experimental Results

ARC-AGI-3. Despite `ARC-AGI-3`'s visual simplicity, the most capable LLM-based agents fail on most games without a specialized harness. For this benchmark, we use Claude Sonnet 4.6 agents through Copilot, which reach a final success rate below 1%.

Table 1 reports the percentage of runs that reach success or come within d levels of it. Across 25 games, team@k outperforms best@k at every level for both $k = 3$ and $k = 5$, and the advantage widens with depth, from 1.2× to 3.6× at $k = 5$. The leftmost panel of Figure 1 shows a further encouraging pattern in which team@k rises monotonically with team size and faster than best@k. Matching a communicating team takes 4.3–6.6× as many independent agents, a multiplier that itself grows with k, though returns may diminish beyond five agents. The improvement is striking on `LP85`, where 64 single-agent trials yield no successes, yet team@5 achieves a 65% solve rate (Figure 8, Appendix C). Nor is the effect limited to the

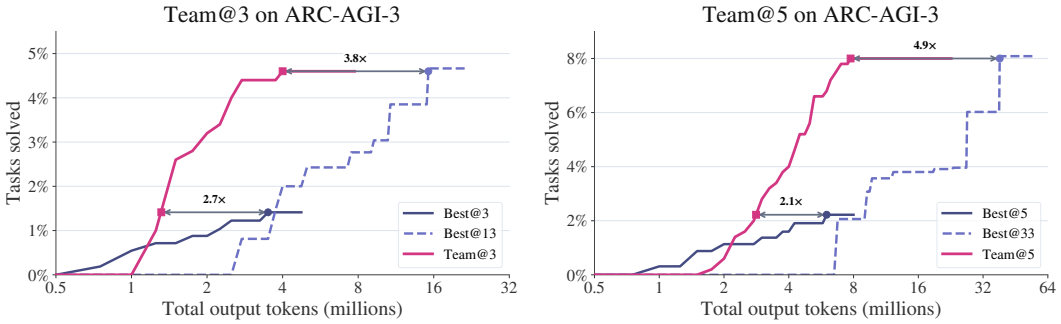


Figure 3: **Token efficiency of communicating agents.** Solve rate against cumulative output tokens for team@k, best@k, and the number of independent agents needed to match the team's final solve rate. Best@k never reaches the team's final rate at any budget. For k=3 and k=5, respectively, matching that rate takes 3.8× and 4.9× more tokens than the team spends. The gain is not immediate, as best@k leads in a low-compute regime

largest team. On FT09, the solve rate climbs from 25.9% with best@3 to 90% with team@3. Although these gains are concentrated, this reflects the limits of the base model rather than of communication, since most of the 25 games remain beyond Sonnet 4.6 under either setting.

Moreover, zero-percent solve rates obscure meaningful gains in progress. Figure 8(a) reports the average furthest level reached. Under team@5, several never-solved games still advance by roughly a full level, showing that test-time communication produces real progress across most games. The largest gain is +4.30 levels, while no loss exceeds 0.85 levels. When coordination fails, the cost is negligible. When it succeeds, the gains can be substantial.

Agents in collaboration outperform agents in isolation. Team@k outperforms best@k in per-level solve rate, while team@k grows faster than best@k as k grows.

Is test-time communication simply a matter of spending more tokens than independent agents do? Figure 3 paints a more nuanced picture. By the end, team@k spends 8 million output tokens with three agents and 24 million with five, nearly twice what best@k spends. However, to solve more than 1% of ARC-AGI-3 tasks, team@k reaches any given accuracy with fewer tokens, with best@3 spending 2.7× as many as team@3 and best@5 spending 2.1× as many as team@5. Matching the solve rate of test-time communication is even costlier, as pools of 13 and 33 independent agents must spend as much as 3.8–4.9× the team's tokens.

We further study SB26 and LP85, two games that benefit from communication, with two action-budget ablations. We first give a single agent 5× the native action budget, matching the total budget of standard team@5 or best@5. This configuration tests whether a longer horizon of actions of a single agent can match the performance of a team of agents. We also reduce team@5's per-agent action budget to 0.2×, so its total budget matches a single agent.

At matched total budget, communication wins. On SB26, best@5 leads early at 80% to 75%, but team@5 overtakes at level 2 and solves the final level in 15% of trials against roughly 8% for best@5 and zero for the long-horizon single agent. LP85 separates further, where team@5 holds 90% through level 5 and finishes at 65% while best@5 falls from 72% to zero by level 4 and the single agent peaks at 10%. A longer horizon alone does not substitute for coordination. This advantage, however, vanishes once the budget is reduced. At 0.2× per agent, team@5 drops to zero after level 1 on LP85 and after level 3 on SB26, in both cases losing to a single agent spending the same total. Communication pays only when each agent has enough budget to explore on its own.

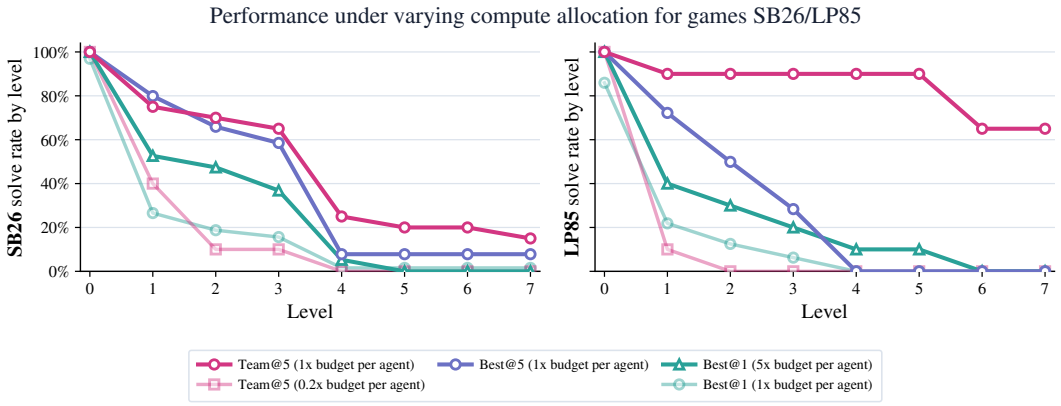


Figure 4: **Effective collaboration depends on compute allocation.** For games SB26 and LP85, we vary the per-agent action budget while holding the total action budget at 5× the native budget (solid line). We also consider the case in which a team must divide one agent's native budget (faded line). The marker shape indicates per-agent budget, while the color indicates agent configuration. When each agent is given the native budget, per-level team@5 leads at every level past the first, ahead of both best@5 and a single agent given 5× the budget. Cutting the same five agents to 0.2× each drops team@5 below a single agent.

Communication is more time- and token-efficient given sufficient compute. Communicating teams pay an initial coordination tax but overtake best@k given enough compute. Independent agents require more output tokens to match team@k, while teams can beat single agents with a budget of an entire team. At small budgets, however, teams divide already constrained resources and underperform.

Throughout this paper, we take solve rate as our primary metric, since our question is whether communication and larger teams, that is, more test-time compute, lead to more tasks solved in `ARC-AGI-3`. The official metric, by contrast, scores a run not only by what it clears but by how economically it plays, using a metric called relative human action efficiency (RHAE) (ARC Prize Foundation, 2026). We define RHAE in Eq. 2 and use it to show that test-time communication improves per-agent efficiency.

Figure 5 compares the RHAE of independent Sonnet 4.6 agents with communicating teams across all 25 games. Communication improves not only the strongest team member but also the average member. The best agent's mean RHAE rises from 3.8% for a single agent to 9.7% with team@3 and 13.6% with team@5, exceeding the corresponding best@3 and best@5 values of 7.0% and 8.8%. This gap shows that the gain is not merely the result of selecting the best outcome from more agents. The average communicating agent also improves with team size, while the average independent agent remains at 3.8% (by definition). Interestingly, the average agent in team@5 matches the best of five independent agents, at 8.9% versus 8.8%. Together, we show that communication improves both the average and the strongest team member, rather than improving team performance only by pooling more attempts.

Communication boosts the efficiency of each agent. Communication improves all team agents by sharing progress and avoiding failure attempts. The average team agent reaches 8.9%, performing as efficiently as the best of five independent agents.

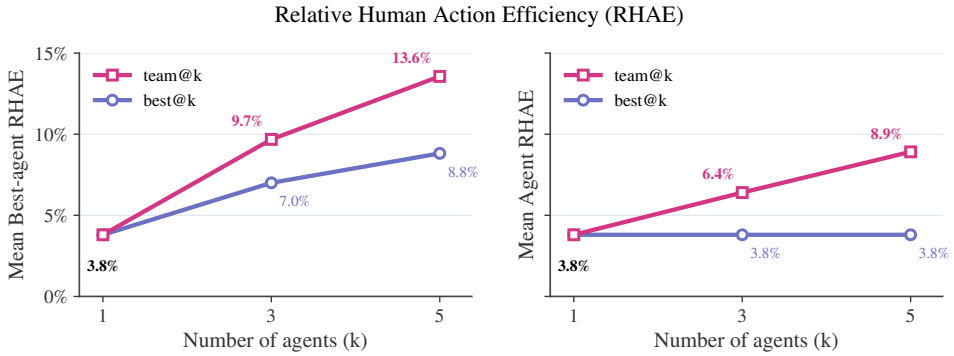


Figure 5: **RHAE of communicating agents.** RHAE (Eq. 2) for the best agent and average agent across all games. RHAE scores trajectories based on how quickly an agent solves each level relative to human performance. Each agent's action efficiency increases with team size, while the best agent in the team is more efficient than the best independent agent.

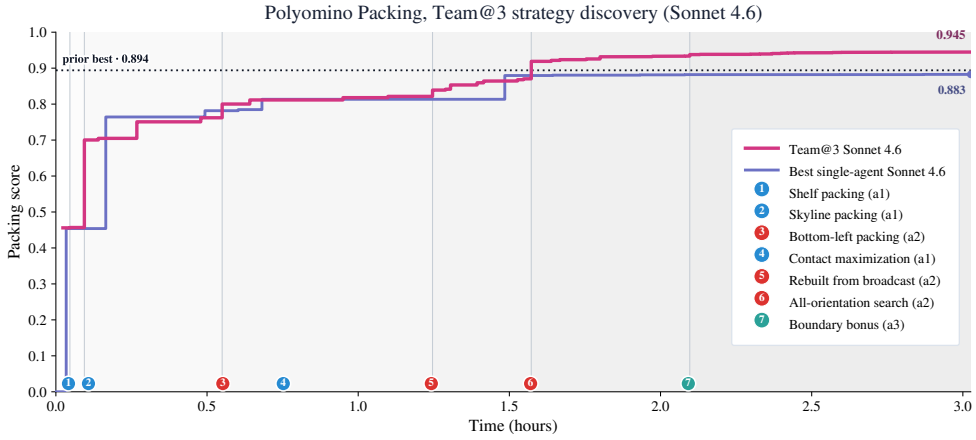


Figure 6: **Communicating agents surpass the single-agent plateau.** Best-so-far packing score for the highest-scoring run among 60 single-agent trials and 20 team@3 trials. Team@3 pulls ahead at 1.6 hours and reaches a new state-of-the-art 0.945, above 0.883 (best single agent) and 0.894 (prior best). Numbered markers identify method discoveries and their contributing agents. Shaded bands track changes in the team's leading method. See Appendix D.1 for a qualitative analysis of the communication logs.

Polyomino Packing. We now turn to Frontier-CS (Mang et al., 2026) to test whether test-time communication yields stronger performance on challenging algorithmic tasks. We study its polyomino packing task, in which agents must pack a set of pieces into a rectangular box of minimal area. The task is NP-hard, so agents must discover good heuristics and design a packing algorithm rather than search exhaustively. The best known score of 0.894 was obtained by Qu et al. (2026) using four Opus 4.6 agents.

We use Claude Sonnet 4.6 and Opus 4.6 agents under the same communication protocol as in `ARC-AGI-3`. We first evaluate Sonnet 4.6 under Frontier-CS's three-hour limit, with 60 single-agent trials and 20 team@3 trials. We then extend the time limit to 72 hours to study long-horizon performance and run 12 single-agent trials and 2 team@4 trials. For each method, we report the trajectory of the run that achieves the highest score. Our focus is on whether a method can discover a solution beyond the existing frontier, where a single breakthrough matters even when other attempts are unsuccessful (Yuksekgonul et al., 2026).

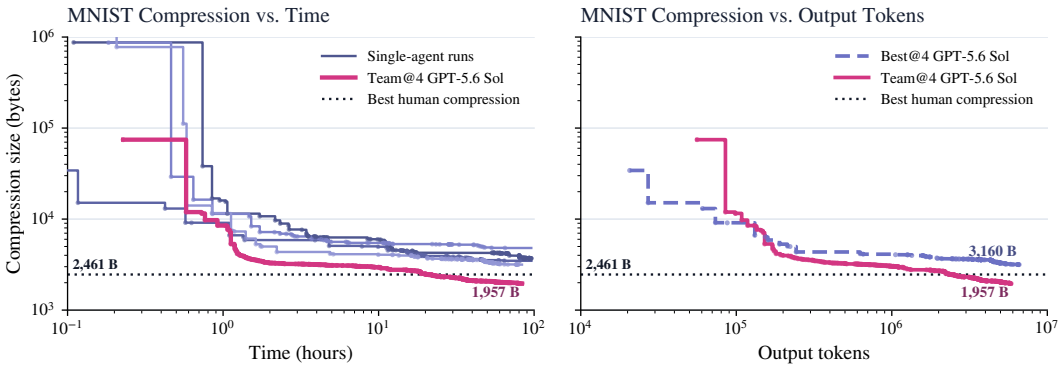


Figure 7: **Communication leads to state-of-the-art ML compression.** Smallest submission with at least 99.4% test accuracy over 96 hours, against wall-clock time and against output tokens. All independent runs converge above the 2,461-byte best-known human solution, while the team reaches 1,957 bytes. See Appendix D.2 for a qualitative analysis of agent communication and the final classifier.

Figure 6 shows that both runs reach roughly 0.82 within the first hour, but the team continues improving as the single agent plateaus. The final gap, 0.945 versus 0.883, more than halves the mean fraction of unused area, and carries a team of Sonnet 4.6 agents past the prior best of 0.894. The team's advantage grows as the single agent's progress slows, resembling the larger gap between team and independent agents at deeper levels of `ARC-AGI-3`.

Figures 9 and 10 in Appendix C.2 show that the gap persists. Independent agents improve slightly and settle at 0.893 and 0.891, still below the prior best. The teams take lasting leads at roughly two and four hours, respectively, and continue to 0.922 and 0.910. (Team scores being lower than 0.945 is likely due to the small number of trials.) Thus, the previous short timeframe does not fully explain the single-agent plateau. These runs demonstrate that communication also outperforms independent agents even in long-horizon settings, as long as communicating agents have enough time and compute to explore and share their discoveries.

Test-time communication breaks through single-agent performance plateaus. For the polyomino packing task, single agents plateau near the prior best while teams continue improving. After an initial coordination cost, progress proceeds as a relay in which different agents introduce and refine the methods that move the frontier forward.

MNIST Classifier Compression. The question is whether communication benefits extend to realistic multi-day engineering and how well they perform relative to the best-known human solutions. In MNIST classifier compression, agents must produce the smallest possible classifier that achieves at least 99.4% test accuracy by minimizing the size of model weights and inference code in 96 hours. The task combines understanding empirical model development with deployment engineering.

In the lower compute regime of Figure 7, independent agents lead. They produce submissions below 15K bytes before the team, which remains near 75K bytes until roughly the first half hour. Team@4 catches the independent frontier after about one hour and 100K output tokens, then stays ahead as further improvements steadily reduce its submission size. Communication therefore incurs an initial cost, but converts later compute into more effective progress.

The difference becomes decisive over the full 96-hour run. The independent runs converge between roughly 3KB and 5KB, with best@4 finishing at 3,160 bytes, and none crosses the

2,461-byte best-known human result. Team@4 crosses that threshold after roughly 20 hours and two million output tokens, then continues to 1,957 bytes, which is about 20% smaller than the best-known human submission. Because the advantage persists when progress is plotted against total output tokens, it cannot be explained by the team merely generating more tokens. Instead, test-time communication allows the four agents' work to be more effective by diversifying their approaches, sharing ideas, and building on each other's progress. While it is certainly possible that a stronger compression could be found given more time or trials, the result is a new state-of-the-art in MNIST classifier compression.

Effective progress over a multi-day horizon and state-of-the-art compression. Team@4 establishes the best known MNIST classifier compression with a 1,957-byte submission that preserves at least 99.4% test accuracy, about 20% smaller than the best-known human solution. The team also surpasses all independent runs.

Considerations for Effective Test-time Communication. Prior work finds that multi-agent systems help when tasks decompose into independent subtasks (Kim et al., 2025). Our tasks do not admit orthogonal decomposition: instead, agents communicate and adopt approaches that build on one another. What prevents agents from adopting bad approaches? We define **verified progress sharing** as sharing discoveries whose value can be objectively measured, so that every agent can judge an approach by its verified result rather than by subjective evaluation (Figure 2). As a result, good approaches spread, bad ones are discarded, and no agent is left exploring a direction that the others have already exhausted.

In a simple pedagogical framework of a task requiring m successive improvements, communication changes a *minimum of sums* into a *sum of minima* (Eq. 5). At matched runtime, team@k success then approaches one exponentially in m, while best@k success approaches zero exponentially (Proposition 1). This demonstrates why communication advantage vanishes when agents lack access to dense verification, as shown in prior negative results. On Terminal-Bench 2.0 (Merrill et al., 2026), where available feedback may not reliably rank approaches, team@2 improves over a single attempt (60.7% vs. 52.5%) , but does not outperform independent pass@2 (62.4%). Appendix E presents the proof, Terminal-Bench 2.0 results, and a discussion of verification and diversity.

Accessible verification helps agents build on shared progress. Communication help agents build upon breakthroughs when they can objectively measure which approaches are worth building on, as long as agents preserve diversity after sharing.

4 Conclusion

In this paper, we studied **test-time communication**, a multi-agent setting in which agents pursue the same open-ended objective and communicate ideas and artifacts through a shared channel of communication. With sufficient compute and verified progress sharing, communication turns individual discoveries into cumulative progress. Across `ARC-AGI-3`, polyomino packing, and MNIST compression, communicating agents match the performance of several times as many independent agents and achieve state-of-the-art results in the latter two, improving upon the best-known human solutions. In fact, this communication multiplier grows as the team scales. These gains emerge only after an initial coordination cost, but test-time communication can outperform independent agents in both performance and compute efficiency in the long run. We leave discussion of future directions in Section F.

AI use statement

In this work, we used generative AI tools to polish writing, search of additional related work, verification of mathematical results, and single- and multi-agent experiment runs. We have also used generative AI tools for drafting implementation detail parts of the paper in the appendix. Generating synthetic datasets with AI is not applicable for us. Additionally, we used generative AI tools for generating code for scientific figures as well as making sure agents in our experiments did not cheat by finding solutions online for `ARC-AGI-3`. We have reviewed all AI-assisted work and read all AI-suggested related work ourselves. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

We provide relevant details for reproducing our experiments in the Appendix. We plan to release the code repository and relevant multi-agent trajectories upon acceptance.

References

Anthropic. Discovering Cryptographic Weaknesses with Claude. Research report, July 2026a. URL https://www.anthropic.com/research/discovering-cryptographic-weaknesses.

Anthropic. Patterns and Problems in Emerging Multiagent Systems. Research report, August 2026b. URL https://www.anthropic.com/research/multiagent-systems.

ARC Prize Foundation. ARC-AGI-3: A New Challenge for Frontier Agentic Intelligence. Technical report, April 2026. URL https://arcprize.org/media/ARC_AGI_3_Technical_Report.pdf.

Juyang Bai and Laixi Shi. MAS-PromptBench: When Does Prompt Optimization Improve Multi-Agent LLM Systems?, 2026. URL https://arxiv.org/abs/2606.23664.

Federico Bianchi, Yongchan Kwon, Aneesh Pappu, and James Zou. Harnessing the collective intelligence of AI agents in the wild for new discoveries. arXiv preprint arXiv:2606.10402, 2026. URL https://arxiv.org/abs/2606.10402.

Kevin J Boudreau and Karim R Lakhani. "open" disclosure of innovations, incentives and follow-on reuse: Theory on processes of cumulative innovation and a field experiment in computational biology. *Research Policy*, 44(1):4–19, 2015.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large Language Monkeys: Scaling Inference Compute with Repeated Sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Justin Chih-Yao Chen, Swarnadeep Saha, and Mohit Bansal. ReConcile: Round-Table Conference Improves Reasoning via Consensus among Diverse LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7066–7085, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.381. URL https://aclanthology.org/2024.acl-long.381/.

Hyeong Kyu Choi, Xiaojin Zhu, and Sharon Li. Debate or Vote: Which Yields Better Decisions in Multi-Agent Large Language Models?, 2025. URL https://arxiv.org/abs/2508.17536.

Igor Costa. AgentSpawn: Adaptive Multi-Agent Collaboration Through Dynamic Spawning for Long-Horizon Code Generation, 2026. URL https://arxiv.org/abs/2602.07072.

Dhairyashil R. G. tiny_MNIST: High accuracy with a very small model. Software repository, 2024. URL https://github.com/dhairyashilRG/tiny_MNIST.

David Donoho. Data science at the singularity. *Harvard Data Science Review*, 6(1), 2024.

Arnaud Doucet and Adam M. Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. In Dan Crisan and Boris Rozovskii (eds.), *The Oxford Handbook of Nonlinear Filtering*, pp. 656–704. Oxford University Press, 2011.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving Factuality and Reasoning in Language Models through Multiagent Debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. From LLM reasoning to autonomous ai agents: A comprehensive review. *IEEE Access*, 2026.

Sudipto Ghosh and Tanmoy Chakraborty. Route, Communicate, and Reason: Gated Routing and Adaptive Depth for Efficient Multi-Agent Reasoning, 2026. URL https://arxiv.org/abs/2607.10836.

GitHub. GitHub Copilot CLI. GitHub Docs, 2026. URL https://docs.github.com/en/copilot/get-started/cli-quickstart.

Neil J. Gordon, David J. Salmond, and Adrian F. M. Smith. Novel approach to nonlinear and non-gaussian bayesian state estimation. *IEE Proceedings F, Radar and Signal Processing*, 140(2):107–113, 1993. doi: 10.1049/ip-f-2.1993.0015.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, Anil Palepu, Keran Rong, Ryutaro Tanno, Khaled Saab, Fan Zhang, Jacob Blum, Andrew Carroll, Kavita Kulkarni, Nenad Tomašev, Dina Zverinski, Ivor Rendulic, Elahe Vedadi, Florian Hasler, Luka Rimanic, Marina Boia, Ivan Budiselic, Ben Feinstein, Mathias Bellaiche, Tom Sheffer, Jan Freyberg, Jeremy Ratcliff, Ottavia Bertolli, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R. D. Costa, José R. Penadés, Gary Peltz, Yossi Matias, James Manyika, Demis Hassabis, Yunhan Xu, Pushmeet Kohli, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Accelerating scientific discovery with Co-Scientist. *Nature*, 655:487–496, 2026. doi: 10.1038/s41586-026-10644-y. URL https://www.nature.com/articles/s41586-026-10644-y.

Zhouhong Gu, Xiaoxuan Zhu, Yin Cai, Hao Shen, Xingzhou Chen, Qingyi Wang, Jialin Li, Xiaoran Shi, Haoran Guo, Wenxuan Huang, Hongwei Feng, Yanghua Xiao, Zheyu Ye, Yao Hu, and Shaosheng Cao. AgentGroupChat-V2: Divide-and-Conquer Is What LLM-Based Multi-Agent System Need, 2025. URL https://arxiv.org/abs/2506.15451.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, 2026. URL https://doi.org/10.5281/zenodo.20953922.

Audrey Huang, Adam Block, Dylan Foster, Dhruv Rohatgi, Cyril Zhang, Max Simchowitz, Jordan Ash, and Akshay Krishnamurthy. Self-improvement in language models: The sharpening mechanism. In *International Conference on Learning Representations*, volume 2025, pp. 76687–76739, 2025.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. Towards a Science of Scaling Agent Systems, 2025. URL https://arxiv.org/abs/2512.08296.

Augustine Kong, Jun S. Liu, and Wing Hung Wong. Sequential imputations and Bayesian missing data problems. *Journal of the American Statistical Association*, 89(425):278–288, 1994. doi: 10.1080/01621459.1994.10476469.

Yann Le Cun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791. URL https://bottou.org/papers/lecun-98h.

Hyunin Lee, Jinglue Xu, Jeffrey Seely, Donghyun Lee, Matei Zaharia, and Yujin Tang. Recursive Harness Self-Improvement, 2026a. URL https://arxiv.org/abs/2607.15524.

Sanghyun Lee, Seungryong Kim, Jongho Park, and Dongmin Park. Lookahead Unmasking Elicits Accurate Decoding in Diffusion Language Models, 2025. URL https://arxiv.org/abs/2511.05563.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-End Optimization of Model Harnesses, 2026b. URL https://arxiv.org/abs/2603.28052.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More Agents Is All You Need. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=bgzUSZ8aeg.

Jun S. Liu and Rong Chen. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443):1032–1044, 1998. doi: 10.1080/01621459.1998.10473765.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A Dynamic LLM-Powered Agent Network for Task-Oriented Agent Collaboration. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=XII0Wp1XA9.

João Loula, Benjamin LeBrun, Li Du, Ben Lipkin, Clemente Pasti, Gabriel Grand, Tianyu Liu, Yahya Emara, Marjorie Freedman, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Alexander K. Lew, Tim Vieira, and Timothy J. O'Donnell. Syntactic and Semantic Control of Large Language Models via Sequential Monte Carlo, 2025. URL https://arxiv.org/abs/2504.13139.

Ziwei Luo, Ziqi Jin, Lei Wang, Lidong Bing, and Thomas B. Schön. Self-Rewarding Sequential Monte Carlo for Masked Diffusion Language Models, 2026. URL https://arxiv.org/abs/2602.01849.

Qiuyang Mang, Wenhao Chai, Zhifei Li, Huanzhi Mao, Shang Zhou, et al. FrontierCS: Evolving Challenges for Evolving Intelligence. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2512.15699. To appear.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*. OpenReview.net, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

Pierre Del Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Springer, New York, 2004. doi: 10.1007/978-1-4684-9393-1.

Stefan Nielsen, Edoardo Cetin, Peter Schwendeman, Qi Sun, Jinglue Xu, and Yujin Tang. Learning to Orchestrate Agents in Natural Language with the Conductor. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=U23A2BUKYt.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. arXiv preprint arXiv:2506.13131, 2025. URL https://arxiv.org/abs/2506.13131.

OpenAI. The Builder's Guide to GPT-5.6. Technical guide, August 2026a. URL https://openai.com/index/builders-guide-to-gpt-5-6/.

OpenAI. On the Navier–Stokes Millennium Prize Problem. Research report, September 2026b. URL https://openai.com/index/navier-stokes-solution/.

Aneesh Pappu, Batu El, Hancheng Cao, Carmelo di Nolfo, Yanchao Sun, Meng Cao, and James Zou. Multi-Agent Teams Hold Experts Back. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Xn8kmKvO9g.

Michael Polanyi. The republic of science: Its political and economic theory. *Minerva*, 1(1): 54–73, 1962.

Isha Puri, Shivchander Sudalairaj, Guangxuan Xu, Kai Xu, and Akash Srivastava. Rollout Roulette: A Probabilistic Inference Approach to Inference-Time Scaling of LLMs Using Particle-Based Monte Carlo Methods, 2025. URL https://arxiv.org/abs/2502.01618.

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, Jiacheng Zhu, Xuan Jiang, Sirui Li, Cathy Wu, Bryan Kian Hsiang Low, Jinhua Zhao, and Paul Pu Liang. CORAL: Towards

Autonomous Multi-Agent Evolution for Open-Ended Discovery. In *Conference on Language Modeling*, 2026. URL https://arxiv.org/abs/2604.01658.

Shuai Shao, Yixiang Liu, Bingwei Lu, and Weinan Zhang. MonoScale: Scaling Multi-Agent System with Monotonic Improvement. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=bwqwuJM68F.

John F. Shortle, James M. Thompson, Donald Gross, and Carl M. Harris. *Fundamentals of Queueing Theory*. John Wiley & Sons, 5 edition, 2018. ISBN 9781118943526.

Raghav Singhal, Zachary Horvitz, Ryan Teehan, Mengye Ren, Zhou Yu, Kathleen McKeown, and Rajesh Ranganath. A General Framework for Inference-Time Scaling and Steering of Diffusion Models, 2025. URL https://arxiv.org/abs/2501.06848.

Andries Petrus Smit, Nathan Grinsztajn, Paul Duckworth, Thomas D. Barrett, and Arnu Pretorius. Should We Be Going MAD? A Look at Multi-Agent Debate Strategies for LLMs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 45883–45905. PMLR, 2024. URL https://proceedings.mlr.press/v235/smit24a.html.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally Can Be More Effective than Scaling Parameters for Reasoning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b623663fd9b874366f3ce019fdfdd44-Abstract-Conference.html.

Chris Snyder, Thomas Bengtsson, Peter Bickel, and Jeff Anderson. Obstacles to high-dimensional particle filtering. *Monthly Weather Review*, 136(12):4629–4640, 2008. doi: 10.1175/2008MWR2529.1.

Michael Strevens. *The Knowledge Machine: How Irrationality Created Modern Science*. Liveright, New York, 2020.

Yujin Tang, Edoardo Cetin, Jinglue Xu, Qi Sun, Stefan Nielsen, Vincent Richard, Haruto Goda, Iaroslav Tymchenko, Nhan Nguyen, Hyunin Lee, Mari Ashiga, Shashank Kotyan, So Kuroki, and Tarin Clanuwat. Sakana Fugu Technical Report, 2026. URL https://arxiv.org/abs/2606.21228.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. Multi-agent collaboration mechanisms: A survey of LLMs. *arXiv preprint arXiv:2501.06322*, 2025.

Xiao Wang, Jia Wang, Yijie Wang, Pengtao Dang, Sha Cao, and Chi Zhang. MARS: Toward More Efficient Multi-Agent Collaboration for LLM Reasoning, 2025. URL https://arxiv.org/abs/2509.20502.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, Hao Cheng, Pengcheng He, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. ThetaEvolve: Test-time learning on open problems. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=XHP568OuIl.

Haolun Wu, Zhenkun Li, and Lingyao Li. Can LLM Agents Really Debate? A Controlled Study of Multi-Agent Debate in Logical Reasoning, 2025. URL https://arxiv.org/abs/2511.07784.

Jinglue Xu, Qi Sun, Peter Schwendeman, Stefan Nielsen, Edoardo Cetin, and Yujin Tang. TRINITY: An Evolved LLM Coordinator. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=5HaRjXai12.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist-v2: Workshop-Level Automated Scientific Discovery via Agentic Tree Search, 2025. URL https://arxiv.org/abs/2504.08066.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time, 2026. URL https://openreview.net/forum?id=96zNuQrH9Y.

Sukwon Yun, Jie Peng, Pingzhi Li, Wendong Fan, Jie Chen, James Zou, Guohao Li, and Tianlong Chen. Graph-of-Agents: A Graph-Based Framework for Multi-Agent LLM Collaboration. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=34cANdsHKV.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive Language Models, 2025a. URL https://arxiv.org/abs/2512.24601.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-Agent Architecture Search via Agentic Supernet. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 75834–75852. PMLR, 2025b. URL https://proceedings.mlr.press/v267/zhang25bi.html.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-Designer: Architecting Multi-Agent Communication Topologies via Graph Neural Networks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76678–76692. PMLR, 2025c. URL https://proceedings.mlr.press/v267/zhang25cu.html.

Hangfan Zhang, Zhiyao Cui, Jianhao Chen, Xinrun Wang, Qiaosheng Zhang, Zhen Wang, Dinghao Wu, and Shuyue Hu. Stop Overvaluing Multi-Agent Debate—We Must Rethink Evaluation and Embrace Model Heterogeneity, 2025d. URL https://arxiv.org/abs/2502.08788.

Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Grosse. Probabilistic Inference in Language Models via Twisted Sequential Monte Carlo, 2024. URL https://arxiv.org/abs/2404.17546.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language Agents as Optimizable Graphs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

A Related Work

Multi-agent systems. Multi-agent systems (MAS) are becoming increasingly popular and widely adopted. As discussed in the introduction, some prior work (Du et al., 2024; Chen et al., 2024) reports gains from multi-agent debate and belief exchange. However, deliberation without a verifier or external feedback can amplify correlated errors or move the team away from its strongest member (Smit et al., 2024; Pappu et al., 2026). In fact, independent sampling followed by aggregation remains a strong baseline (Wang et al., 2023; Li et al., 2024; Choi et al., 2025). Benefits also depend on task structure and model strength. Kim et al. (2025) find stronger gains on decomposable tasks, but these gains diminish as the underlying model becomes stronger with respect to the task. While Kim et al. (2025) comprehensively cover a range of tasks and configurations, our work studies the benefits of MAS in a more open-ended discovery setting with a clear verifier signal. As discussed in Section 3, it is *verified progress sharing* that allows us to avoid "agent collapse" that previous works have observed. For more detailed surveys of MAS, we refer to Ferrag et al. (2026); Tran et al. (2025).

Test-time discovery. Test-time discovery often uses LLMs (or agents) to generate, evaluate, and iteratively improve candidate solutions, rather than producing an answer in a single attempt from a pretrained model. Within this paradigm, Novikov et al. (2025) keep the underlying LLMs frozen and accumulate progress through evolutionary program search, evaluator feedback, and a database of generated attempts. Wang et al. (2026); Yuksekgonul et al. (2026), by contrast, train the model via reinforcement learning during discovery, updating the LLM using feedback obtained from the target problems. Recent work also focuses on custom agentic workflows, where agents are assigned specialized roles to generate, critique, and refine hypotheses, often orchestrated by a central agent (Gottweis et al., 2026; Yamada et al., 2025). Our setting, however, is orthogonal to the works above as they focus on either single-agent capabilities or multi-agent workflows without establishing an advantage over single-agent systems under matched compute.

Recent work illustrates the potential of multi-agent communication for discovery. Anthropic (2026a) describe an improved attack on HAWK whose key idea emerged through an exchange between two agents, while OpenAI (2026b) report a Navier–Stokes result produced by a group of roughly 10,000 concurrent agents. Similarly, Bianchi et al. (2026) explore mathematical discoveries by an open community of heterogeneous agents sharing solutions and encouraging public discussion. These are systems in which agents choose how to explore and communicate without a fixed workflow.

These findings motivate controlled tests of whether communication improves on independent search, especially under matched compute. Anthropic (2026b) report more vulnerabilities from coordinating agents than from independent agents, though Claude Mythos Preview uses comparable tokens per vulnerability when both methods are evaluated on the same search scope. On other tasks, they observe failures to integrate agents' work, conformity, and failures to share decisive evidence. Closely related to our setup is Qu et al. (2026), whose agents share discoveries through persistent memory and outperform the best of four independent runs under matched wall-clock budgets. In contrast, we diagnose clear conditions for when test-time communication is advantageous, through compute-controlled studies of scaling coordinating agents on `ARC-AGI-3` and long-horizon algorithm/ML tasks, including negative results at low-compute budgets and on Terminal-Bench 2.0.

Harness optimization and orchestration. One line of work aims to learn the ideal topology and communication protocol of MAS. Agent graphs and communication links can be optimized directly (Zhuge et al., 2024; Zhang et al., 2025c), while architecture search can

adapt the system to the task (Zhang et al., 2025b; Yun et al., 2026). A second line controls when and which agents participate. Systems can select teams or gate communication (Liu et al., 2024; Ghosh & Chakraborty, 2026), regulate interaction to avoid harmful scaling (Wang et al., 2025; Shao et al., 2026), or add agents dynamically and decompose work into explicit subtasks (Costa, 2026; Gu et al., 2025). Gu et al. (2025) investigate agent diversity and observe weaker returns from homogeneous teams, though heterogeneity may bring down the performance of the best model in the team even when explicitly told who the best is (Pappu et al., 2026).

Hierarchical systems place these decisions with a central coordinator. Coordinators can be trained to assign work to frozen agents (Xu et al., 2026; Nielsen et al., 2026), while adaptive hierarchies can delegate work and synthesize results, primarily for parallel decomposition (Tang et al., 2026; OpenAI, 2026a). Recursive delegation has also been used for long-context inference rather than multi-agent orchestration (Zhang et al., 2025a). Harness optimization is broader and need not involve multiple agents. Executable or prompt-level agent loops can be optimized from scores, traces, or revision feedback (Lee et al., 2026b;a). Prompt optimization has also been studied for multi-agent protocols (Bai & Shi, 2026).

Unlike these approaches, we do not study orchestration or multi-agent harness optimization. Instead, we use a fixed shared-workspace scaffold of identical, unassigned peers and compare its performance with compute-matched independent runs to isolate communication from parallel exploration.

Agent coordination as interactive systems. We view sequential Monte Carlo as an interacting-population counterpart to best-of-k, with partial solutions repeatedly scored and resampled (Gordon et al., 1993; Liu & Chen, 1998; Moral, 2004). Populations degenerate over long horizons (Kong et al., 1994; Doucet & Johansen, 2011), while resampling concentrates shared ancestors, and harder problems require larger populations (Liu & Chen, 1998; Snyder et al., 2008). These phenomena correspond to our gains at depth, herding failures, and non-monotone returns to team size.

Related methods now guide language model inference. Particle methods support probabilistic inference and test-time scaling (Zhao et al., 2024; Puri et al., 2025), as well as constrained generation (Loula et al., 2025). Interacting particles have also been adapted to diffusion models (Singhal et al., 2025; Luo et al., 2026), with related path selection for diffusion language model decoding (Lee et al., 2025). The analogy is conceptual and suggests that coordination is most useful over long horizons with informative intermediate feedback and a diverse population.

B Full Experimental Setup

This section provides the communication-prompt and implementation details deferred from Section 2, followed by task-specific evaluation details for the experiments reported in the main text.

B.1 Full Benchmark Descriptions

We study multi-agent communication on the following three tasks, each covering a different aspect of open-ended research tasks.

ARC-AGI-3 (ARC Prize Foundation, 2026) tests novel problem solving and interactive discovery by asking agents to solve unfamiliar grid-world games without instructions, inferring the rules, the goal, and the effect of each control from play alone. Each game comprises

Table 2: **Summary table.** We evaluate multi-agent communication and compare the best of k independent agents with a team of k communicating agents on three tasks. All agents are run using the GitHub Copilot CLI and communication harness we describe in Section 2.

Benchmark	ARC-AGI-3	Polyomino packing	MNIST Compression
Model	Sonnet 4.6	Sonnet, Opus 4.6	GPT-5.6 Sol
Budget per agent	per-level action budget	3-72 hours	96 hours
Team size k	3, 5	3, 4	4
Metric	game solve rate	packing score	qualified bytes
Prior best-known result	–	0.894	2,461 B
Independent best@k	1.4% (k=3), 2.2% (k=5)	0.893	3,160 B
Communicating team@k	**4.6%** (k=3), **8.0%** (k=5)	**0.945**	**1,957 B**

$l_{\max} \in [6, 10]$ levels, and progress depends on carrying forward what was learned in earlier ones. This leveled structure makes breakthroughs cleanly measurable. Real-world research problems may not offer such a clean signal. A genuine conceptual advance in a domain such as approximation factors for NP-hard problems may move the reported number by only a small constant, an improvement easily lost in run-to-run variance.

We evaluate on all 25 public games with Claude Sonnet 4.6 under the benchmark's native per-level action budget and measure solve rate as the proportion of trials that clear all levels successfully. Though more recent models such as GPT-6 Astra now succeed easily on the benchmark, we focus on one model, especially one that does not saturate, to isolate the effect of test-time communication. For teams, an agent that exhausts its action budget is terminated, while the remaining agents continue. We run 64 single-agent trials per game and label a game unsolved by single agents if no trial clears all levels. For teams of three and five agents, we run 20 trials for every combination of game and team size.

Frontier-CS polyomino packing (Mang et al., 2026) tests algorithmic optimization on an NP-hard problem. Agents write and repeatedly improve a C++17 program that packs reflected and rotated polyominoes into a minimum-area rectangle. The scorer evaluates the submitted solution on 70 hidden test cases and returns continuous partial credit in $[0, 1]$. The *packing score* is the mean reward across the 70 cases, and the highest valid submission in a run is retained. The best published score is 0.894, obtained by Qu et al. (2026) using four Claude Opus 4.6 agents. We test on both Claude Sonnet 4.6 and Opus 4.6 with teams of three or four agents.

MNIST Classifier Compression tests empirical ML research. Agents must train and compress a self-contained MNIST classifier (Le Cun et al., 1998). An artifact qualifies only at 99.4% test accuracy or better and agents cannot inspect the test images, labels, or individual errors, as described in Appendix B.6. Among qualifying artifacts, only compressed size counts. We measure size using a deterministic gzip-9 compression of the submission, which includes inference code and model weights (they may be separate files or one file, depending on the agent's design). To the best of our knowledge, the best prior result comes from the open-source model introduced by Dhairyashil R. G. (2024), which we reproduce as a 2,461-byte submission under our artifact format. We test a team of four GPT-5.6 Sol agents over a 96-hour period. Further implementation details can be found in Appendix B.

Runtime Environment. Every agent operates through GitHub Copilot CLI (GitHub, 2026), with each trial isolated as a Harbor task (Harbor Framework Team, 2026). Single-agent trials contain one CLI agent. Team@k trials contain k agents in the same task container. Agents receive the same task, tools, and base filesystem, but have private scratch directories.

Within a team, agents communicate explicitly. They append timestamped claims, evidence, failures, and adoption events to shared logs, publish measured candidates to a board, and use file locks to serialize changes to a shared graded artifact. Distinct trials share neither files nor messages. Hidden evaluation data are kept outside the agent container. Polyomino submissions are sent to a separate scorer service, while MNIST submissions are evaluated by a host-owned oracle in a fresh network-disabled container. These services return evaluation feedback but never expose the underlying cases, labels, or retained program outputs.

Metrics. Our central comparison is the outcome of k communicating agents (**team@**k) versus the best outcome among k independent agents (**best@**k). We measure both metrics against wall-clock time and against total output tokens to account for compute. For `ARC-AGI-3`, a level counts as solved if any of the k agents clears it, and a game counts as solved if any of the k agents clears all levels. For a finite ARC solo pool with s successes among n trials, we compute best@k exactly, without replacement, as best@$k = 1 - \binom{n-s}{k} / \binom{n}{k}$. We apply the same calculation level by level and average over games, giving each game equal weight. Throughout the paper, solve rate refers to the final solve rate averaged over all 25 games, and per-level rates are labeled explicitly. We measure team@k and best@k as the best-so-far packing score (mean packed-cell density across 70 cases) or MNIST classifier compression size (gzipped submission size including code and weights).

Every agent operates through GitHub Copilot CLI (GitHub, 2026), with each trial isolated as a Harbor task (Harbor Framework Team, 2026), and hidden evaluation data are kept outside the agent container. Within every comparison, communicating and independent agents use the same model, maximum reasoning effort, longest context-length setting, and per-agent resource allocation.

B.2 Communication prompts, runtime, and accounting

Communication-prompt summary. Beyond the shared-workspace instructions in Section 2, the prompt specifies when agents may adopt a peer's approach. Adoption requires a measured improvement, replication, an actionable alternative to a blocked approach, or final convergence. After adoption, agents are instructed to preserve a meaningful variation and record what they adopted and why.

For MNIST classifier compression, agents build and validate candidates in scratch space, then recheck the current best under the lock before promoting a strictly better submission. These are instructions to the agents, rather than automatic verification or adoption by the harness.

Runtime versions. The ARC-AGI-3 and Terminal-Bench 2.0 experiments use GitHub Copilot CLI 1.0.54, the long-horizon packing experiments use version 1.0.70, and MNIST classifier compression uses version 1.0.78.

Shared and per-agent resources. CPU and host-memory limits apply to the entire task container. All workers in a team share those limits, rather than each receiving a separate container-sized allocation. Independent trials each have their own container. The configured allocations are shown below.

The ARC and packing allocations do not scale with team size. MNIST scales the container's CPU and host-memory limits by four for team@4, but does not enforce separate per-worker host-memory partitions. Its CPU thread pools are set to three threads per worker. Each agent is instructed to use at most 1,000 MiB of aggregate GPU memory across all its processes. A 400-MiB PyTorch allocator setting leaves headroom for CUDA context and

Table 3: **CPU/GPU and container setup for each task.**

Task	Solo container	Team container	Task GPU
ARC-AGI-3	4 CPUs, 8 GiB	4 CPUs, 8 GiB	None
Polyomino packing	2 CPUs, 6 GiB	2 CPUs, 6 GiB	None
MNIST compression	3 CPUs, 8 GiB	12 CPUs, 32 GiB ($k = 4$)	A100
Terminal-Bench 2.0	Task-defined	Task-defined, shared	Task-defined

library memory, but is not a hard limit on total GPU memory. The per-agent GPU-memory rule is self-enforced. These task-compute resources are distinct from model inference through the Copilot service and from ARC's separate per-agent action budgets.

B.3 Agent Communication Prompt

Every agent in a team receives the prompt below, in addition to the task instruction. The text is stored as a template, and before a run starts the harness substitutes the team size into `{n}` and `{nm1}` and the shared-scratch paths into the remaining braced fields. Every agent in a trial then receives the identical filled-in text, so the only asymmetry between agents comes from the slot each one wins at runtime.

```
{n} agents share this container and work the same task in parallel, all with this
identical prompt. Search widely without herding, coordinate as you go, and keep
improving until time runs out.

Shared scratch (create on first use):
  - slots/approaches: {slots}/
  - findings:         {findings}
  - disconfirmations: {disconfirm}
  - score log:        {plateau}
  - coordination:     {coordination}   (empty -- conventions you author)
Your private scratch is {base}/work-$S after you claim slot $S.

1. CLAIM A SLOT AND PICK A DISTINCT APPROACH:
     mkdir -p {slots}
     for i in $(seq 0 {nm1}); do mkdir "{slots}/slot-$i" 2>/dev/null && S=$i && break; done
     mkdir -p {base}/work-$S
     touch {coordination}
     echo "<your approach + what you will deliberately not assume>" > "{slots}/slot-$S/approach"
     cat {slots}/slot-*/approach
   If a lower-numbered slot already took your approach, change yours. Cover a
   different part of the search space; do not agree early.

2. ALWAYS BE ACTING. Never end a turn with only prose or a plan. Every turn must
   run a command that advances or tests the work: take a real action the task
   accepts and read back its result. Publishing, disconfirming, coordinating, and
   pivoting are bookkeeping around real actions, never a substitute for taking one.

3. EVALUATE AND RECORD EVERY ATTEMPT. Score each change with the task's own
   scoring or feedback mechanism (find it in AGENT.md or the task instructions),
   then log the result:
     echo "[slot $S $(date -u +%H:%M:%S)] score=<score-or-progress> family=<approach>" >> {plateau}

4. SHARE WITHOUT HERDING. Append concise findings with evidence and cost to
   {findings}, and label weak claims as weak. Spend part of your effort trying to
   FALSIFY the leading idea or your own, recording negative results in
   {disconfirm}. Do not write prose whose only purpose is to make peers copy you.

5. COORDINATE ON SHARED RESOURCES. You share the graded output, common files, and
   the environment with peers who run blind to your session. Treat every shared
   resource as contested: before you touch one, re-check {coordination} and the
   resource's current state; after, confirm your change survived and did not just
   repeat a peer's. A collision is any overwritten, duplicated, or conflicting work
   that wastes effort. No coordination scheme is provided -- {coordination} is empty
```

```
   and yours to author: on a collision, write a convention there concrete enough for
   a peer to follow, that changes your next action, then follow it.

6. HIGH BAR TO CONVERGE. Keep your own approach unless another clearly beats it on
   a measured, reproduced result, or yours is blocked, or the run is wrapping up.
   Even then, keep one real difference (a parameter, subcase, representation, or
   fallback) until the very end.

7. NEVER STOP WHILE TIME REMAINS. A working result is not the finish line; the
   clock running out is the only acceptable reason to stop. Do not declare the
   task done, final, solved, or "at the ceiling" and go idle -- a suspected
   ceiling is a claim to disconfirm, not a reason to quit.

8. BREAK PLATEAUS BY CHANGING FAMILY. You are plateaued when your best score has
   not strictly improved over 3 consecutive attempts. Then stop tuning
   and switch to a STRUCTURALLY DIFFERENT approach -- a different core principle or
   assumption, not a variant of the current one. Keep a short list of untried
   families in {base}/work-$S so you always have a next one ready. Read peers'
   approaches and {plateau} first and pick a family no active peer is on; adopting
   a peer who is also plateaued is not progress.

Run a tight loop -- change -> evaluate -> record -> repeat -- without pausing. Do
not use the internet, curl, wget, HTTP libraries, or secrets.
```

B.4 ARC-AGI-3

Task and interface. Agents interact with the games by receiving integer-valued grids and selecting from the available discrete controls or coordinate clicks. Each level has its own cap, so unused budget from a later level cannot rescue an agent stalled earlier. We use the benchmark-native cap of five times the corresponding human-action baseline, independently for every agent.

Every team member owns a separately authenticated ARC session. Sharing an action trace does not copy state and does not clear a peer's level. Time-dependent progress curves use the first logged completion of each level.

Synchronization. Team runs synchronize at intervals of half the current level's action budget, rounded up to an integer number of actions. The harness tracks each agent's phase by its level and the number of these intervals it has used. An agent ahead of the slowest live teammate, either within a level or by reaching the next level, cannot take another game action until its teammates catch up or leave the live set. During this pause, agents can still inspect state, run shell commands, and exchange notes. Refused game actions consume no budget. Finished, game-over, and budget-exhausted agents do not block their teammates, and inactive sessions are excluded after an idle timeout. This synchronization applies only to communicating teams, not to best@k.

Closed-book condition. The task container requires network egress for the model control plane, but agents receive no browser or web-search tool and are explicitly forbidden to fetch public solutions, replays, or ARC pages. Only the task prompt, local run files, the agent's own environment observations, and teammates' within-trial notes are admissible. We inspect run trajectories afterwards to verify that no agent accessed external ARC information.

B.5 Frontier-CS polyomino packing

Objective and scorer. Each of 70 hidden cases contains $n \in [100, 10^4]$ edge-connected polyominoes of one to ten cells. A program may reflect, rotate by multiples of 90°, and translate each piece. All pieces must lie without overlap in one integer-grid rectangle. If C_j is the total number of occupied cells and $A_j = W_j H_j$ the returned area, the normalized

case quality is C_j/A_j and the reported score is its mean over the fixed cases. Invalid output causes the submission to be rejected. Candidate programs compile as GNU C++17 and receive 2 seconds and 256 MiB per case.

The hidden instance directory is mounted read-only inside a separate scorer service and is never mounted into the agent container, which contains the task statement and a submission client. On each attempt, the client sends the candidate C++ source over an internal endpoint. The scorer compiles and executes it, with case-output retention disabled, and returns the aggregate score together with compact status, timing, and scoring metadata. The scorer never returns the instances or program outputs. Agents may submit repeatedly without penalty, but cannot inspect the evaluation data. Every attempt, timestamp, status, and score is recorded in a submission ledger. The final verifier retains the highest valid scored submission in the run, protecting an earlier champion from a broken final edit.

B.6 MNIST classifier compression

Data and artifact contract. The official 60,000-image MNIST training split is partitioned once into 55,000 training and 5,000 development images. The agent image contains only these two partitions, while the official 10,000-image test archive remains on the host and is never copied into the task container. Agents may train only on the 55,000 images, use the development set for selection, and query the sealed oracle only after a full-development accuracy of at least 99.4%.

An oracle request snapshots the canonical submission and passes that snapshot to a host-owned evaluator. Each request runs in a fresh container with networking disabled, a read-only root filesystem, privilege escalation disabled, and the submission mounted read-only. Within it, a trusted verifier privately shuffles the test set, makes the test archive and labels unreadable to submitted code, and invokes inference under an unprivileged user. The submission receives only batches of test images and writes predictions to isolated scratch space. The trusted verifier alone reads the labels and computes accuracy. The container is discarded after evaluation, and the oracle returns only aggregate accuracy. It never returns examples, labels, predictions, or per-example errors. Final grading uses the same isolation boundary, with a 600-second limit for classifying the complete test set.

The graded directory must include a `predict.py` entry point and every weight, table, constant, generator, and decoder it needs. We normalize file order and metadata, form a deterministic tar archive, and apply gzip level 9. Training code outside the submission and the preinstalled numerical runtime are not charged.

Open-source reference. The 2,461-byte reference in Section 2 is our benchmark-format reproduction of the open-source `tiny_MNIST` model (Dhairyashil R. G., 2024). The published architecture has 3,130 trainable parameters and reports accuracy above 99.4%. We retrained it from the published recipe, folded BatchNorm exactly into the adjacent convolutions, and quantized the resulting weights per output channel to four bits.

For the deployable artifact, we bit-pack this state, use a compact decoder, and for fair comparison, code-golf the inference path by eliminating general training machinery, redundant metadata, whitespace, and intermediate file structure while preserving predictions. The complete normalized archive, including executable inference code, is 2,461 bytes under the same deterministic gzip-9 metric used for every submission. The reproduced artifact attains exactly 99.40% on the sealed test set. Thus 2,461 bytes is a measured end-to-end submission size, not the size of the upstream training checkpoint.

Environment. The pinned stack is Python 3.12.13, PyTorch 2.5.1 with CUDA 12.4, torchvision 0.20.1, NumPy 2.5.1, SciPy 1.18.0, and scikit-learn 1.9.0. Agents train from scratch on the provided data. Downloading additional data or pretrained weights is prohibited, and inference evaluation has no network access.

Checkpoints and trajectory accounting. Agents are instructed to save each smaller submission that passes the full development-set check, together with its result, timestamp, provenance, and recomputed archive size. Saved checkpoints are audited, and the paper figures use only submissions confirmed at $\geq 99.4\%$ on the sealed test set. Figure 7 compares four independent GPT-5.6 Sol runs with one team@4 run. In the elapsed-time panel, best@4 at time t is the smallest qualifying submission found by any of the four independent agents within its own first t hours. The output-token panel uses the same best-so-far trajectory, placing each improvement at the sum of tokens consumed by all four independent runs by that elapsed time. Team tokens likewise sum all four workers. Neither curve counts only the tokens of the agent that produced an improvement.

B.7 Terminal-Bench 2.0

Task environments and execution. The harness loads `terminal-bench@2.0` through Harbor and runs each task in its Docker environment. CPU, host-memory, storage, and GPU requests come from the individual task configuration, rather than a uniform benchmark-wide allocation. The harness does not automatically multiply these requests by the number of workers. In a team@2 trial, both workers share the task container and its resource limits. Agent execution and verifier timeouts are configured separately by each task, so there is no single wall-clock horizon analogous to the packing or MNIST experiments. The verifier runs after agent execution and produces the trial's reward from the resulting environment.

Aggregation. For Table 4, pass@1 averages the four independent outcomes for each task. The saved independent results are grouped into two batches of two attempts. Pass@2 counts a task as successful within a batch if either attempt passes, then averages the two batch accuracies. Team@2 averages the two team-trial outcomes for each task. Unfinished tasks are assigned zero, and all 89 tasks remain in the denominator. Max Accuracy selects the higher batch or team-trial accuracy over the whole benchmark, rather than selecting a different batch or team trial for each task.

C Full Experimental Results

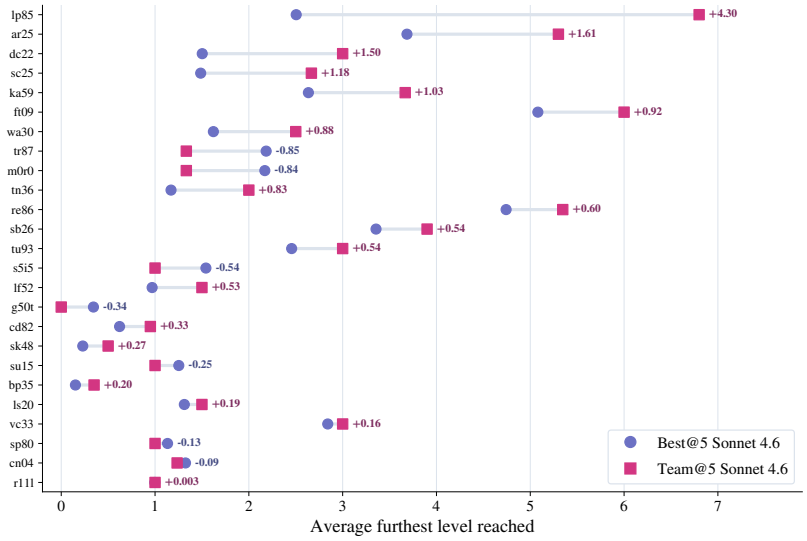
In this section, we present experiments and remaining results that were only briefly mentioned in Section 3.

C.1 ARC-AGI-3

Relative Human Action Efficiency. An action is a discrete interaction that changes the game state, so reasoning, tool calls, and read-only inspection are not charged. For level ℓ of game e, let $a_{\ell,e}$ be the actions the agent spends and $h_{\ell,e}$ the human baseline, taken as the upper-median best action count over first-time human players. The level score is

$$S_{\ell,e} = \min\left\{1.15,\ \left(\frac{h_{\ell,e}}{a_{\ell,e}}\right)^2\right\}, \tag{1}$$

(a) Communication effect in ARC-AGI-3



(b) Final solve rate

Game	best@5	team@5
LP85	0.0	**65.0**
AR25	7.8	**20.0**
DC22	0.0	0.0
SC25	0.0	0.0
KA59	0.0	0.0
FT09	39.9	**100.0**
WA30	0.0	0.0
TR87	0.0	0.0
M0R0	0.0	0.0
TN36	0.0	0.0
RE86	0.0	0.0
SB26	7.8	**15.0**
TU93	0.0	0.0
S5I5	0.0	0.0
LF52	0.0	0.0
G50T	0.0	0.0
CD82	0.0	0.0
SK48	0.0	0.0
SU15	0.0	0.0
BP35	0.0	0.0
LS20	0.0	0.0
VC33	0.0	0.0
SP80	0.0	0.0
CN04	0.0	0.0
R11L	0.0	0.0

Figure 8: **Net communication effect on each ARC-AGI-3 game. (a)** Average furthest level reached by team@5 and best@5, sorted by absolute difference. Communication helps on 18 of the 25 games and hurts on 7, and the best gain is +4.30 levels on LP85 while no loss exceeds 0.85. **(b)** Success rate across 64 single-agent trials and 20 team@5 trials. The aggregate gain comes from a handful of games, as most remain out of reach for Sonnet 4.6. Nevertheless, where communication helps, it helps substantially. Across the four games with any team@5 success, solve rate rises from 13.8% to 50.0% on average. Most strikingly, LP85 goes from no single-agent success to 65%.

with $S_{\ell,e} = 0$ for any level the agent never completes. Game scores weight level ℓ by $w_\ell = \ell$ and are capped by the weighted fraction of levels completed,

$$E_e = \min\left\{ \frac{\sum_{\ell=1}^{d} w_\ell}{\sum_{\ell=1}^{n} w_\ell}, \; \frac{\sum_{\ell=1}^{n} w_\ell \, S_{\ell,e}}{\sum_{\ell=1}^{n} w_\ell} \right\}, \tag{2}$$

where d of n levels are completed, and the reported RHAE is the mean of E_e over games.

C.2 Frontier-CS: Polyomino Packing

We present the team@4 figures for Sonnet 4.6 and Opus 4.6, which we could not include in the main body due to space restriction. The discussion of these results can be found in Section 3.

D Qualitative Study on Communication

D.1 Case Study: Polyomino Packing

In this subsection, we examine the successful three-hour Sonnet run that set a new state-of-the-art score of 0.945 to qualitatively understand and assess how good communication leads to breakthroughs. With the help of Opus 5, we analyze the run's traces and identify the key

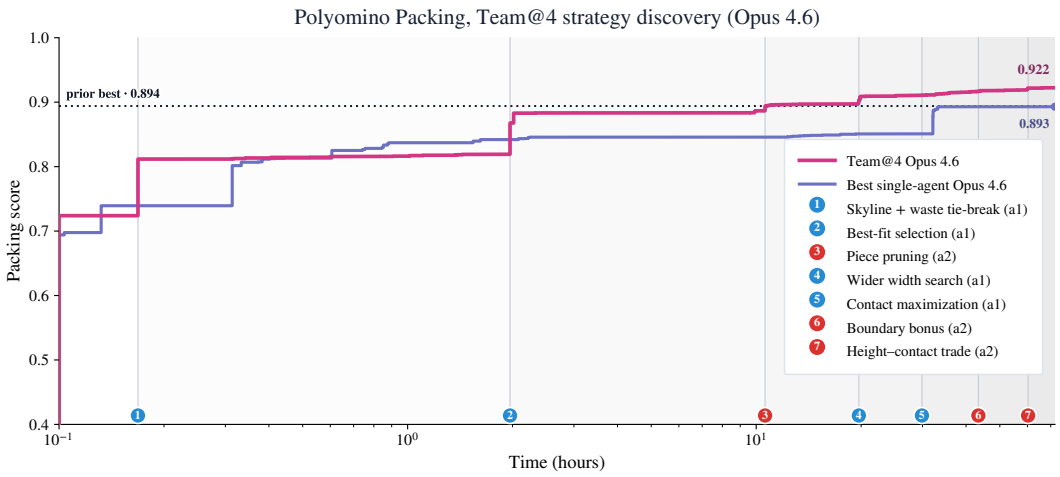


Figure 9: **The Opus 4.6 team sustains its advantage over a longer horizon.** Best-so-far packing score for the highest-scoring run among 12 single-agent trials and two team@4 trials, each allowed up to 72 hours. Time is shown on a logarithmic axis. The team takes a lasting lead at about two hours and reaches 0.922, while the single agent plateaus at 0.893, just below the dotted prior-best reference of 0.894. Markers and shading follow Figure 6.

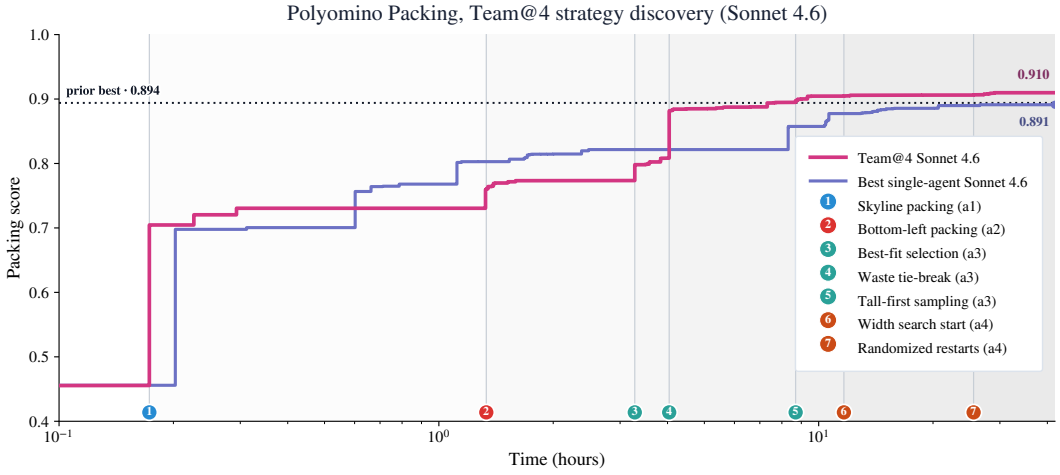


Figure 10: **The longer Sonnet 4.6 team run also clears the single-agent plateau.** Best-so-far packing score for the highest-scoring run among 12 single-agent trials and two team@4 trials, each allowed up to 72 hours, with the selected team trajectory ending at roughly 42 hours. Time is shown on a logarithmic axis. The team takes a lasting lead at roughly four hours and reaches 0.910, while the single agent finishes at 0.891. The dotted line marks the prior best of 0.894. Markers and shading follow Figure 6.

discoveries that led to the final score. We follow the numbered discoveries in Figure 6, using the agent labels shown in the figure.

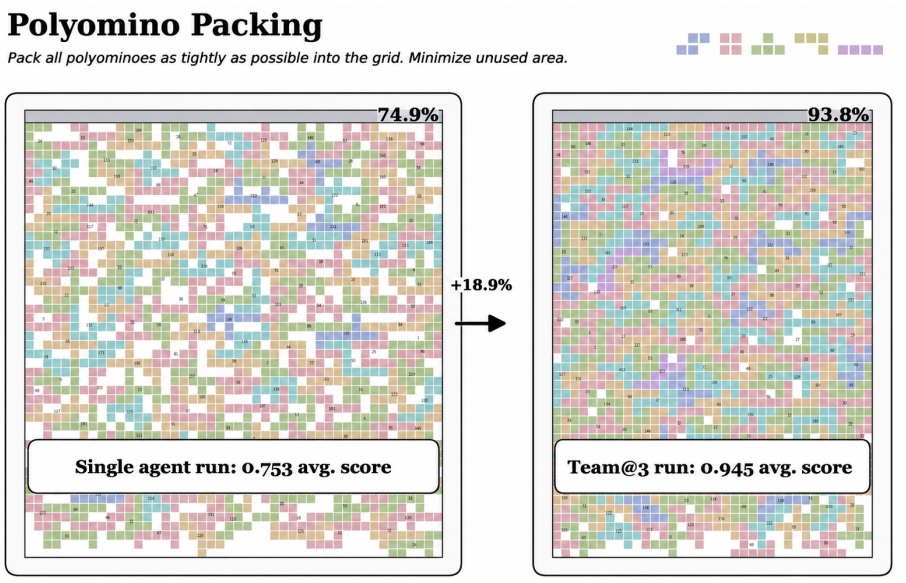


Figure 11: **Communication yields a denser polyomino packing.** The single-agent solution and the best Sonnet team@3 solution pack the same 172 pieces (1,654 cells) at 74.9% and 93.8% density, respectively, a gain of 18.9 percentage points. The labels 0.753 and 0.945 are the runs' benchmark-average scores.

From compact placement to filling gaps. Agent a1 begins with *shelf packing*, arranging pieces in rows, then *skyline packing*, placing them along the upper contour of the occupied region. Agent a2's *bottom-left packing* searches for the lowest legal placement, breaking ties toward the left. These methods successively lead the run at markers 1–3 in Figure 6, but stall near 0.80. Keeping placements low still leaves enclosed gaps that later pieces cannot fill. Agent a1 identifies this limitation and proposes *contact maximization*, favoring placements that touch more already-occupied cells. Contact serves as a local measure of fit, encouraging pieces to nest together rather than merely keeping the current height small.

Building upon another agent's failed idea. The proposal for contact maximization appears at marker 4, but a1's first implementation is slow and the score barely changes for roughly half an hour. Agent a2 then rebuilds the method from a1's message in the communication logs without reading its code, describing the rule as counting "how many adjacent cells are already occupied for each piece that fits" and choosing the highest-contact candidate. This independent implementation and continued efficiency improvements raise the score to nearly 0.87 after marker 5. Agent a2 then evaluates every piece orientation within the placement lookahead, comparing how different orientations fit against the existing packing. Running this *all-orientation search* at full strength produces the decisive jump past 0.90 at marker 6.

The difficulty is making that search affordable within the evaluator's two-second limit per instance. Examining more orientations improves placement quality but risks a timeout, while conservative search budgets leave useful candidates unexplored. The original traces show other teams considering the same search but struggling with timeouts and keeping its budget too small. The strongest single agent also explores contact maximization without reaching the same quality in three hours. The successful transfer therefore includes substantial

implementation work. Agent a2 takes a shared scoring idea and makes a more thorough search practical under the execution limit.

Extending the shared objective. With the run above 0.90, a3 adds a *boundary bonus* at marker 7. The contact score now counts box walls as well as occupied cells, rewarding pieces that fit snugly against edges and into corners. This extends the same objective developed by a1 and implemented by a2, and subsequent refinements bring the run to 0.945. The sequence makes the dependence between contributions concrete. One agent identifies a better placement criterion, another turns it into an effective search, and a third improves the criterion on top of that implementation.

Figure 11 illustrates the resulting improvement on one 172-piece instance. The team packs it at 93.8% density, compared with 74.9% for an illustrative single-agent solution, leaving far fewer gaps between pieces. This single-agent run averages 0.753 across the benchmark and is distinct from the strongest run in Figure 6.

Related exchanges over longer horizons. In the Opus run (Figure 9), a1 introduces *best-fit selection*, comparing remaining pieces and candidate positions before committing the best placement. Agent a2 improves it and reduces its cost by restricting the candidate pool to the largest piece sizes. Agent a1 spends the saved time searching more box widths, carrying the score past 0.90. A later contribution from a2 changes how height and contact interact. Instead of considering contact only after minimizing height, it scores their weighted sum, allowing a slightly taller placement if it gains enough contact. The other agents incorporate the change within an hour. Here, one agent's speed improvement enables another's broader search, and the final refinement relaxes a placement rule that earlier methods treated as fixed.

The Sonnet run (Figure 10) again develops through agents that did not supply the initial solution. After a1's skyline and a2's bottom-left methods, a3 introduces best-fit selection to minimize the increase in box height. It then breaks equal-height ties by wasted space, producing the sharp jump near four hours, and adds tall-first sampling to pass 0.90. Agent a4 subsequently extends a2's width heuristic. When a1 identifies unproductive parts of the deterministic width sweep, a4 reallocates that time to randomized restarts. As in the three-hour case, progress comes from successive changes to a shared algorithm, including improvements to where it spends its limited search time.

D.2 Case Study: MNIST Classifer Compression

Finding a model worth sharing. The four agents begin with different bets. Agent a1 tries digit prototypes, a2 fixed gradient features, a3 spectral models and then binary networks, and a4 distilled CNNs. After its prototype classifier misses the accuracy threshold, a1 changes the compression strategy. A recurrent CNN reuses one convolutional cell nine times, buying depth without storing a fresh set of filters at every step. Its 5,329-byte qualifying submission persuades the other three agents to adopt this architecture within ten minutes. Their separate searches now improve different parts of a shared model.

Giving lost features a way around the bottleneck. a3 targets the final scoring layer, also called the classification head. Its 10×128 matrix converts 128 features into ten digit scores using 1,280 learned parameters, plus ten bias parameters. It first compresses the 128 input features into eight learned weighted sums before computing the ten digit scores. This stalls at 99.30% validation accuracy. To recover information lost in that compression, it also sends selected original features directly to the final scoring layer. Figure 12(b) shows these two paths in red and teal. With five original features selected using their covariance on the

training data, the model reaches 99.42% even before further fitting. The resulting 3,120-byte model qualifies, and a1 explicitly adopts the design. a4 subsequently combines seven learned weighted sums with seven directly selected features at 2,790 bytes. Passing original features to the scoring layer helps recover accuracy as the learned matrix shrinks.

Learning that fewer parameters can cost more bytes. Later, a2 reduces the learned weighted sums to six on the improved backbone. Additional directly selected features recover accuracy, but the new parameters initially require more compressed bytes than those of the larger head they replace. a2 finds that their values are less repetitive. It trains the projection parameters using only the stored integers $-2, -1, 0, 1, 2$, multiplied by a learned scale factor for each weighted sum. With ten directly selected features, this produces a qualifying 1,983-byte submission. Reducing the number of parameters works together with restricting their stored values.

Rescuing a branch that has fallen behind. Meanwhile, a1 makes the last two spatial stages share their channel offsets. Its 1,993-byte model qualifies, but the shared best has already reached 1,970 bytes. Rather than discard the branch, it combines this backbone with a2's design with six learned weighted sums, using routines originally written by a1 and extended by a2. Variants that pass ten or eleven original features directly to the scoring layer fail the test threshold. Passing twelve succeeds while keeping the projection parameters restricted to the same five stored integer values, and a1 advances the combined model to 1,960 and then 1,959 bytes. Figure 12(a) shows the adoption statements connecting these two branches. a2 then reorders the directly selected features and their matching scoring parameters to reach 1,957 bytes without changing predictions. The final architecture emerges from joining two branches, including one that could no longer win on its own.

D.2.1 The 1,957-Byte MNIST Classifier

Architecture. Figure 12(b) shows a recurrent convolutional network with 2,900 quantized parameters and 30 scale factors. The input is cyclically shifted upward by one pixel. A 3×3 convolution maps the single input channel to 32 channels, followed by SiLU to form h_0. Nine residual updates then reuse the same depthwise 3×3 filters D and pointwise 1×1 filters C. For update t, the feature maps change according to

$$\widetilde{h}_t = \mathrm{SiLU}\big(h_{t-1} + g_t \odot \mathrm{GN}_4\big(C\big(\mathrm{SiLU}(D(h_{t-1}))\big)\big) + b_t\big). \tag{3}$$

Here D filters each channel separately, C mixes channels, and GN_4 is GroupNorm with four groups, no learned affine parameters, and $\epsilon = 10^{-5}$. Each g_t contains 32 learned gains, broadcast over spatial positions. The offset b_t is the 32-parameter vector A for updates 1–3 and the shared vector B for updates 4–9. The shortcut adds the unchanged input to the learned correction, making these updates residual. All convolutions have stride one and no bias, with padding one for 3×3 filters. After updates 3 and 6, 2×2 max-pooling with stride two forms h_t from $\widetilde{h}_t$. Otherwise $h_t = \widetilde{h}_t$. This gives three updates at each of 28×28, 14×14, and 7×7 resolution.

After update 9, 4×4 average pooling with stride three produces $32 \times 2 \times 2$ features, flattened in channel-first order into $x \in \mathbb{R}^{128}$. The scoring layer combines six learned weighted sums Px with twelve directly selected features x_S,

$$\text{scores} = W \begin{bmatrix} Px \\ x_S \end{bmatrix}, \qquad P \in \mathbb{R}^{6\times 128}, \quad W \in \mathbb{R}^{10\times 18}. \tag{4}$$

The ordered, zero-based indices are $S = (2, 32, 1, 40, 102, 30, 124, 48, 93, 122, 53, 126)$. These features are selected greedily using training feature statistics to recover scoring information

(a) Communication log excerpts

a1 adopting a3's smaller scoring layer

"slot 3 rank-8-plus-residual classifier factorization [...] measured 3,117 bytes, 99.50% dev and 99.41% sealed test clearly beats the dense-head frontier [...]"

a1 adopts a3's design, combining learned weighted sums with directly selected features.

a1 borrowing a2's scoring layer

"Adopted peer rank6 head after measured 1970-byte/99.42% sealed frontier [...] 12 bypasses rather than peer 10."

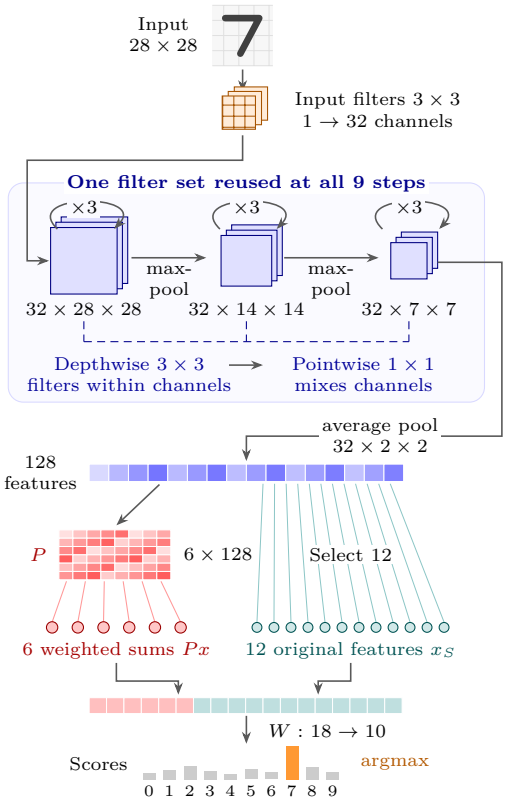
Later, a1's offset-sharing model at 1,993 bytes trails the best-so-far of 1,970. Combining it with a2's six weighted sums and twelve directly selected features reaches 1,959 bytes.

a2 taking up the combined model

"The new 1,959-byte frontier combines tied stage biases with a rank-6/twelve-bypass head and seals at 99.41%. I'm adopting it [...]"

a2 adopts the combined model and reorders features and matching scoring parameters, reaching **1,957 bytes** with unchanged predictions.

(b) Final classifier architecture



Each update uses GroupNorm, step-specific gains, residual connections, and SiLU.

Figure 12: **Combining complementary ideas produces the final MNIST classifier.** **(a)** Two shared-log entries from a1 and a subsequent excerpt from a2 show successive adoption of a3's and a2's scoring-layer designs, followed by a2 taking up the combined model. **(b)** The same filter set, comprising depthwise 3×3 and pointwise 1×1 convolution filters, updates the 32 feature maps in three loops at each resolution (see Eq. 3). The scoring layer combines six learned weighted sums with twelve selected features. The final network contains 2,900 quantized parameters and 30 float16 scale factors and achieves 99.41% test accuracy.

lost in the six-component approximation, and their indices remain fixed for every image at inference. Neither scoring matrix has a bias, and the largest score determines the digit. The convolutional network has 1,952 parameters and the scoring matrices have 948. Filter reuse supplies depth without storing nine separate filter sets, while the gains and offsets allow the updates to behave differently.

Training parameters for compression. a2's smaller scoring layer initially costs more compressed bytes, motivating a change in how its weights are trained. Quantization-aware training rounds weights during each forward pass while learning the underlying weights and scales, restricting the 768 projection parameters to scaled integers $-2, -1, 0, 1, 2$. A small weight-magnitude penalty encourages small values alongside the classification and teacher-

matching losses. This makes the smaller scoring layer competitive by favoring repeated integer values that lossless compression can exploit. Across the final model, 44.6% of stored integers are 0 or ± 1, measured before applying scales. The compressed integer payload occupies 1,262 bytes, compared with 1,390 bytes for fixed-width packing of the same values using each tensor's range, by exploiting redundancy in the learned weights.

The human reference, `tiny_MNIST` (Dhairyashil R. G., 2024), uses five separate 3×3 convolutions with widths $6, 6, 8, 8, 10$, BatchNorm, ReLU, and a dense 90-to-10 head. It has 3,130 training parameters before BatchNorm folding. Our reproduction folds BatchNorm into the convolutions and applies per-channel four-bit quantization, yielding 2,461 bytes at 99.40% test accuracy. The agents' 1,957-byte submission is 504 bytes (20.5%) smaller and achieves 99.41% test accuracy.

E Analysis for Verified Progress Sharing

This appendix gives the calculations behind the conceptual model and Proposition 1. The mathematics is largely standard rather than new. It combines elementary facts about exponential order statistics, Erlang sums, and Chernoff bounds. We keep the model deliberately simple so that it isolates how reusable, verified progress can change the value of parallel search. It is not intended as an empirical model of agent completion times or as a general theory of multi-agent communication.

E.1 Verified Progress Sharing Framework

In the following discussion, we provide a simple pedagogical model that formalizes this intuition. We show how it can lead to an exponential separation between communicating and independent agents, and also provide a small-scale experiment on Terminal-Bench 2.0 (Merrill et al., 2026) where team@k improves over single-agent attempts but not independent best@k, illustrating the importance of the availability of an agent-accessible verifier.

From independent trajectories to cumulative progress. Consider a task requiring m successive improvements, with an accessible verifier reporting score j/m after improvement j. Assume that continuation difficulty depends only on the stage, improvements can be transferred freely, and agents conduct fresh, independent searches after each transfer. Both methods can query the verifier. Writing X_{ij} for agent i's search time at stage j, their completion times would be

$$T_{\text{best}} = \min_{i \le k} \sum_{j=1}^{m} X_{ij}, \qquad T_{\text{team}} = \sum_{j=1}^{m} \min_{i \le k} X_{ij}. \tag{5}$$

Thus $T_{\text{team}} \le T_{\text{best}}$. Independent sampling needs one agent to complete the entire sequence quickly, whereas a team can use a different agent's breakthrough at each stage. Communication changes a *minimum of sums* into a *sum of minima*. Figure 2 illustrates this effect visually.

Suppose $X_{ij} \overset{\text{iid}}{\sim} \text{Exp}(\lambda)$, which is the standard constant-rate, memoryless model used in queueing theory (Shortle et al., 2018). Here we use this model as a pedagogical example, not an empirical claim about agents. The first of k independent discoveries arrives at rate $k\lambda$, giving

$$T_{\text{team}} \sim \text{Erlang}(m, k\lambda), \qquad \mathbb{E}T_{\text{team}} = \frac{m}{k\lambda}.$$

The Erlang law (a Gamma distribution with integer shape) follows from summing m exponential waiting times. Doubling k therefore halves the team's mean completion time.

Now give each agent runtime $\tau = \alpha m/\lambda$, where $1/k < \alpha < 1$. This is shorter than one agent's mean completion time but longer than the team's. Both methods receive the same $k\tau$ agent-time budget.

Proposition 1 (Exponential separation from progress sharing)**.** *Under this model, writing* $I(a) = a - 1 - \log a$*,*

$$\Pr(T_{\text{team}} \le \tau) \ge 1 - e^{-mI(k\alpha)},$$
$$\Pr(T_{\text{best}} \le \tau) \le ke^{-mI(\alpha)}.$$

For fixed k, α*, team@k approaches one exponentially in* m*, while best@k success approaches zero exponentially.*

The separation concerns completion probability at matched runtime, not an exponential expected-time speedup. The proof is in Appendix E.

The benefit also depends on preserving diverse continuations. Sometimes, communicating agents may converge on a subset of approaches, which can reduce diversity and slow progress. To model *herding*, we can imagine the k agents acting as essentially $g \le k$ independent groups whose members duplicate the same search. The team's discovery rate becomes $g\lambda$ and its mean completion time becomes $m/(g\lambda)$. The high-success regime now requires $g\alpha > 1$. Sharing progress helps only to the extent that independent search survives after sharing.

Ineffective communication in Terminal-Bench. Terminal-Bench 2.0 evaluates agents on hard, realistic tasks performed in command-line environments (Merrill et al., 2026). We evaluate Claude Sonnet 4.6 through the GitHub Copilot CLI on the complete set of 89 tasks. Test-time communication consists of two independent team@2 trials, where both agents contribute to one final container state. The baseline consists of four independent single-agent trials. This is a small-scale comparison with only a handful of trials, so we treat it as descriptive evidence rather than a precise estimate of the gap or its cause.

Table 4: **Communication does not outperform pass@k on Terminal-Bench 2.0.**

Method	Mean Accuracy	Max Accuracy
Single agent pass@1	52.53%	–
Independent pass@2	**62.36%**	**64.04%**
Communicating team@2	60.67%	61.80%

As seen in Table 4, communication improves over an individual attempt, but does not outperform independent sampling (pass@k) with the same number of agents. This result can be partially explained by Proposition 1 when m is effectively 1. In Terminal-Bench, tasks require long sequences of actions, but the official verifier runs after agent execution. Available feedback varies across tasks, from package tests and direct performance measurements to checks of selected requirements. For example, the public evaluator in `largest-eigenval` checks the eigenvector equation and reports timings without verifying that the eigenvalue is dominant. In `mailman`, the public evaluator checks subscription but omits announcement delivery and unsubscription, which the official grader also tests. Such checks can confirm progress on individual requirements without establishing overall correctness.

In contrast, ARC-AGI-3 supplies immediate action and level feedback, while our packing and MNIST tasks expose numerical scores that agents can query repeatedly. Thus, setting $m = 1$ in Proposition 1 gives us $T_{\text{team}} = \min_{i \le k} X_{i1} = T_{\text{best}}$, and therefore predicts no communication advantage, even though the final outcome can be graded objectively. What matters is not evaluation alone, but access to dense faithful verification at test-time.

A secondary disadvantage may be the loss of diversity. If communication causes k agents to behave like only $g < k$ independent groups, then g end-to-end candidates cover only $1-(1-p)^g$, below pass@k's $1-(1-p)^k$, where p denotes the success probability of an independent candidate. This penalty is especially relevant here because team@2 contributes one shared final state, while pass@2 retains two isolated states and receives oracle selection after evaluation. Premature convergence or interference in the shared state can therefore make communication slightly worse than independent sampling. Together, the results suggest that communication is most useful when feedback is accessible and discriminative, though we do not exclude the possibility that test-time communication could be useful when feedback is sparse, if the agents can construct their own intermediate verification signals or have better communication protocol harnesses.

E.2 Proof for Verified Progress Sharing Framework

We first compare independent and communicating agents under the same collection of stage-level search times. Recall that X_{ij} is the time agent i would take to discover the next improvement at stage j. A communicating team uses the first discovery at every stage. Therefore, for every agent i,

$$T_{\text{team}} = \sum_{j=1}^{m} \min_{\ell \le k} X_{\ell j} \le \sum_{j=1}^{m} X_{ij}.$$

Taking the minimum over agents gives $T_{\text{team}} \le T_{\text{best}}$. This is the pointwise comparison in the main text. It captures the distinction between combining stage-level breakthroughs and selecting one complete trajectory after all runs finish.

Under the modeling assumption $X_{ij} \overset{\text{iid}}{\sim} \text{Exp}(\lambda)$, define the waiting time for the team's next verified improvement as $M_j = \min_{i \le k} X_{ij}$. Its survival probability is

$$\Pr(M_j > t) = \prod_{i=1}^{k} \Pr(X_{ij} > t) = e^{-k\lambda t}, \qquad t \ge 0.$$

Thus each M_j is distributed as $\text{Exp}(k\lambda)$. Independence across stages then gives

$$T_{\text{team}} \sim \text{Erlang}(m, k\lambda), \qquad \mathbb{E}T_{\text{team}} = \frac{m}{k\lambda}.$$

For comparison, each independent agent must complete all m stages itself. Its completion time $T_i = \sum_{j=1}^{m} X_{ij}$ satisfies

$$T_i \sim \text{Erlang}(m, \lambda), \qquad T_{\text{best}} = \min_{i \le k} T_i.$$

These distributions provide the ingredients for the fixed-budget comparison.

E.3 Completion probability at a fixed budget

We now prove Proposition 1. The only additional ingredient is a standard exponential tail bound for an Erlang random variable.

Proof of Proposition 1. Let $S \sim \text{Erlang}(m, 1)$. Its moment-generating function is

$$\mathbb{E}[e^{\theta S}] = (1-\theta)^{-m}, \qquad \theta < 1.$$

For $a > 1$, apply Markov's inequality with $\theta = 1 - 1/a > 0$ to obtain

$$\Pr(S \ge am) \le e^{-\theta am}(1-\theta)^{-m} = e^{-mI(a)},$$

where $I(a) = a - 1 - \log a$. For $0 < a < 1$, the same choice gives $\theta < 0$. Since $S \le am$ implies $e^{\theta S} \ge e^{\theta am}$, Markov's inequality gives the corresponding lower-tail bound. Hence

$$\begin{aligned} \Pr(S \ge am) &\le e^{-mI(a)} \quad \text{for } a > 1, \\ \Pr(S \le am) &\le e^{-mI(a)} \quad \text{for } 0 < a < 1. \end{aligned} \tag{6}$$

Set $\tau = \alpha m/\lambda$, with $1/k < \alpha < 1$. Since $k\lambda T_{\text{team}}$ has the same distribution as S and $k\alpha > 1$,

$$\Pr(T_{\text{team}} > \tau) \le e^{-mI(k\alpha)}.$$

Likewise, λT_i has the same distribution as S, so $\alpha < 1$ gives

$$\Pr(T_i \le \tau) \le e^{-mI(\alpha)}.$$

Taking a union bound over the k independent agents yields

$$\Pr(T_{\text{best}} \le \tau) = \Pr\left(\bigcup_{i=1}^{k} \{T_i \le \tau\}\right) \le k e^{-mI(\alpha)}.$$

Combining the two bounds proves

$$\Pr(T_{\text{team}} \le \tau) \ge 1 - e^{-mI(k\alpha)}, \qquad \Pr(T_{\text{best}} \le \tau) \le k e^{-mI(\alpha)}.$$

Because $I(a) > 0$ for $a > 0$ and $a \ne 1$, both exponents are strictly positive for fixed k and $\alpha \in (1/k, 1)$. □

F Future Directions

In this paper, we studied the capabilities of test-time communication, a multi-agent setting in which agents pursue the same open-ended objective and exchange evidence and artifacts through a shared workspace. This allows agents to explore different approaches in parallel while turning individual discoveries into cumulative progress. On `ARC-AGI-3`, team@3 matches the solve rate of 13 independent agents and team@5 matches that of 33, with the multiplier growing from 4.3× to 6.6× as the team scales. Communication also solves a game that remains unsolved across single-agent trials and makes the average member of a team as action-efficient as the best of independent agents. The same pattern extends to longer-horizon tasks. Communicating agents reach a packing score of 0.945 on Frontier-CS polyomino packing, above the prior best-known score of 0.894, and produce a 1,957-byte MNIST classifier, improving on both the best@4 result and the 2,461-byte best-known human solution. These gains emerge only after an initial coordination cost, but with adequate compute, communication can outperform independent agents in both performance and compute efficiency.

The agent-accessible verifier in each task is central to our experimental design. Thus, our setting differs from multi-agent debate, which asks agents to reconcile answers without necessarily grounding the exchange in environmental feedback. Level completion, packing score, and compression size give agents an objective basis for rejecting failures, comparing approaches, and building on improvements. Whether communication produces similar gains when feedback is sparse, noisy, or subjective remains an important open question.

On the other hand, multi-agent harnesses may matter as much as single-agent harnesses. Our controlled design is deliberately narrow. We fix one communication harness and use homogeneous agents with identical instructions and no assigned roles or central orchestrator. This isolates communication, but it does not imply that roles, model diversity, communication

topology, or orchestration are unimportant. Future work should consider how agents communicate, which peers receive their messages, or how teams should be organized. These policies could be engineered or evolved, similarly to harness optimization (Lee et al., 2026b;a). Our results provide a baseline for future work, and hopefully motivate further study of multi-agent collaboration and test-time communication.