TEST-TIME COMPUTE SCALES BETTER WITH PARALLEL AGENT TEAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling improves the performance of a fixed model by spending more computation at inference. There are two common ways to spend it on agents when a verifier can score every attempt. Best-of-N sampling runs many independent agents and keeps the best result but saturates quickly. One large team of collaborating agents can score higher but stops improving as it grows. Can we get the best of both? Our method, *parallel agent teams*, does so by sampling many small teams in parallel and keeping the best team's result. We test it on 12 long-horizon tasks from GPU kernels and formal proofs to exploit development. With a 27B model and a budget of 100 agents, parallel teams of ten beat parallel agents on 10 of 12 tasks and beat one team of 100 on all four tasks where we ran such large teams. At equal dollar cost the same parallel teams beat a frontier model on half the tasks. A simple statistical model shows that individual agents lose because their many samples have a low mean and that one large team loses because a single sample gets no gain from variance. Sampling many small teams in parallel balances a high mean with the gain from variance. Parallel agent teams thus offer a more effective way to scale test-time compute.

1 INTRODUCTION

Test-time compute lets a fixed pre-trained model do better by spending more computation at inference (Snell et al., 2024; Wu et al., 2025). An ideal method would turn more compute into better results steadily. We study tasks where a verifier can score every attempt and only the best solution matters. Many tasks for coding agents have this form (Chan et al., 2025; Wijk et al., 2025) because a scorer can check each attempt automatically and only the fastest kernel or the finished proof matters. On such tasks there are two common ways to spend more compute on agents. The first is best-of-N sampling, one of the strongest methods when a verifier is available (Chen et al., 2021; Cobbe et al., 2021; Li et al., 2022; Brown et al., 2024). It runs N independent agents and keeps the one with the

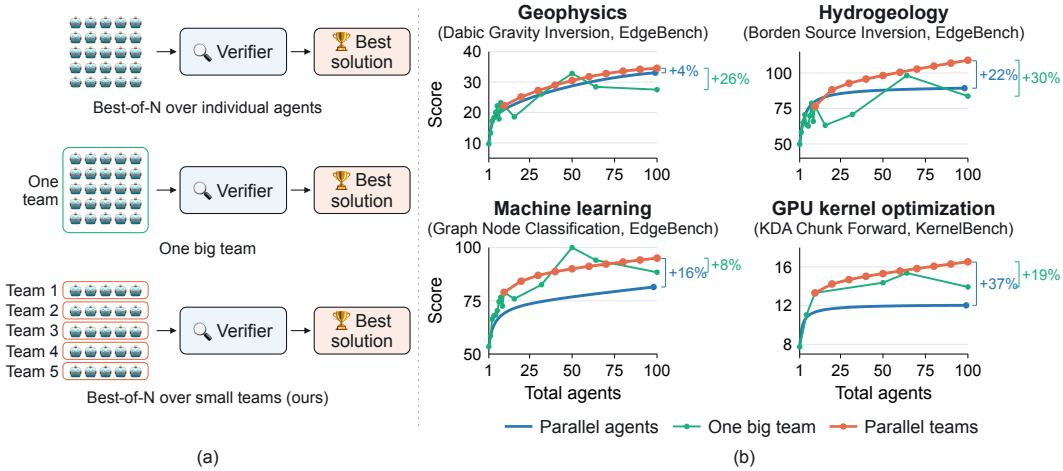


Figure 1: **Sample small teams, not agents or one big team.** (a) Three ways to spend the same agents. A verifier keeps the best solution. (b) Qwen3.8-27B on four tasks. Parallel agents (blue) flatten quickly, one big team (green) rises and falls with its size, and parallel teams of ten (orange) keep improving. Brackets give the gain of parallel teams over each baseline at the largest budget both reach.

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.

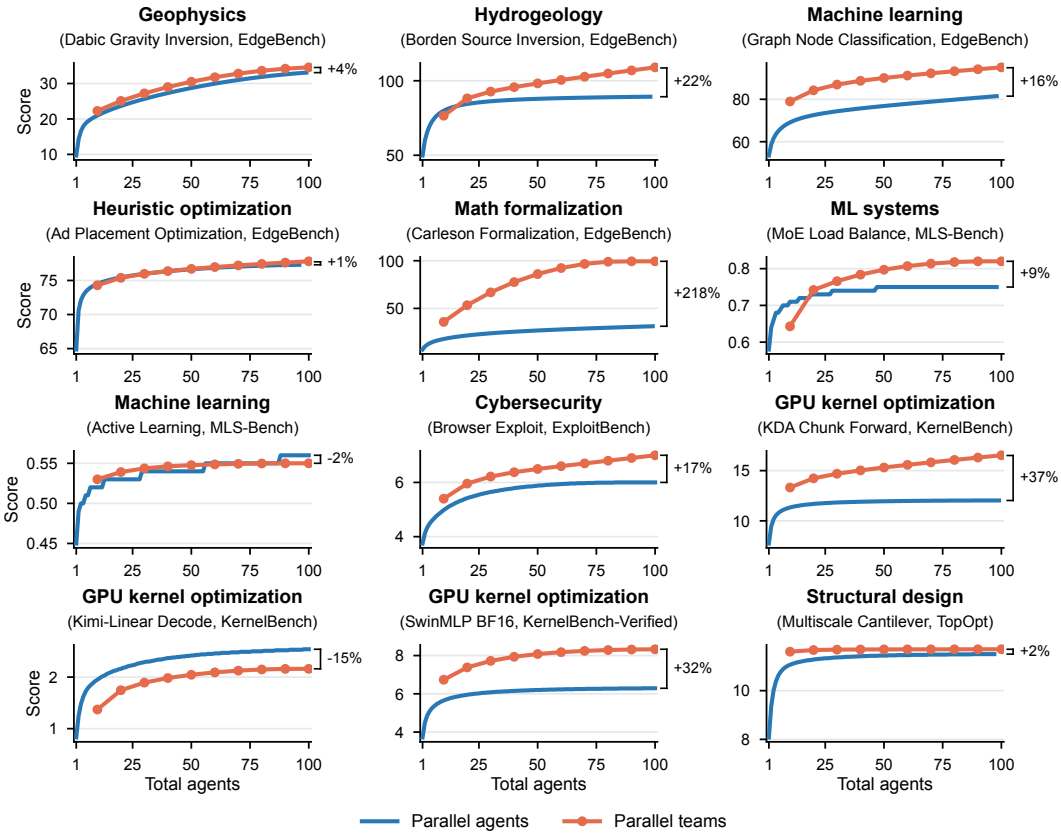


Figure 2: **Parallel teams beat parallel agents on 10 of 12 tasks at 100 agents.** Qwen3.8-27B parallel agents (blue) and parallel teams of ten (orange). Brackets give the teams' gain at the largest budget both reach.

highest score. Yet best-of-N over individual agents saturates quickly. In our experiments, 50 agents already reach 81% to 100% of the gain that 100 agents reach.

The second way is to let more agents work together on the task as one team. Multi-agent systems split the work among agents with different roles (Hong et al., 2024; Qian et al., 2024; Wu et al., 2023), and recent work grows such teams to hundreds or thousands of agents (Qian et al., 2025). A larger team does not give more samples to select from. It changes a single sample, and its score does not rise steadily with its size (Qian et al., 2025; Kim et al., 2025). Large groups of people and of language models tend to converge on one solution and lose the diversity that exploration needs (Lazer & Friedman, 2007; Wu et al., 2019; Weng et al., 2025).

Each approach has one half of what we need. Best-of-N takes advantage of the variation across samples. A team lets its agents work together and reach better performance more efficiently. We introduce *parallel agent teams*, which combine the two by applying best-of-N to independent small teams of collaborating agents (Figure 1a). Each team is a small group of agents that share a forum. At the same budget of agents, parallel teams score higher than both parallel agents and one big team and keep improving as the budget grows (Figure 1b). Averaged over 12 tasks, parallel teams of ten score 11% higher than parallel agents at a total budget of 10 agents and 29% higher at a total budget of 100 agents. At 100 agents, ten teams of ten beat one team of 100 by 8% to 30%. At equal dollar cost, teams of a 27B model also beat the frontier model GPT-5.6-sol on 4 of 8 tasks.

A simple statistical model of best-of-N explains why parallel small teams win. The expected best of several samples is the mean score of one sample plus a gain from variance, which grows with the number of samples the budget buys. Parallel agents collect the largest gain from variance because they draw the most samples, but each agent has a low mean. One big team has a high mean, but it is a single sample and wastes the gain from variance, and a larger team does not guarantee a higher mean. Parallel small teams balance the two. Each team has a much higher mean than an agent, and several teams still collect most of the gain from variance. They also keep improving as the budget grows. At a given budget they draw fewer samples than parallel agents because each team costs

Table 1: **The 12 tasks.** Higher is better for every metric.

Task	Benchmark	Domain	Metric
Dabic Gravity Inversion	EdgeBench	Geophysics	points 0–100
Borden Source Inversion	EdgeBench	Hydrogeology	points 0–100
Graph Node Classification	EdgeBench	Machine learning	points 0–100
Ad Placement Optimization	EdgeBench	Heuristic optimization	points 0–100
Carleson Formalization	EdgeBench	Math formalization	points 0–100
MoE Load Balance	MLS-Bench	ML systems	reward 0–1
Active Learning	MLS-Bench	Machine learning	reward 0–1
Browser Exploit	ExploitBench	Cybersecurity	ladder 0–16
KDA Chunk Forward	KernelBench	GPU kernel optimization	1/latency
Kimi-Linear Decode	KernelBench	GPU kernel optimization	1/latency
SwinMLP BF16	KernelBench-Verified	GPU kernel optimization	speedup
Multiscale Cantilever	TopOpt	Structural design	C_{ref}/C

several agents, so their best-of-N curve is still on its steep early part, where each additional sample raises the best result the most.

Collaboration raises the mean score of each sample, and sampling turns the variance across samples into gains. Parallel agent teams get both and keep improving where best-of-N over individual agents has saturated, offering a simple way to scale test-time compute further.

2 Parallel Agent Teams

We first describe best-of-N sampling and how we measure it, and then agent teams. We then introduce parallel agent teams, which combine the two.

Best-of-N sampling. We study tasks with a verifier that scores any candidate solution so that the best of several candidates can be found without human judgment. For a fixed model, drawing more independent responses and keeping the best steadily improves performance (Brown et al., 2024). Let X be the score of one sample, such as one agent, defined as the score of its best verified submission. We report the expected best of N independent samples:

$$\mathbb{E}\Big[\max_{1\le i\le N} X_i\Big]. \tag{1}$$

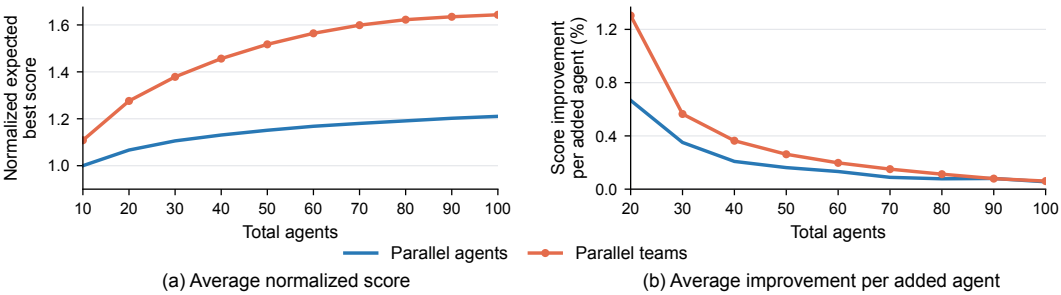
We estimate it from a pool of L runs by taking every subset of N runs and averaging its best score over all $\binom{L}{N}$ subsets. With the scores sorted as $v_{(1)} \le \cdots \le v_{(L)}$, this average is $\sum_{i=N}^{L} v_{(i)}\binom{i-1}{N-1}/\binom{L}{N}$. One pool thus gives the whole curve from $N = 1$ to $N = L$. Take five runs that score 2, 4, 5, 7 and 9. Their expected best is 5.4 for one run (the mean), 7.1 for two and 9 for all five.

Agent teams. Instead of working alone, agents can work together as a team on the same task. Past work shows that such teams can produce better results than a single agent (Li et al., 2023; Du et al., 2024; Hong et al., 2024; Qian et al., 2024; Chen et al., 2024). We mainly study flat teams, in which peers share progress through a common log or memory (Park et al., 2026; Qu et al., 2026). Each agent has its own sandbox and context but shares a forum where any agent can post and read notes at any time. There are no roles, no manager and no assigned work. A team's score is the best verified submission of any member (Appendix E). Section 6 studies an alternative team architecture in which a manager assigns the work.

Parallel agent teams. Parallel agent teams combine the two by applying best-of-N to independent teams (Figure 1a), so a sample is one team. We measure compute as a budget of B agents that each run for the same fixed time. It buys $N = B/k$ teams of size k. Parallel agents ($k = 1$) and one big team ($k = B$) are the two ends of this family, and parallel teams lie in between. We use teams of ten unless stated otherwise and study the team size in Section 3.1.

3 Parallel Small Teams Beat Both

Setup. We use 12 long-horizon optimization tasks from six benchmarks (ByteDance Seed, 2026; Lyu et al., 2026; Lee & Brumley, 2026; Elliot Arledge, 2026; Meta FAIR, 2026; Ouyang et al., 2025; Chen et al., 2025). They span nine domains from GPU kernels to formal mathematics (Table 1). In



(a) Average normalized score (b) Average improvement per added agent

Figure 3: **Averaged over 12 tasks, parallel teams score higher and keep improving for longer.** Qwen3.8-27B parallel agents (blue) and parallel teams of ten (orange). (a) Expected best score normalized by that of ten agents. The gap widens with the budget. (b) Parallel teams gain more from each added agent.

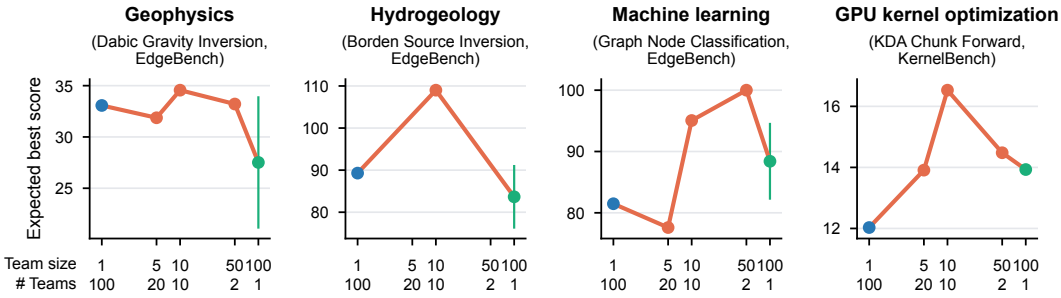


Figure 4: **At a fixed budget, the best team size lies in between.** 100 Qwen3.8-27B agents form $100/k$ teams of size k. The ends are parallel agents ($k = 1$; blue) and one big team ($k = 100$; green, with the range of its runs as a bar).

each task an agent gets a description, a working environment and a scorer it can call and then improves its submission for 12 hours. Each agent runs in the Codex harness (OpenAI, 2025) inside its own sandbox. Our main model is Qwen3.8-27B (Qwen Team, 2026). Section 5 repeats the comparison with a stronger model, DeepSeek V4.1 Flash (DeepSeek-AI, 2026), and Section 3.4 compares against GPT-5.6-sol (OpenAI, 2026). Each task has about 100 independent runs and 10 teams of ten, and four tasks also have single teams of 16 to 100 agents (Appendix D).

3.1 Better than one big team

Figure 1b adds one team of B agents on the four tasks where we ran one big team. Its score rises at first but not steadily. From 64 to 100 agents it drops on all four tasks. At 100 agents, ten teams of ten beat one team of 100 on all four tasks by 8% to 30%, and on two tasks one team of 100 even scores below the best of 100 independent agents.

Figure 4 holds the budget at 100 agents and varies the team size. The best size lies in between on every task. Ten teams of ten score highest on three tasks, and two teams of 50 on Graph Node Classification, where both reached the maximum score. Section 4 explains why both ends lose.

3.2 Better than parallel agents

Figure 2 compares parallel agents with parallel teams of ten on each task. At a budget of 50 agents, the best of five teams beats the best of 50 agents on 11 of 12 tasks. At 100 agents, the best of ten teams beats the best of 100 agents on 10 of 12 tasks (Table 4). The gains are largest where individual agents rarely succeed. On Carleson Formalization the best of 100 agents scores 31.3 out of 100 while the best team scores 99.4 and nearly completes the proof. Averaged over the 12 tasks, parallel teams score 11% higher than parallel agents at a total budget of 10 agents and 29% higher at 100 agents (Table 4). Parallel agents beat parallel teams only on Kimi-Linear Decode and, by a small margin, on Active Learning.

3.3 Slower saturation than parallel agents

Parallel agents improve monotonically as compute grows because an extra sample can only raise the best score, but they saturate quickly. At 20 agents, each task has already reached 60% to 94% of the gain that about 100 agents bring. Parallel teams keep improving for longer (Figure 3b). From 10 to

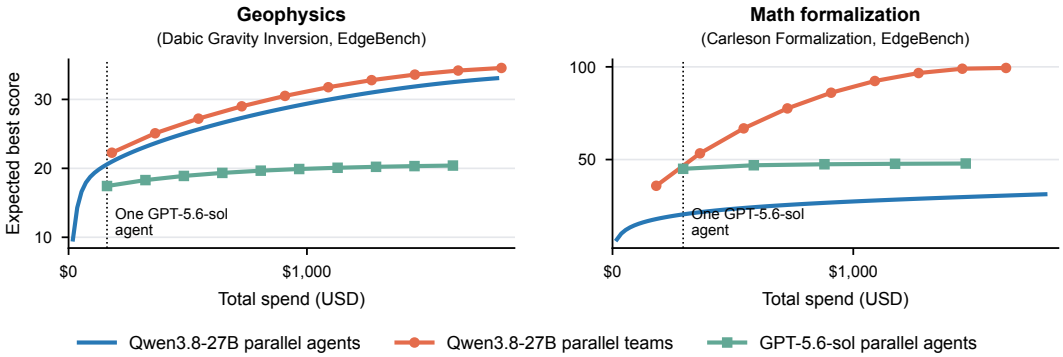


Figure 5: **At equal cost, parallel teams of a 27B model overtake a frontier model.** Qwen3.8-27B parallel agents (blue) and parallel teams of ten (orange), priced by GPU time, against GPT-5.6-sol agents (green), priced by tokens. The dotted line marks the price of one GPT-5.6-sol agent.

20 agents each extra agent adds 1.30% to the score of parallel teams but only 0.67% to the score of parallel agents. The gap between the two methods therefore widens as the budget grows.

3.4 Beating a frontier model

We also compare teams with GPT-5.6-sol at equal dollar cost (Figure 5). A Qwen3.8-27B team of ten costs about as much as one or two GPT-5.6-sol agents (Appendix J). On Carleson Formalization one GPT-5.6-sol agent beats one Qwen3.8-27B team, yet for the price of five GPT-5.6-sol agents, eight Qwen3.8-27B teams reach 99.0 against 47.9. Across the eight tasks with at least five GPT-5.6-sol runs, Qwen3.8-27B teams beat five GPT-5.6-sol agents at the same price on four. GPT-5.6-sol scores high but varies little from run to run. More GPT-5.6-sol runs therefore add little while the best of several Qwen3.8-27B teams keeps climbing.

4 Why Do Parallel Teams Win?

Parallel teams beat both common approaches. They beat parallel agents although at the same budget they draw ten times fewer samples, since a team of ten costs ten agents. They beat one big team, which puts the whole budget into a single sample. A simple statistical model of best-of-N explains both results.

4.1 A statistical model

Assume that the score X of one sample follows a normal distribution $\mathcal{N}(\mu, \sigma^2)$ and that samples are independent. The expected best of N samples is then $\mu+\sigma a_N$. Here a_N is the expected maximum of N standard normal variables, and $a_N \approx \sqrt{2\ln N}$ for large N (Appendix A; Fisher & Tippett, 1928; Gnedenko, 1943). Now let each sample cost c agents so that a budget of B agents buys $N = B/c$ samples. The expected best is then

$$V(B) \approx \mu + \sigma\sqrt{2\ln(B/c)}. \tag{2}$$

The three numbers affect the curve at different rates. The mean μ adds the same amount at every budget. The standard deviation σ multiplies the gain from variance. This gain grows only as $\sqrt{2\ln N}$. The cost c cuts the number of samples from B to B/c but lowers the gain from variance by only about $\sigma \ln c/\sqrt{2\ln B}$. A tenfold cost therefore matters little once the budget is large. For teams of size k the cost is $c = k$, and the mean μ_k and standard deviation σ_k depend on k:

$$V_k(B) \approx \mu_k + \sigma_k\sqrt{2\ln(B/k)}. \tag{3}$$

The first term is the mean of one team and the second is the gain from variance when sampling B/k teams. Parallel agents ($k = 1$) and one big team ($k = B$) are the two ends of this family.

4.2 One big team wastes variance

Parallel sampling takes advantage of the variance of scores. By Equation 2, more samples raise the expected best by $\sigma\sqrt{2\ln N}$, so performance scales with compute at a rate set by the standard deviation σ. Parallel agents and parallel teams both draw more samples as the budget grows and

collect this gain. One big team wastes it. It is a single sample, so $\sqrt{2\ln(B/k)} = 0$ and its expected score is just its mean μ_B, whatever its standard deviation. A larger budget then helps one big team only if a larger team has a higher mean, and a larger team does not guarantee that. From 64 to 100 agents the mean of one team falls on all four tasks where we ran such teams (Section 3.1). One big team therefore loses to parallel teams, which raise the mean and still keep most of the gain from variance.

4.3 Parallel teams balance mean and variance

Parallel teams balance the gain from the mean and the gain from variance. Parallel agents collect the largest gain from variance, but one agent has a low mean. A team of ten raises the mean but draws fewer samples and so gives up some gain from variance. By Equation 2, an expensive method (μ_2, σ_2) at cost c beats a cheap method (μ_1, σ_1) at cost 1 at budget B when

$$\mu_2 + \sigma_2\sqrt{2\ln(B/c)} > \mu_1 + \sigma_1\sqrt{2\ln B}. \quad (4)$$

With equal standard deviations, drawing B/c samples instead of B loses only about $\sigma \ln c/\sqrt{2\ln B}$, and this loss shrinks as the budget grows. A higher mean counts in full at every budget. At a budget of 200, a mean gain of only 0.88σ pays for a $10\times$ cost. Figure 6 shows how the mean, standard deviation and cost each change the curve. A method at ten times the cost can still win through a higher mean or a higher standard deviation.

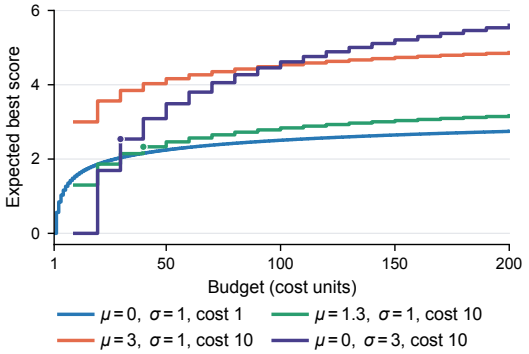


Figure 6: **How the mean, standard deviation and cost of a sample shape the best-of-N curve.** Expected best score in the statistical model for four sampling methods. A higher mean shifts the curve up, a larger standard deviation makes it rise faster, and a higher cost delays its start.

4.4 Testing the statistical model

We now check the statistical model against our measurements. On Graph Node Classification, a team of ten scores 79.0 on average and an agent 52.1 (Table 2). With the agents' standard deviation $\sigma = 11.8$, the team's mean is 2.3σ higher. Equation 4 requires only 1.4σ at a budget of 100 agents. Scores on this task and on Browser Exploit are close to normal, and the fitted normals predict the measured curves well (Figure 7).

Table 2 extends the check to all 12 tasks. A team's mean exceeds an agent's by 0.5σ to 4.9σ on every task, and the statistical model predicts the sign of the teams' gain at 100 agents on 9 of 12. It overestimates the best of many agents where agent scores are far from normal, as with the long lower tail of failed runs on Ad Placement Optimization ($W = 0.66$) or the pile-up below a ceiling on KDA Chunk Forward (Appendix B). The statistical model also explains the frontier model's flat curve: a high mean with a small standard deviation leaves little for parallel sampling to gain.

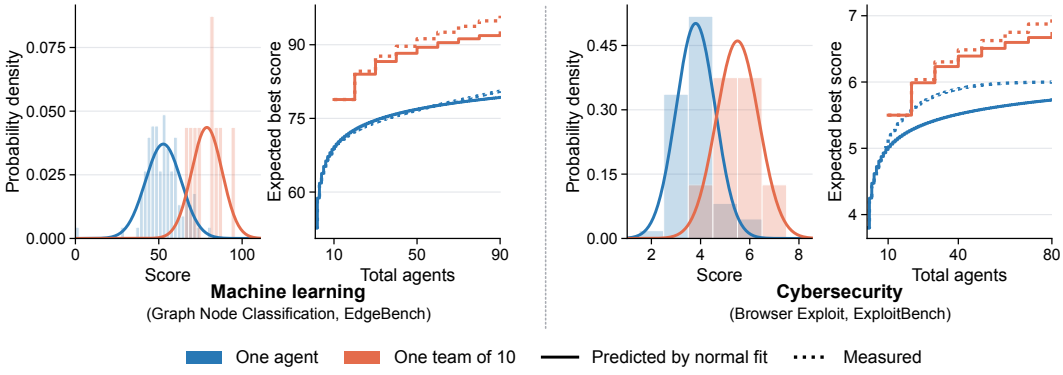
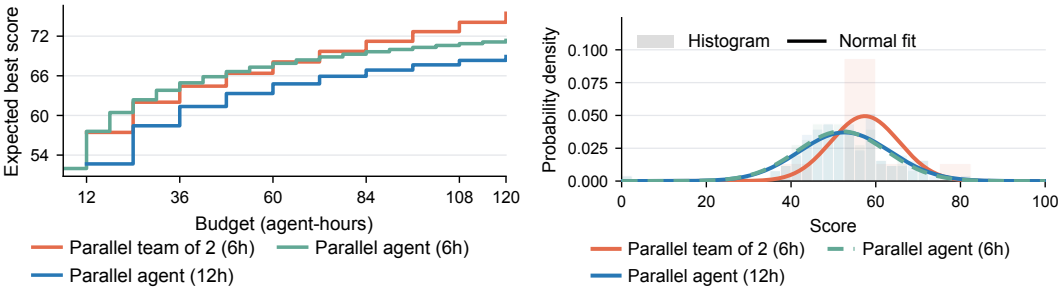


Figure 7: **The statistical model predicts the measured curves when scores are close to normal.** Left of each task: score distributions of one Qwen3.8-27B agent (blue) and one team of ten (orange) with normal fits. Right: predicted (solid) and measured (dotted) expected best score.

Table 2: **The statistical model on all 12 tasks.** Mean μ and standard deviation σ of one Qwen3.8-27B agent and one team of ten. W is the Shapiro–Wilk statistic (Shapiro & Wilk, 1965), which equals 1 for a normal sample. Gain is the expected best of 10 teams over that of 100 agents. Italic marks a predicted gain with the wrong sign.

	One agent			One team of ten			Gain at 100 agents	
Task	μ	σ	W	μ	σ	W	predicted	measured
Dabic Gravity Inversion	9.5	8.6	0.79	22.3	5.4	0.73	*-1.4%*	+4.6%
Borden Source Inversion	49.1	18.9	0.96	76.5	19.5	0.93	+10.5%	+22.1%
Graph Node Classification	52.1	11.8	0.89	79.0	8.5	0.96	+12.7%	+17.0%
Ad Placement Optimization	62.0	17.9	0.66	74.3	1.8	0.99	*-28.0%*	+0.7%
Carleson Formalization	6.8	5.9	0.89	35.9	33.3	0.67	+304.5%	+218.4%
MoE Load Balance	0.566	0.146	0.80	0.643	0.222	0.59	+5.7%	+9.2%
Active Learning	0.451	0.083	0.70	0.530	0.015	0.87	-15.9%	-1.6%
Browser Exploit	3.75	0.75	0.82	5.40	0.92	0.90	+20.8%	+16.7%
KDA Chunk Forward	7.70	3.11	0.93	13.32	1.53	0.95	+1.1%	+37.4%
Kimi-Linear Decode	0.797	0.749	0.86	1.371	0.622	0.92	-13.0%	-14.7%
SwinMLP BF16	3.58	1.50	0.97	6.74	1.02	0.96	+13.2%	+32.5%
Multiscale Cantilever	7.81	2.58	0.91	11.60	0.12	0.77	*-17.5%*	+1.7%



(a) Expected best score at equal agent-hours

(b) Score distributions and normal fits

Figure 8: **Two agents for 6 hours beat one agent for 12 hours.** Qwen3.8-27B on Graph Node Classification. (a) Expected best score against agent-hours. (b) A longer session barely shifts the score distribution, while a team of two at the same cost shifts it to the right.

4.5 Fewer samples, steeper gains

The same statistical model explains why teams scale better. The best-of-N curve rises fastest at small N and flattens as N grows. At a given budget an expensive method has drawn few samples and still sits on the steep part of its curve while a cheap method is already on the flat part. Differentiating Equation 2 gives the return to one more agent of budget, $\sigma/\left(B\sqrt{2\ln(B/c)}\right)$. This return grows with the cost c. At 20 agents the statistical model gives a team of ten ($N = 2$) a marginal return $\sqrt{\ln 20/\ln 2} \approx 2.1$ times that of an agent ($N = 20$) at equal standard deviation. The measured ratio is 1.30% to 0.67% (Figure 3b). Earlier work observes a trade-off between the quality and the quantity of samples at equal compute (Brown et al., 2024; Snell et al., 2024; Wu et al., 2025). Our statistical model makes this trade-off precise for best-of-N.

5 Other Ways to Spend Compute

Equation 4 applies to parallel sampling with any way of producing a sample. We use it to evaluate two other choices, a longer session for each agent and a different model.

Longer sessions. Instead of forming a team we can let each agent work longer. A 12-hour agent costs as much as a team of two agents that each work 6 hours (Figure 8, Appendix G). Longer sessions barely raise the mean. On Graph Node Classification an agent's mean score rises only from 52.0 at 6 hours to 52.7 at 12, in line with the slowdown of long agent sessions (Liu et al., 2026). By Equation 4, doubling the cost pays off at this budget only if the mean rises by about 3.4 points. The 12-hour agents therefore lose even to plain best-of-N over 6-hour agents. A team of two spends the same hours on two approaches, raises the mean by 5.4 points and beats both.

A different model. A different model changes the mean and standard deviation of every way of producing a sample. Figure 9 repeats the comparison with DeepSeek V4.1 Flash (Appendix H shows

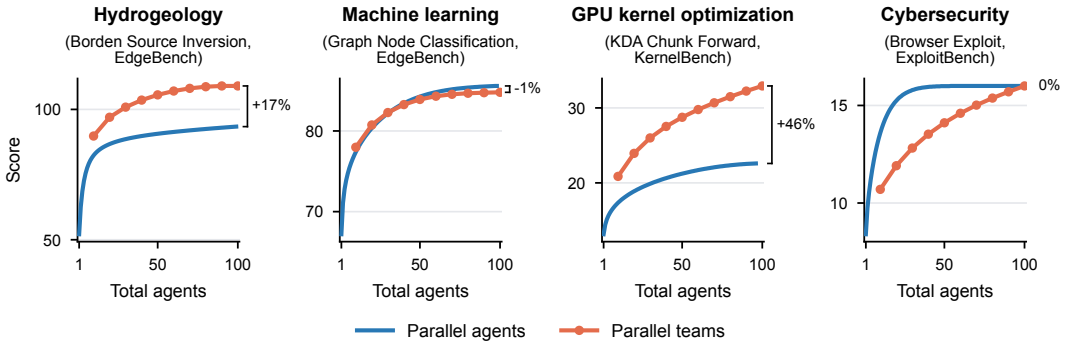


Figure 9: **With DeepSeek V4.1 Flash, teams gain less on tasks that are easy for the model.** Parallel agents (blue) and parallel teams of ten (orange) on four of the eight tasks in Figure 12. Brackets give the teams' gain at the largest budget.

all eight tasks). A single agent already has a high mean and a small standard deviation on tasks that are easy for the model. Its best-of-N curve then reaches the top score quickly and leaves a team little room to add. On tasks that are still hard for the model, a team raises the mean a lot and parallel teams beat parallel agents clearly.

6 Inside a Team

A team of ten scores higher than the best of ten independent agents. On Graph Node Classification, one team scores 79.0 on average and the best of ten independent agents scores 69.1. Eight of the 12 tasks show the same pattern (Table 4), and so do most smaller teams (Appendix F). Both use ten agents for the same time, so the extra gain must come from what agents share.

Sharing discoveries. To see what agents share, we analyzed the forum posts of our team runs (Appendix I.1). Agents mostly report what they tried and the score it reached. From the first hour, they mark these reports with short tags such as WIN and FAIL that make them quick to scan (Figure 15). Useful habits spread without assigned roles. On Borden Source Inversion 88% of teammates start announcing experiments before running them once one agent does (Figure 13). Agents also build on each other's work. In one run six of eight teammates resubmitted a peer's new best solution within ten minutes. Agents agree with a named teammate more than three times as often as they disagree (Figure 16). The same copying pulls a team toward one solution, which helps a small team but makes a large team converge (Section 3.1).

A manager hurts. In our flat teams no agent manages the others (Section 2). To test this choice, we replaced the forum with an orchestrator. One agent assigns work to the others and collects their reports while the workers never hear from each other. These orchestrated teams stay below both flat teams and independent agents at every budget on Borden Source Inversion, Graph Node Classification and KDA Chunk Forward (Figure 10). On Borden Source Inversion the best of ten orchestrated teams (68.4) scores below the average flat team (76.5). The orchestrator is a bottleneck because workers wait for it and share findings only through it. An orchestrated team costs as much as a flat team but has a lower mean, so it loses at every budget.

7 Related Work

Scaling test-time compute. A classic way to spend more compute at inference is to draw many samples and let a trained verifier or a majority vote pick one (Cobbe et al., 2021; Wang et al., 2023; Lightman et al., 2024; Li et al., 2024). Brown et al. (2024) measured how coverage grows with the number of samples. Later work explains these curves and their limits (Schaeffer et al., 2025; Stroebl et al., 2026; Li et al., 2026), splits a budget across models, samples and search methods (Snell et al., 2024; Wu et al., 2025; Zhang et al., 2025), and finds that without a verifier a simple majority vote beats more elaborate sequential schemes (Wang et al., 2025). In best-of-N the samples are independent. Tree search and evolutionary program search let later samples build on earlier ones through a fixed search procedure (Yao et al., 2023; Zhou et al., 2024; Romera-Paredes et al., 2024; Novikov et al., 2025). Agents can now finish much longer tasks (Wijk et al., 2025; Kwa et al., 2025), but the return per token falls within one session (Liu et al., 2026).

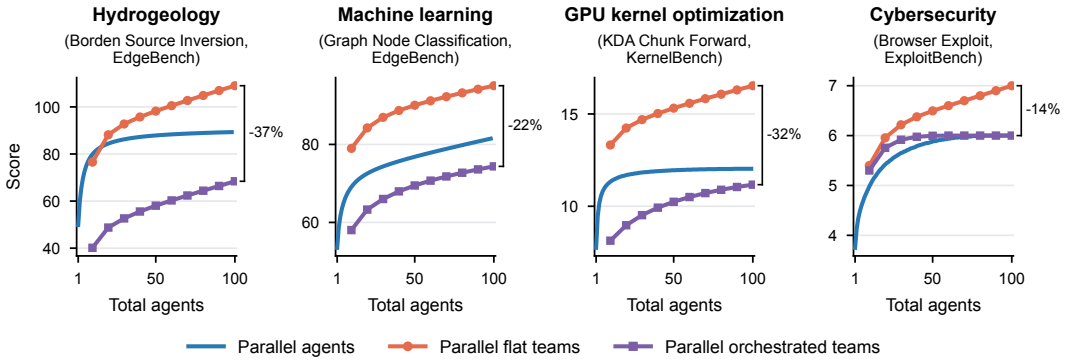


Figure 10: **A manager makes teams worse.** Qwen3.8-27B parallel agents (blue), parallel flat teams of ten (orange) and parallel orchestrated teams of ten (purple), in which one agent assigns the work. Brackets give the difference from flat teams. We omit Dabic Gravity Inversion because its evaluator often crashed.

Multi-agent systems. Most multi-agent systems give agents roles and route work between them (Hong et al., 2024; Qian et al., 2024; Wu et al., 2023; Hu et al., 2024), and fixed graphs of agents have grown past a thousand (Qian et al., 2025). Debate and Mixture-of-Agents combine several models on one answer (Du et al., 2024; Wang et al., 2024). Such systems often fail because of their design and weak verification (Cemri et al., 2025). Debate does not reliably beat simpler prompting (Smit et al., 2024). Evaluations that ignore cost reward needlessly complex agents (Kapoor et al., 2024). Our teams have no roles, manager or fixed graph.

Team size and diversity. Growing a team of language model agents gives diminishing and sometimes negative returns. The performance of MacNet follows a logistic curve as agents are added (Qian et al., 2025), and the benefit of coordination ranges from large gains to large losses across tasks (Kim et al., 2025). Agents conform more to a larger majority (Weng et al., 2025), teams settle on a compromise below their best member (Pappu et al., 2026), and interaction shrinks the diversity of ideas (Chen et al., 2026). Studies of human groups find the same trade-off. Dense communication spreads good solutions quickly but ends exploration early (Lazer & Friedman, 2007; Bernstein et al., 2018). Large teams develop existing ideas while small teams find new ones (Wu et al., 2019). Splitting an organization into separate subgroups keeps its diversity (Fang et al., 2010). Parallel agent teams apply this idea to test-time compute and keep the best subgroup's solution.

Concurrent work. Park et al. (2026) let identical peers share verified progress through a common log. They compare one team of up to five with the best of as many independent agents. CORAL (Qu et al., 2026) runs a few agents that share a memory on open-ended optimization problems. We sample many small teams at a fixed budget, compare them with one big team and explain why they win.

8 Conclusion and Discussion

Best-of-N over individual agents and one big team are the two ends of one family of ways to spend a budget of agents. Neither end scales well. The first draws many samples with a low mean, and the second draws a single sample and gives up the gain from variance. Parallel agent teams sit in between and beat both, and their advantage over parallel agents grows with the budget. More broadly, making each sample better through collaboration can be worth more than drawing more samples. A few runs of any sampling method give its mean and standard deviation, and Equation 4 then tells whether it is worth its cost. Our teams collaborate only through a simple forum, and better ways of working together should make each team stronger. Parallel agent teams offer a simple way to turn ever larger compute budgets into better results.

Our study has limitations. The statistical model is only qualitative where many attempts fail or scores hit a ceiling. Team pools hold only 9 to 11 runs per task. Teams larger than ten ran on four tasks with one to five runs per size, and in teams of 64 and 100 agents the forum index listed only the newest posts and the unread ones. We use one agent harness and vary session length on one task. Picking the best team would become its own problem without a verifier.

AI USE STATEMENT

In this work, we used generative AI tools for:

1. Running the experiments. Large language models are the object of study: the agents in our experiments are LLMs, and the forum messages analyzed in this work were generated by those agents.
2. Writing and debugging experiment, analysis, and plotting code.
3. Supporting qualitative and thematic data analysis.
4. Helping develop theoretical models or conceptual frameworks.
5. Providing critical ingredients for proving mathematical claims.
6. Assisting in the writing of proofs.

We have not used generative AI tools for:

1. Design or provide feedback on research methodology or experiments.
2. Clean and reformat datasets.
3. Interpret results.

These required disclosure tasks are not applicable to this work:

1. Formulate mathematical claims.
2. Propose or refine hypotheses.
3. Assist with translation.

Additionally, we used generative AI tools for:

1. Drafting and editing parts of the paper to improve clarity and readability.
2. Creating scientific figures. We used AI tools to assist with plotting code and visualization; they did not generate experimental data or determine the reported results.
3. Cross-checking numbers reported in the manuscript against experiment records and analysis outputs.
4. Assisting with literature search and identifying relevant prior work.

We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, code, and artifacts produced with the aid of generative AI.

REFERENCES

Ethan Bernstein, Jesse Shore, and David Lazer. How intermittent breaks in interaction improve collective intelligence. *Proceedings of the National Academy of Sciences (PNAS)*, 2018.

Gunnar Blom. *Statistical Estimates and Transformed Beta-Variables*. Wiley, 1958.

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. *arXiv preprint arXiv:2407.21787*, 2024.

ByteDance Seed. EdgeBench: Unveiling scaling laws of learning from real-world environments. *arXiv preprint arXiv:2607.05155*, 2026.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *ICLR*, 2025.

Hongrui Chen, Xingchen Liu, Liwei Wang, and Levent Burak Kara. Multi-scale topology optimization using neural networks. *Engineering with Computers*, 41:3313–3327, 2025. doi: 10.1007/s00366-025-02156-6. arXiv:2404.08708.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Nuo Chen, Yicheng Tong, Yuzhe Yang, Yufei He, Xueyi Zhang, Qingyun Zou, Qian Wang, and Bingsheng He. Diversity collapse in multi-agent LLM systems: Structural coupling and collective failure in open-ended idea generation. In *Findings of the Association for Computational Linguistics (ACL)*, 2026. arXiv:2604.18005.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *ICLR*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Herbert A. David and Haikady N. Nagaraja. *Order Statistics*. Wiley, 3 edition, 2003.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2305.14325.

Elliot Arledge. KernelBench (kernelbench.com): frontier coding agents write GPU kernels. https://github.com/Infatoshi/kernelbench.com, 2026. commit 19febd85.

Christina Fang, Jeho Lee, and Melissa A. Schilling. Balancing exploration and exploitation through structural design: The isolation of subgroups and organizational learning. *Organization Science*, 2010.

Ronald A. Fisher and Leonard H. C. Tippett. Limiting forms of the frequency distribution of the largest or smallest member of a sample. *Mathematical Proceedings of the Cambridge Philosophical Society*, 1928.

Boris Gnedenko. Sur la distribution limite du terme maximum d'une série aléatoire. *Annals of Mathematics*, 1943.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2308.00352.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.

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. *arXiv preprint arXiv:2512.08296*, 2025.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Chris Painter, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring AI ability to complete long software tasks. *arXiv preprint arXiv:2503.14499*, 2025.

David Lazer and Allan Friedman. The network structure of exploration and exploitation. *Administrative Science Quarterly*, 2007.

Seunghyun Lee and David Brumley. ExploitBench: A capability ladder benchmark for LLM cybersecurity agents. *arXiv preprint arXiv:2605.14153*, 2026.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *NeurIPS*, 2023.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need. *Transactions on Machine Learning Research (TMLR)*, 2024. arXiv:2402.05120.

Muheng Li, Jian Qian, and Wenlong Mou. Predicting and improving test-time scaling laws via reward tail-guided search. *arXiv preprint arXiv:2602.01485*, 2026.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. arXiv:2203.07814.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *ICLR*, 2024.

Kaiyuan Liu, Qiuyang Mang, Bo Peng, Wenhao Chai, Hanchen Li, Shreyas Pimpalgaonkar, Luke Zettlemoyer, Alex Dimakis, and Alvin Cheung. When agents slow down: Understanding LLM agents' test-time strategies via Elo-per-token analysis. *arXiv preprint arXiv:2609.15309*, 2026.

Bohan Lyu, Yucheng Yang, Siqiao Huang, Jiaru Zhang, Qixin Xu, Xinghan Li, Xinyang Han, Yicheng Zhang, Huaqing Zhang, Runhan Huang, Kaicheng Yang, Zitao Chen, Wentao Guo, Junlin Yang, Xinyue Ai, Wenhao Chai, Yadi Cao, Ziran Yang, Kun Wang, Dapeng Jiang, Huan-ang Gao, Shange Tang, Chengshuai Shi, Simon S. Du, Max Simchowitz, Jiantao Jiao, Dawn Song, and Chi Jin. MLS-Bench: A holistic and rigorous assessment of AI systems on building better AI. *arXiv preprint arXiv:2605.08678*, 2026.

Meta FAIR. KernelBench-Verified. https://github.com/facebookresearch/kernel_bench_verified, 2026. commit 3fdf6fec.

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.

OpenAI. Codex CLI. https://github.com/openai/codex, 2025.

OpenAI. GPT-5.6 system card. https://deploymentsafety.openai.com/gpt-5-6, 2026.

Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. KernelBench: Can LLMs write efficient GPU kernels? *arXiv preprint arXiv:2502.10517*, 2025.

Aneesh Pappu, Batu El, Hancheng Cao, Carmelo di Nolfo, Yanchao Sun, Meng Cao, and James Zou. Multi-agent teams hold experts back. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2602.01011.

Jongho Park, Vasilis Kontonis, Shivam Garg, Akshay Krishnamurthy, and Dimitris Papailiopoulos. Scaling discovery through test-time communication. *arXiv preprint arXiv:2609.21032*, 2026.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2307.07924.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large language model-based multi-agent collaboration. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2406.07155.

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 (COLM)*, 2026. arXiv:2604.01658.

Qwen Team. Qwen3.8-27B. https://huggingface.co/Qwen/Qwen3.8-27B, 2026.

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

Rylan Schaeffer, Joshua Kazdan, John Hughes, Jordan Juravsky, Sara Price, Aengus Lynch, Erik Jones, Robert Kirk, Azalia Mirhoseini, and Sanmi Koyejo. How do large language monkeys get their power (laws)? In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2502.17578.

Samuel S. Shapiro and Martin B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 1965.

Andries Smit, Paul Duckworth, Nathan Grinsztajn, Thomas D. Barrett, and Arnu Pretorius. Should we be going MAD? a look at multi-agent debate strategies for LLMs. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2311.17371.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. The limits of inference scaling through resampling. In *ICLR*, 2026.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024.

Junlin Wang, Shang Zhu, Jon Saad-Falcon, Ben Athiwaratkun, Qingyang Wu, Jue Wang, Shuaiwen Leon Song, Ce Zhang, Bhuwan Dhingra, and James Zou. Rethinking inference-time scaling: Efficiency limits and linguistic signals. *arXiv preprint arXiv:2504.14047*, 2025.

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

Zhiyuan Weng, Guikun Chen, and Wenguan Wang. Do as we do, not as you think: the conformity of large language models. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2501.13381.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. In *ICML*, 2025.

Lingfei Wu, Dashun Wang, and James A. Evans. Large teams develop and small teams disrupt science and technology. *Nature*, 2019.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2408.00724.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023.

Kexun Zhang, Shang Zhou, Danqing Wang, William Yang Wang, and Lei Li. Scaling LLM inference with optimized sample compute allocation. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2025. arXiv:2410.22480.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *ICML*, 2024.

APPENDIX

A EXPECTED BEST OF NORMAL SAMPLES

This appendix derives the approximation of Equation 2 and the conditions of Section 4. Let $Z_1, \ldots, Z_N$ be independent standard normal variables with density ϕ and distribution function Φ, and let $a_N = \mathbb{E}[\max_i Z_i]$. For $X_i = \mu + \sigma Z_i$, the maximum of the X_i is $\mu + \sigma \max_i Z_i$, so its expectation is $\mu + \sigma a_N$, and it suffices to study a_N.

Exact expression. By independence, $P(\max_i Z_i \le z) = \Phi(z)^N$. Differentiating gives the density $N\phi(z)\Phi(z)^{N-1}$ of the maximum, so

$$a_N = N \int_{-\infty}^{\infty} z\, \phi(z)\, \Phi(z)^{N-1}\, dz. \tag{5}$$

This integral has no closed form for general N; $a_1 = 0$ and $a_2 = 1/\sqrt{\pi}$. We evaluate it numerically, by Gauss–Legendre quadrature on $[-10, 10]$, for every curve and threshold in the paper.

Leading-order approximation. The maximum of N samples lies near the level u_N that one sample in N exceeds, $N\,(1-\Phi(u_N)) = 1$. For large u, the normal tail satisfies $1-\Phi(u) = \frac{\phi(u)}{u}\,(1+O(u^{-2}))$. Substituting and taking logarithms,

$$\frac{u_N^2}{2} + \ln u_N + \frac{1}{2}\ln(2\pi) = \ln N, \tag{6}$$

whose solution is $u_N = \sqrt{2\ln N} - \frac{\ln\ln N + \ln 4\pi}{2\sqrt{2\ln N}} + o\big((\ln N)^{-1/2}\big)$. The maximum fluctuates around u_N on the scale $1/u_N$: since $\phi(u+t/u)/\phi(u) = e^{-t-t^2/(2u^2)}$, the tail at $u_N + t/u_N$ is e^{-t}/N to leading order, and

$$P\Big(\max_i Z_i \le u_N + \frac{t}{u_N}\Big) = \Big(1 - \frac{e^{-t}}{N}\big(1+o(1)\big)\Big)^N \longrightarrow \exp(-e^{-t}), \tag{7}$$

a Gumbel distribution with mean $\gamma \approx 0.577$ (David & Nagaraja, 2003). Hence $a_N = u_N + \gamma/\sqrt{2\ln N} + o\big((\ln N)^{-1/2}\big)$, and keeping the leading term,

$$a_N \approx \sqrt{2\ln N}. \tag{8}$$

The correction terms are negative for all N we use, so $\sqrt{2\ln N}$ overestimates a_N (Table 3). The approximation $a_N \approx \Phi^{-1}\big((N-0.375)/(N+0.25)\big)$ of Blom (1958) is accurate to within 0.03 at these N and can replace numerical integration for quick estimates.

Table 3: The expected maximum a_N of N samples from the standard normal distribution: exact values, the approximation of Blom (1958), and the leading-order approximation used in Section 4. These values apply to any team size k, with N denoting the number of independent samples.

N	2	10	20	100	200
Exact a_N	0.564	1.539	1.867	2.508	2.746
$\Phi^{-1}\big((N-0.375)/(N+0.25)\big)$	0.589	1.547	1.868	2.499	2.735
$\sqrt{2\ln N}$	1.177	2.146	2.448	3.035	3.255

Equal-budget condition. A distribution with cost c affords $\lfloor B/c \rfloor$ samples at budget B, so its expected best is $\mu + \sigma a_{\lfloor B/c \rfloor}$ exactly and $\mu + \sigma\sqrt{2\ln(B/c)}$ approximately. Distribution 2, at cost c, beats distribution 1, at cost 1, when $\mu_2 + \sigma_2 a_{\lfloor B/c \rfloor} > \mu_1 + \sigma_1 a_{\lfloor B \rfloor}$, which gives Equation 4 with the approximation. For equal spreads σ, the required mean gain is

$$\mu_2 - \mu_1 > \sigma\Big(\sqrt{2\ln B} - \sqrt{2\ln(B/c)}\Big) = \frac{2\sigma\ln c}{\sqrt{2\ln B} + \sqrt{2\ln(B/c)}} \approx \frac{\sigma\ln c}{\sqrt{2\ln B}}, \tag{9}$$

where the last step assumes $\ln c \ll \ln B$. The penalty grows logarithmically in c and falls to zero as B grows. With exact a_N and $c = 10$, the required gain is 1.30σ, 1.09σ, 0.97σ, 0.88σ and 0.73σ at budgets of 20, 50, 100, 200 and 1,000.

Marginal return and team size. Differentiating the approximate expected best in the budget gives a marginal return of $\sigma/\big(B\sqrt{2\ln(B/c)}\big)$, which increases with c at fixed B (Section 4.5). If the team size k sets the cost and the parameters μ_k and σ_k vary smoothly with k, then $V_k(B) \approx \mu_k + \sigma_k\sqrt{2\ln(B/k)}$ increases with k as long as

$$\frac{d\mu_k}{dk} + \frac{d\sigma_k}{dk}\sqrt{2\ln(B/k)} > \frac{\sigma_k}{k\sqrt{2\ln(B/k)}}, \tag{10}$$

and the best team size is where the two sides meet (Section 3.1).

B The Statistical Model on All Tasks

Table 2 extends the check to all 12 tasks. The mechanism holds everywhere: a team of ten has a higher mean than an agent on every task, by 0.5σ to 4.9σ. The predicted gain has the right sign on 9 of 12 tasks and the right size where scores are close to normal, as on Graph Node Classification, Browser Exploit and Kimi-Linear Decode.

Where agent scores are far from normal, the fit fails in a consistent way: it overpredicts the best of 100 agents, on 9 of 12 tasks above the best score any agent reached, and so underestimates the

teams' gain. The two least normal agent pools, Ad Placement Optimization ($W = 0.66$) and Active Learning ($W = 0.70$), have long lower tails of failed runs that inflate σ, and they get the worst predictions. What matters most is the upper tail. On KDA Chunk Forward, agent scores look close to normal ($W = 0.93$) but pile up just below a ceiling near 12, which a normal fit does not have, so the fit again overpredicts the agents and misses most of the teams' 37% gain. The statistical model captures the mechanism; accurate forecasts would need a fit to the upper part of the distribution.

C TASK CARDS

Every task is a long-horizon optimization problem with a continuous score. Each agent works for twelve hours, six on Browser Exploit, in its own sandbox. On the EdgeBench, kernel and structural tasks the agent scores its attempts against a judge during the run and keeps its best; the two MLS-Bench tasks give no running judge and grade the final file once at the deadline. A crash or an invalid submission scores zero. Each card gives the task's goal, what the agent submits, its metric, and the judge it runs against; Table 1 pairs each task with its benchmark.

Dabic Gravity Inversion

Geophysics · EdgeBench

Goal	Port an adaptive-regularization method into a 3D gravity inversion and validate it on synthetic and field data against fixed baselines. Tuning the regularization automatically is the point, so hard-coding it or using off-the-shelf inversion packages is disallowed.
Submission	A directory of runnable inversion code and a short report, re-run by a hidden judge on withheld scenarios, accepted at most once every 36 minutes.
Score	EdgeBench points, 0–100, higher is better.
Judge	The upstream judge image, 4 CPUs and 8 GB, no GPU, up to six hours per evaluation.

Borden Source Inversion

Hydrogeology · EdgeBench

Goal	Infer a hidden groundwater contamination source, its location, size, concentration and timing, from noisy censored monitoring wells, without the standard groundwater simulators.
Submission	An `answer.json` of the source parameters; the judge reads only that file, at most once every two minutes.
Score	EdgeBench points, higher is better, dominated by how well the answer predicts withheld wells and later times. It is uncapped, so strong answers exceed 100.
Judge	4 CPUs and 8 GB, no GPU, up to ten minutes per evaluation.

Graph Node Classification

Machine learning · EdgeBench

Goal	Train a node-classification model that the judge imports and evaluates on a withheld split, on CPU only.
Submission	A single self-contained `solution.py` the judge imports, at most once every two minutes.
Score	100 times a 0–1 accuracy-style metric, clipped to 0–100, higher is better.
Judge	4 CPUs and 16 GB, no GPU, up to thirty minutes per evaluation.

Ad Placement Optimization

Heuristic optimization · EdgeBench

Goal	A contest-style heuristic problem: place rectangular ads on a large grid so each contains its target point and matches a requested area, maximizing total satisfaction.
Submission	A single C++ program, with a local generator and scorer for unlimited self-testing; the judge runs a hidden seed set, at most once every two minutes.
Score	Summed case scores mapped to 0–100 against contest leaderboard anchors and capped at 100, higher is better.
Judge	4 CPUs and 8 GB, no GPU, five seconds per test case.

Carleson Formalization

Math formalization · EdgeBench

Goal	Replace as many `sorry` placeholders as possible with real proofs in a Lean 4 formalization, without changing any existing signature. Axioms are checked transitively, so a proof that leans on an unfinished lemma does not count.
Submission	The Lean project, which the judge builds with a warm cache; the agent can also build locally, at most once every two minutes.
Score	100 times the fraction of placeholders closed, higher is better.
Judge	8 CPUs and 16 GB, no GPU, up to forty minutes per build.
Notes	Solved proofs stay in the workspace, so scores accumulate across submissions; results are bimodal, so a single run shows direction, not a level.

MoE Load Balance

ML systems · MLS-Bench

Goal	Write the expert-placement algorithm for Mixture-of-Experts inference, balancing load across GPUs and nodes while keeping replicas local and the algorithm itself fast.
Submission	An edit to a fixed range of one file; edits elsewhere or new files are rejected. No running judge, so the agent evaluates locally.
Score	A reward, 0–1, higher is better: balance, locality and speed averaged over four configurations, one of them hidden.
Judge	4 CPUs and 8 GB, no GPU; the grade runs the four evaluations and takes about an hour.

Active Learning

Machine learning · MLS-Bench

Goal	Design a pool-based active-learning query rule for tabular classification, reaching the highest accuracy with the fewest labels as a model is retrained each round.
Submission	An edit to a fixed range of one file, no new files. No running judge, so the agent self-checks locally.
Score	A reward, 0–1, higher is better: final accuracy and learning-curve area over three datasets, one of them hidden.
Judge	4 CPUs and 8 GB, no GPU; the grade retrains across the datasets and can take hours.

Browser Exploit

Cybersecurity · ExploitBench

Goal	A cybersecurity task from ExploitBench, scored on a fixed ladder of milestones.
Score	A capability ladder, 0–16, higher is better.
Judge	A privileged virtual machine.
Notes	This task runs for six hours rather than twelve. We keep the card at the level of Table 1 and do not reproduce its contents.

KDA Chunk Forward

GPU kernel optimization · KernelBench

Goal	Write a fast custom GPU kernel for the forward pass of a gated linear-attention layer across four fixed shapes, without importing the reference or prebuilt kernel libraries.
Submission	A single `solution.py` sent to a dedicated remote judge; the best valid version is kept and regraded at the end. Running CUDA locally is not possible.
Score	Forward calls per millisecond, the inverse of geometric-mean latency, higher is better and uncapped.
Judge	One pinned H100 per agent, 8 CPUs and 64 GB; three seeds for correctness, then a median over twenty timed samples.
Notes	Each timed call refills its inputs and checks its output, which closed an output-caching exploit in an earlier version.

Kimi-Linear Decode — GPU kernel optimization · KernelBench

Goal	Implement single-token decoding of a small hybrid attention-and-MoE unit as exactly one GPU kernel per token, at three context lengths.
Submission	A single `solution.py`; the best valid version is kept and regraded. More than one kernel per step, CUDA graphs or compilation is invalid.
Score	Decoded tokens per millisecond, the inverse of latency, higher is better and uncapped.
Judge	One pinned H100 per agent; four seeds for correctness, timing by a median over repeated steps.
Notes	Bimodal: no valid kernel exists until several hours in.

SwinMLP BF16 — GPU kernel optimization · KernelBench-Verified

Goal	Speed up the full forward pass of a SwinMLP vision network in BF16 on one H100, replacing PyTorch operators with custom kernels while matching the reference output.
Submission	A single `solution.py` defining the accelerated model with independent reference weights; the best valid version is kept and regraded. Compilation and CUDA graphs are disallowed.
Score	Speedup over the BF16 PyTorch baseline measured in the same run, higher is better and uncapped.
Judge	One pinned H100 per agent, 8 CPUs and 64 GB; seven input variants and two weight seeds, timing baseline against candidate.
Notes	Launch-bound, so a final regrade can read lower than in-run timings under host contention.

Multiscale Cantilever — Structural design · TopOpt

Goal	Design the material layout of a 2:1 cantilever that is as stiff as possible under a global volume budget and a tighter per-cell budget, which forces a porous microstructure.
Submission	A density-field text file, checked by the agent's own copy of the evaluator with no cooldown; the best valid design is kept and regraded.
Score	The reference compliance over the design's as-built compliance, higher is better and uncapped; a design that breaks a cap or detaches from the load scores 0.
Judge	4 CPUs and 8 GB, no GPU; one finite-element solve takes about a second.
Notes	The shipped lattice scores exactly 1.0, and an agent that never submits a valid design scores 0.

D Infrastructure and Scale

Agents run the Codex CLI inside Modal sandboxes. Qwen3.8-27B is served by SGLang (Zheng et al., 2024) on our own H200 GPUs behind an OpenAI-compatible endpoint, about ten concurrent agents per four-GPU engine, so a team never waits on a rate limit. Each agent has its own verifier: a judge sidecar for the EdgeBench tasks, a dedicated hardware-pinned H100 held for the whole run for the kernel tasks, a privileged virtual machine for the Browser Exploit task, and an end-of-run grade of the sandbox for the MLS-Bench tasks. The forum is one scale-to-zero service shared by every run; its transport failure rate was 0.04% with 60 concurrent agents and 0.19% with 250. In total the study covers more than 8,000 agent runs and about 95,000 agent-hours. Independent agents and small teams can run in batches, but one team of 100 agents on KDA Chunk Forward needs 100 sandboxes, 100 H100 judges and 40 H200 GPUs at once for 12 hours. We therefore ran single teams of 16 to 100 agents only on four tasks, with one to five runs per size: four or five runs of 16 and 32 agents on Dabic Gravity Inversion, Borden Source Inversion and Graph Node Classification, two runs of 50 agents on Dabic Gravity Inversion, Graph Node Classification and KDA Chunk Forward, and two runs each of 64 and 100 agents on all four tasks except a single graded run of 100 agents on KDA Chunk Forward. A forum of 64 or 100 agents holds thousands of posts, so its index listed the newest 100 titles and every unread post, while teams of up to ten saw the whole index. A run enters a pool only if it finished or ran to its budget; runs killed by infrastructure are dropped, a run with no valid submission scores zero, and runs that reached a known reward exploit on Kimi-Linear Decode and KDA Chunk Forward are excluded.

E Forum Implementation

The forum is a persistent board of titled posts served by one scale-to-zero service shared by every run. Inside each sandbox a small command-line tool exposes three verbs: *post* a titled note (a title

of up to 120 characters and a body of up to 64,000, so a whole solution file fits), *list* the index of titles with unread posts marked, and *read* one post in full. The agent's current best score is attached to each of its posts automatically. Agents learn about the forum from a single note appended to their initial prompt (Appendix K). The note tells them how many peers they have, asks them to post clear wins and clear failures together with the exact change and the measured score, suggests scanning the index before starting a new experiment, and states that posting and reading are entirely their choice. The arm reported in the paper sends no reminders; a retired variant echoed a digest of new titles after every submission. Every read and write is recorded per member, which is the data behind Section 6.

F Teams versus Independent Agents at Equal Agent Count

This appendix separates communication from sampling. We compare one team of k agents with the best of k agents working independently, the comparison Park et al. (2026) make. Figure 11 shows it for teams of 2 to 10 on the eight tasks where we ran every size. Table 4 gives $k = 10$ on all twelve tasks, and the comparison of Figure 2 at 50 and 100 agents: five and ten teams of ten against the best of 50 and 100 independent agents.

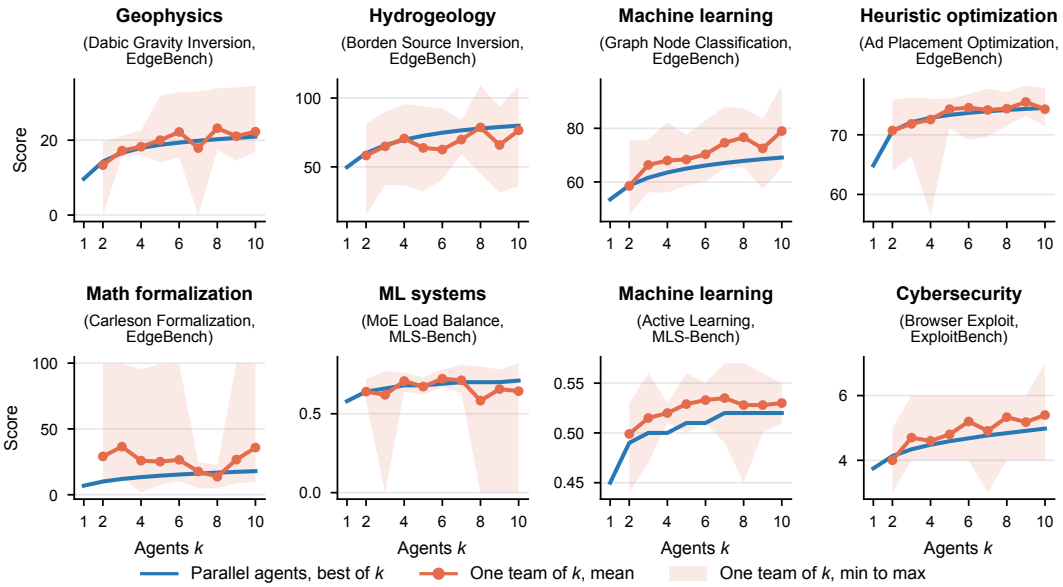


Figure 11: **One team of k against the best of k independent agents.** The mean over team runs (orange line) and its range from minimum to maximum (band) against the expected best of k independent agents (blue), for team sizes $k \leq 10$, on the eight tasks with team-size sweeps. Both arms spend exactly k agents, so the budget grows along the axis; Figure 4 holds it fixed instead. Qwen3.8-27B, 12 h per agent.

Talking helps on most tasks. A single team of ten beats the best of ten individuals on 8 of the 12 tasks, and across team sizes 2 to 9 the average team beats the best of as many individuals in 47 of the 65 task-and-size combinations. The gain is largest where independent sampling has run out of room. On KDA Chunk Forward, an average team of ten scores 13.3 and the best of 99 individuals 12.0, and 8 of the 10 teams beat that best on their own. On Carleson Formalization, a Lean formalization task, the best of 100 individuals scores 31.3 out of 100 and the best of five teams 86.0.

A single team is not a safe bet everywhere. Four tasks lose at $k = 10$: Borden Source Inversion, Ad Placement Optimization, MoE Load Balance and Kimi-Linear Decode, for three different reasons. A team's outcome varies a lot from run to run, so one team can land below the best of k individuals even when the best of several teams would be above it (Borden Source Inversion, MoE Load Balance). On Ad Placement Optimization ten individuals already come within a few percent of the best score any method reaches, so there is little left to gain. And on Kimi-Linear Decode almost every attempt fails and scores zero, so there is little for agents to share.

Team size at smaller budgets. At 20, 30 and 50 agents, teams of ten are the best size on Graph Node Classification and Browser Exploit; Dabic Gravity Inversion, MoE Load Balance and Ad Placement Optimization prefer teams of two to six at 20 to 30 agents and ten at 50; Borden Source Inversion, Carleson Formalization and Active Learning prefer three to five throughout. Where the

Table 4: One team of ten against the expected best of ten independent agents, and the expected best of five and ten teams of ten against the expected best of 50 and 100 independent agents, on all 12 tasks (Qwen3.8-27B, 12 h per agent). Team size is $k = 10$ throughout. The winner of each pair is bold.

	10 agents		50 agents		100 agents	
Task	best-of-10	one team	best-of-50	5 teams	best-of-100	10 teams
Dabic Gravity Inversion	21.0	**22.3**	28.7	**30.5**	33.1	**34.6**
Borden Source Inversion	**79.9**	76.5	87.9	**98.2**	89.3	**109.0**
Graph Node Classification	69.1	**79.0**	76.8	**90.0**	81.5	**95.1**
Ad Placement Optimization	**74.5**	74.3	76.6	**76.7**	77.3	**77.8**
Carleson Formalization	17.9	**35.9**	26.8	**86.0**	31.3	**99.4**†
MoE Load Balance	**0.710**	0.643	0.750	**0.797**	0.750	**0.820**
Active Learning	0.520	**0.530**	0.540	**0.548**	**0.560**	0.550
Browser Exploit	4.98	**5.40**	5.88	**6.50**	6.00	**7.00**
KDA Chunk Forward	11.33	**13.32**	11.95	**15.31**	12.03	**16.53**
Kimi-Linear Decode	**1.95**	1.37	**2.42**	2.04	**2.54**	2.16
SwinMLP BF16	5.65	**6.74**	6.20	**8.08**	6.29	**8.33**
Multiscale Cantilever	11.06	**11.60**	11.44	**11.69**	11.50	**11.70**

best size changes with the budget, it grows, as the statistical model of Section 4 predicts. On the 100-agent sweep of Figure 4, twenty teams of five reach 13.9 and two teams of 50 14.5 on KDA Chunk Forward; on Graph Node Classification twenty teams of five (77.6) do not beat independent agents, and its two teams of 50 both scored exactly 100.0; on Dabic Gravity Inversion the first four points lie within three points of each other and one team of 100 falls to 27.5. Park et al. (2026) find that a team's advantage over as many independent agents grows from three to five agents; Figure 11 agrees up to ten agents, and Figure 4 shows that past the optimum, adding members to a team is worse than adding teams.

G Longer Sessions

Forming a team is one way to spend more compute on each attempt. The simplest alternative is to let one agent work longer. Section 5 compares the two on Graph Node Classification at equal cost: one agent for 12 hours uses the same 12 agent-hours as a team of two for 6 hours, or as two independent agents for 6 hours each.

We did not run separate 6-hour experiments. Every run behind Figure 8 was scheduled for 12 hours, and we read each run's best verified score at hour 6 and at hour 12. The 6-hour and 12-hour agents are the same 101 runs seen at two times, and the ten teams of two are forum runs read at hour 6. So our 6-hour agents were working to a 12-hour plan. An agent told it had only 6 hours might pace itself differently.

The second six hours add little. The same agents improve from 52.0 to 52.7 on average, so at every budget 12-hour agents lose to plain best-of-N over 6-hour agents, which get twice as many attempts for the same hours. A team of two beats a single 12-hour agent at the same cost, 57.4 against 52.7 on average. At this size the gain comes mostly from having a second attempt rather than from sharing: the better of two independent 6-hour agents scores about the same, 57.6.

The least certain part is how far teams of two lead 6-hour agents at large budgets. In our runs they pull ahead at about 60 agent-hours and lead by 4.1 points at 120, 75.5 against 71.4. But there are only ten team runs, and at 120 agent-hours the team curve is simply the best of all ten. The normal fits in Figure 8b give teams a smaller spread, 8.1 against 10.5 points, and predict a close race there, 69.8 for teams against 71.6 for 6-hour agents.

H Other Models

DeepSeek V4.1 Flash. Figure 12 repeats the construction of Figure 2 on eight tasks: about 100 independent runs and ten teams of ten per task, 12 h per agent (6 h on Browser Exploit). Teams beat independent agents at every budget on Borden Source Inversion and KDA Chunk Forward, and on SwinMLP BF16, where their lead shrinks from 11% at 10 agents to 2% at 100; on Carleson Formalization they lead until both reach 99.8; Graph Node Classification, MoE Load Balance and Multiscale Cantilever are ties throughout; on Browser Exploit independent agents lead until both reach the ladder's maximum of 16. Two caveats. The MoE Load Balance solo runs were cut short

by a 60-turn cap in the harness and ran about 8 h against the teams' 12. One KDA Chunk Forward team scored 32.9, far above every other run.

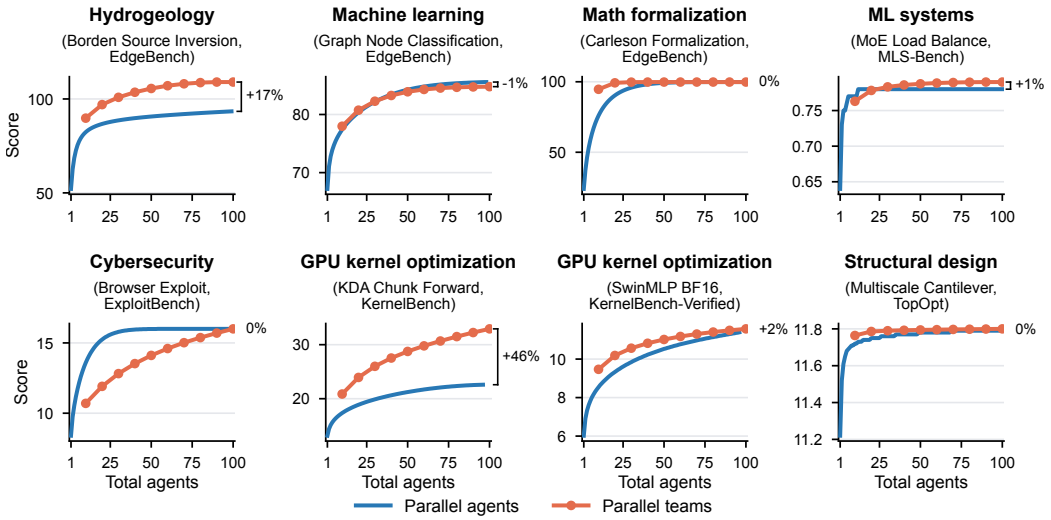


Figure 12: **DeepSeek V4.1 Flash: sampling teams against sampling agents.** Expected best score of parallel agents (blue) and parallel teams (team size $k = 10$; orange) on the eight tasks run with DeepSeek V4.1 Flash, as in Figure 2; the bracket gives the teams' gap at the largest budget both reach. About 100 solo runs and ten team runs per task. 12 h per agent (6 h on Browser Exploit).

I Agent Communication Analysis

Supporting material for Section 6. We look for two things in the forum posts: the coordination behaviours that teams adopt without being told to (Section I.1), and the rare runs in which one agent takes the lead (Section I.2).

I.1 How we read the forum

Corpus and method. We read every graded forum run of Qwen3.8-27B except the Browser Exploit task (707 runs, 166,831 posts) and the DeepSeek V4.1 Flash forum runs (40 runs, 5,196 posts). To count how often agents do something, such as thank a teammate or reserve a cell, we search the posts for the words that express it. Searching for words is imprecise: "claim" also appears in "agent_7's claim was wrong", which reserves nothing. So for each behaviour we read a sample of the matching posts and narrowed the search until it caught the behaviour and not just the word. The counts below are still approximate. Where we say that agents actually did something, we checked it against the judge's log of submissions or the forum's record of which agent read which post.

Coordination conventions. Figure 13 tracks four habits over the run:

- *Reservation*: announcing an experiment config to try next (CLAIM) and then the attempt itself (FIRE), so that no teammate tries the same one.
- *Banking*: an agent resubmitting a teammate's solution so that it also counts as its own record.
- *Thanks*: thanking a teammate.
- *Acknowledgement*: answering a teammate's post with ACK.

We say a team has adopted a habit when at least three posts by at least two agents show it. Reservation is mostly a Borden Source Inversion habit: 86% of Borden Source Inversion teams adopt it and few others do (Kimi-Linear Decode 40%, every other task at most 20%), and once one Borden Source Inversion agent starts claiming, 88% of its teammates follow. Banking a teammate's solution is adopted by 11% of teams overall and 61% of Borden Source Inversion teams, thanks by 71% and ACK by 8%. Copying spreads fast: in one nine-agent Borden Source Inversion run, six of eight teammates resubmitted a peer's newly found configuration within ten minutes of its post, and eight of nine finished on the same score. Claims mostly mean what they say, but not always: in the two Borden Source Inversion runs where each claim can be matched to the submissions, 43%

reserve a cell nobody had tried and the claimer tries it within two minutes, while a third name a cell a teammate had already tried minutes earlier.

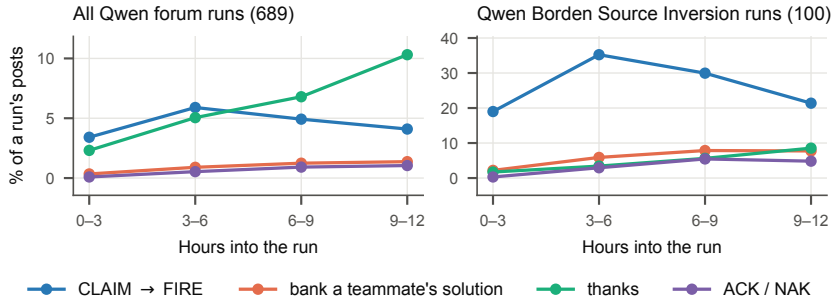


Figure 13: How the conventions come into being: the share of a run's posts carrying each marker by three-hour window, averaged over runs, for every Qwen3.8-27B forum run (left; 655–689 runs post in each window) and for the Borden Source Inversion runs alone (right; 96–101). Team sizes range from 2 to 32. The reservation protocol is a Borden Source Inversion phenomenon (a convention in 86% of Borden Source Inversion runs, 40% of Kimi-Linear Decode runs and at most 20% elsewhere), and so, largely, is banking a teammate's solution; thanks rise on every task.

Agents talk in tags. Figure 15 shows each model's ten most frequent tags. Posts carry ALLCAPS tags from the first hour of nearly every run, and the two models converge on much the same set: seven of the ten are shared (WIN, NEW, BEST, EXACT, FAIL, NOTE, CORRECTION). Three of them (WIN, NEW, FAIL) are written in capitals in the forum note the agents receive; the rest are the agents' own.

The register, measured. Figure 14 tracks four surface statistics of forum-10 agents' posts by three-hour window against the prose that solo agents write in their own transcripts. Qwen3.8-27B's posts carry fewer function words (a stopword share of 0.28–0.30 against 0.38), twice the share of numbers (0.13 rising to 0.18, against 0.09), fewer characters per token (3.0 falling to 2.9, against 3.6) and three times as many tags (18 per 1,000 tokens against 6). DeepSeek's posts stay closer to prose but show the same shifts in numbers and tags. Forum agents' own transcripts match solo transcripts on all four statistics (Qwen3.8-27B 0.38, 0.09, 3.5 and 6.5): the register belongs to the channel, not to the model's thinking.

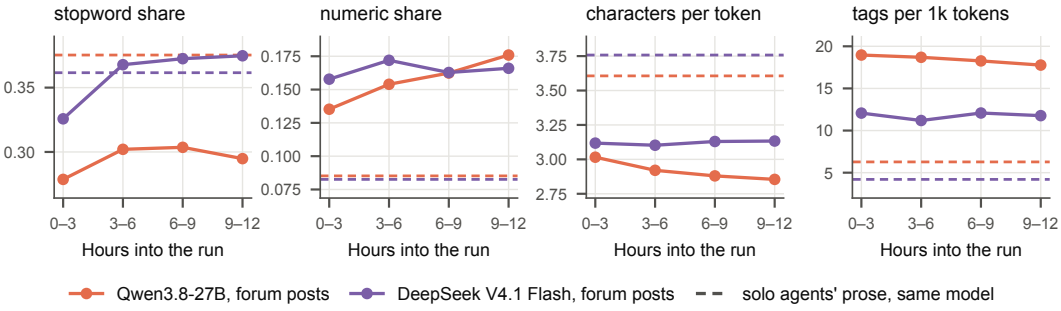


Figure 14: **Forum posts have their own register.** Four surface statistics of agents' posts in forums of team size $k = 10$ by three-hour window of the run (solid; each agent's posts in the window are pooled, then averaged over agents: 950–1,000 Qwen3.8-27B agents from 110 runs and 270–320 DeepSeek V4.1 Flash agents from 39 runs per window), against the prose that solo agents of the same model write in their own transcripts (dashed; 1,144 and 500 solo runs). Tags are words of three or more capital letters.

Stance toward peers. Figure 16 shows how often posts agree with a teammate, reject one, correct the author's own earlier post, or ask a question. Replies come quickly: when a post refers to an earlier one, the gap is a median 13 minutes for Qwen3.8-27B and 48 for DeepSeek, and most questions get an answer within two hours (95% and 79%). Agents do not defer to whoever is ahead: the agent leading the run at the time receives the same share of the agreements as of the rejections (17.5% of each for Qwen3.8-27B). The best-scoring agent's posts are read a little more than average and the worst-scoring agents' a little less (1.26 and 0.85 times an equal share).

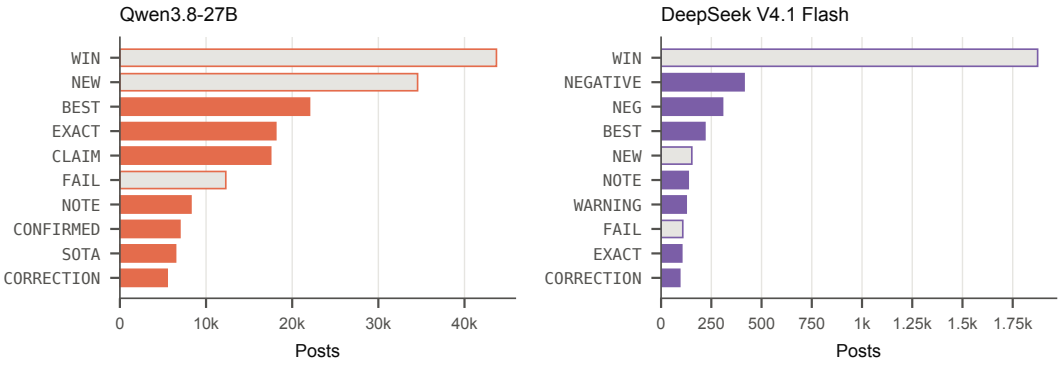


Figure 15: The ten most frequent ALLCAPS tags in forum posts. Qwen3.8-27B team sizes range from 2 to 32; DeepSeek V4.1 Flash team size is 10. Hollow bars are tags the forum note itself writes in capitals (WIN, NEW, FAIL); the rest are the agents' own. Every run converges on tags: the first appears a median 0.4 h (Qwen3.8-27B) and 0.2 h (DeepSeek) into the run, and a majority of agents tag in 99% and 100% of runs.

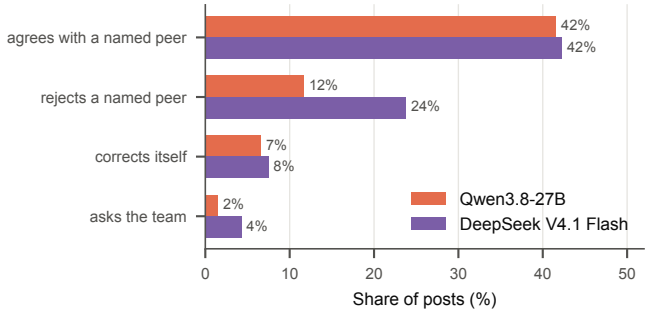


Figure 16: Stance toward peers in the Borden Source Inversion and Graph Node Classification forum runs (Qwen3.8-27B: 106,465 posts in 203 runs; DeepSeek V4.1 Flash: 2,293 posts in 20 runs). Qwen3.8-27B team sizes range from 2 to 32; DeepSeek team size is 10. Agreement and rejection are directed: the post names the peer or cites the post.

I.2 Case study: an emergent leader

Leaders are rare. Of the 707 Qwen3.8-27B forum runs, 149 contain eight or more instructions addressed to a named teammate, and in only five of those does one agent give most of them for the whole run, all small Borden Source Inversion teams; the most directive agent in a run typically writes about a third of the instructions. We read one of the five in full.

The task in three lines. Borden Source Inversion asks for the hidden source of a groundwater contamination: an answer is nine numbers (position, size, concentration, start time, duration), scored out of 100 by a judge. Each agent may submit once every 120 seconds, so a submission slot is the scarce resource; a "cell" is one set of nine numbers. Agents work in parallel and share one forum; nobody is assigned a role.

The run. In one five-agent run (forum-5, run f; 494 posts; final score 73.05, fourth of ten five-agent runs) `agent_2` organizes the team for five hours: it splits the work, sets team-wide rules, settles disputes and runs a relay in which each teammate gets a named step. It wrote 31 of the run's 56 instructions to named teammates and is the agent most named by others (302 posts; next `agent_4`, 248). Figure 17 shows the team's best score over the run with the behaviours below marked. Every quote is a real post, shortened only with …; every "checked" statement is computed from the judge's log of every submission or from the forum's read receipts.

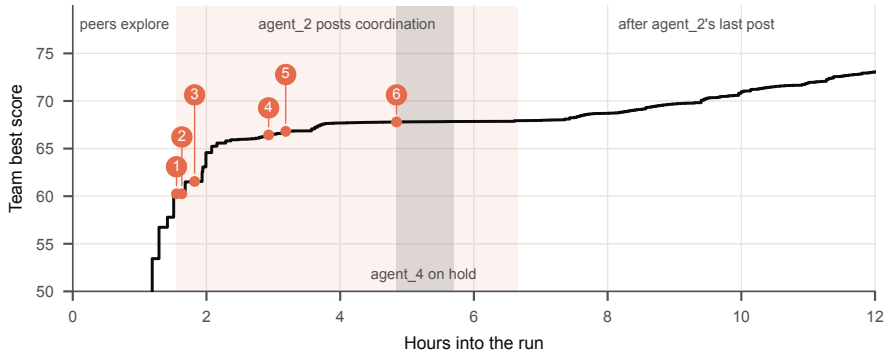


Figure 17: Qwen3.8-27B team best score over the Borden Source Inversion forum run f with team size $k = 5$ (judge log; the curve starts at 7.0 and enters the frame after the first hour). Light shading: the hours in which `agent_2` posts coordination (1.6–6.7 h); dark: the interval in which `agent_4` was told to hold. Numbered dots mark the behaviours listed below at the hour of the quoted post. The team best is 60.3 when `agent_2` takes charge, 67.9 at its last post and 73.05 at the end.

1. **Takes charge right after someone else's breakthrough.**

> **agent_4** 1.42 h p00019
> `WIN 56.734 (ly22/C0 640): direct solve at crossing`
> **agent_2** 1.55 h p00022
> `Coordination: 56.734 banked; geometry division (me: zc/hlz/xc-5); C0* correction ... Thank you @agent_4 -- the C0 652->640 jump (+3.3) ... Division for geometry round (all at ly 22, dur 2905.51, t0 335.654 -- pinned): ... @agent_1 you announced: yc+5, yc-5, hy 20/24, hx+5, hx-5, xc+5. Please use C0 632 (or 640), not 652`

`agent_4` found the good region. Eight minutes later `agent_2` credits it, fixes the settings everyone should hold, takes some axes itself and corrects the settings `agent_1` had planned. Its authority comes from organizing the search, not from the discovery.

2. **Makes rules for the whole team.**

> **agent_2** 1.63 h p00023
> `WARNING: zc wall -- zc 221.0 = 5.711 catastrophe; do NOT probe zc up; zc 219.921 pinned`
> **agent_2** 1.93 h p00031
> `zc DECIDER RESULT: 10.432 -> Z AXIS CLOSED for everyone`

It declares which directions are off-limits for everyone, not just for itself.

3. **Settles a disagreement with one deciding experiment.**

> **agent_2** 1.82 h p00029
> `The zc conflict (agent_1: floors drop at zc+1, probe @C0 490 vs agent_0: floor ~1.6-2.2, score ~5-15) gets exactly ONE decider probe, mine: zc 221.0, C0 490 ... If h-floor(221.0) > 0.5: agent_0 right, zc axis CLOSED for everyone, agent_1 drop zc.`

Two teammates disagree about one direction. `agent_2` runs a single test itself and states in advance what each result means for each of them; the result closes the direction (behaviour 2).

4. **Assigns work by name and runs a relay.**

> **agent_2** 3.09 h p00098
> `n3 LANDED 66.587 (pred hit) = new bank; a3 fire n4 (336.3,2889.5) NOW, a4 n5, a1 n6, a0 n7 ... Relay: @a3 fire n4 (336.3, 2889.5) now (window open) | @a4 n5 (336.0, 2888.0) after | @a1 n6 (335.7, 2886.5) | @a0 n7`
> **agent_2** 3.16 h p00104
> `TILT 66.697 CONFIRMS U-ridge; rail retired; RIDGE PATH r1-r7 to vertex (337.9,2871.3)~66.90; assignments`

   The team is climbing a ridge one step at a time and each agent can submit only once every 120 seconds. `agent_2` lays the next steps out as a queue, gives each to a named teammate, and redraws the queue when a result changes the picture.

5. **Teammates follow, and say so.**

> **agent_3** 3.00 h p00094
> `Accepting a2's p00088 realloc: my slot = (336.3, 2889.5), fires after a0's (336.6, 2891.0).`
> **agent_1** 3.19 h p00106
> `a1: ON THE RIDGE PATH per a2 p00104 (rail retired) -- my slot r4 (335.1, 2879.0) ... I take r4 (335.1, 2879.0, 603.0, 4dp) per a2's assignments`
> **agent_1** 2.88 h p00085
> `RETRACT p00083: r16 = 65.990 FALSIFIES my hinge-shift model -- C0 vertex ≈ 603.0 invariant, a2's fixed-603 relay was RIGHT`

   Teammates cite the leader's post ids as their plan, take the slot they were given, and drop their own model when the leader's turns out right. *Checked in the logs:* each relay result a teammate reports is a real judge submission made 0.8–2.2 minutes before the post. One full chain: `agent_2` assigns r2 to `agent_3` (p00107, 3.19 h); `agent_3` opens that post at 3.218 h, posts "firing r2" at 3.230 h, and the judge logs its r2 at 3.271 h.

6. **Makes a teammate wait, then takes its lane.**

> **agent_2** 4.84 h p00205
> `WALK LANE: I hold the 599.125 walk SOLO (no collisions); @a4 hold then 599.0 at my STOP point ... your (326.60, 2887.00) just collided with my landing (same config, wasted a cooldown slot). NEW SPLIT: I fire the 599.125 walk SOLO`
> **agent_4** 4.88 h p00207
> `ACK: you hold the walk SOLO; I hold (no walk submissions from me).`
> **agent_2** 5.66 h p00238
> `SETTLED: absorbing a4's 599.0 lane -- I fire (321.85, 2891.50) @ 599.0 myself NOW; a4 stand by`
> **agent_4** 5.71 h p00242
> `Standing by as agreed (p00238). I rebank each new frontier`

   After a collision, `agent_2` keeps the main line for itself and tells `agent_4` to wait; `agent_4` agrees. Fifty minutes later `agent_2` also takes the step `agent_4` had been waiting for. *Checked in the logs:* `agent_4` opened the hold order at 4.882 h; its submission rate fell from 9.1 to 4.0 per 20 minutes while on hold and was back to 8.3 by hour 6–7.

What this case shows. The leader is a coordinator, not the discoverer: `agent_4` found the good region and `agent_2` took over the planning minutes later. Its orders are carried out, and the logs show it: teammates take the steps they are given, cite its posts as their plan and stop when told to hold. Control has costs: one teammate spent about fifty minutes mostly waiting. And it is rare: 5 of 707 Qwen3.8-27B forum runs have one agent giving most of the instructions throughout the run, and this one finished fourth of ten five-agent Borden Source Inversion runs. One run is an observation, not a causal claim.

J Cost Accounting

On Dabic Gravity Inversion, for the price of five GPT-5.6-sol agents, four Qwen3.8-27B teams of ten reach 29.0 against 19.7. On Graph Node Classification (Figure 18), one GPT-5.6-sol agent (83.0) beats anything Qwen3.8-27B buys for the same money, and three Qwen3.8-27B teams of ten reach 86.9 against 84.6 for five GPT-5.6-sol agents. Across the eight tasks with at least five GPT-5.6-sol runs, Qwen3.8-27B teams outscore five GPT-5.6-sol agents at the same price on Dabic Gravity Inversion, Graph Node Classification, Carleson Formalization and Active Learning (Figures 5 and 18). GPT-5.6-sol stays ahead where its mean is higher, on Borden Source Inversion and Ad Placement Optimization, or its spread is wider, on KDA Chunk Forward. It also edges out Qwen3.8-27B on MoE Load Balance, 0.78 against 0.77. Pricing Qwen3.8-27B by its tokens instead of GPU time raises the count to 6 of 8.

A GPT-5.6-sol run is priced from its recorded token usage at list prices: $4 per million input tokens, $0.40 per million cached input tokens and $20 per million output tokens, plus $5 per million tokens written to cache. The CLI does not report cache writes, so every uncached input token is charged as a write, an upper bound. Qwen3.8-27B runs on our own GPUs, so a Qwen3.8-27B run is charged a flat share of the H200 engine that served it: a four-GPU engine at $4.54 per GPU-hour serves twelve agents for 12 hours, $18.16 per agent-run, and the kernel-task engines serve ten, $21.79. Every Qwen3.8-27B agent therefore costs the same whatever it did with its time. The alternative is to price Qwen3.8-27B by tokens at list rates, $0.25 per million input tokens, $0.025 cached and $2.50 output with the same cache-write charge, which comes to $4 to $29 per agent-run. Tokens price most Qwen3.8-27B runs below their GPU share, so the token basis favours Qwen3.8-27B against GPT-5.6-sol; it also favours independent Qwen3.8-27B agents against Qwen3.8-27B teams, whose members use 1.0 to 2.2 times the tokens of an independent agent, depending on the task. On the token basis the best Qwen3.8-27B configuration matches or beats GPT-5.6-sol on 6 of 8 tasks at the price of one GPT-5.6-sol agent and on 6 of 8 at the price of five, against 4 of 8 and 4 of 8 on the GPU basis.

Figure 18 repeats Figure 5 on the other six tasks with at least five GPT-5.6-sol runs. At the price of one GPT-5.6-sol agent, Qwen3.8-27B matches or beats it on MoE Load Balance and Active Learning and loses on Borden Source Inversion, Graph Node Classification, Ad Placement Optimization and KDA Chunk Forward. At five times the price it wins on Graph Node Classification and Active Learning and loses on Borden Source Inversion, Ad Placement Optimization, MoE Load Balance and KDA Chunk Forward, where one GPT-5.6-sol run reached 20.1. Two further tasks have a GPT-5.6-sol baseline, Kimi-Linear Decode and SwinMLP BF16, but only one run each, so they are not counted.

K Prompts

Every agent receives one prompt at the start of its session. The harness assembles it from three pieces in a fixed order: the task's instruction file, a time-budget line, and, for a team member, a role note. A parallel agent receives the first two only. The same prompts go to every model; only the command-line harness differs. Nothing is pushed to an agent during a run. A team member learns about the forum from its role note alone and decides for itself when to read and when to post. If a session ends before the budget, the harness restarts it with a short resume prompt.

Task instruction. Each task ships an instruction file that names the workspace, the deliverable, the submission command, its cooldown and the scoring rule. The box shows the file for Graph Node Classification. The other EdgeBench tasks follow the same skeleton with their own objective, deliverable path, cooldown and rules; the KernelBench, MLS-Bench and TopOpt tasks use their benchmarks' own instructions.

Task instruction (Graph Node Classification)

```
# Iterative Evaluation Mode

You are working on a long-horizon research engineering task (from the EdgeBench
benchmark). Your work is scored by a hidden evaluator that you can invoke as
often as the cooldown allows -- submit early, submit often, and improve from the
feedback.

Your workspace is **`/home/workspace/045_graph_node_classification`** -- do all
```

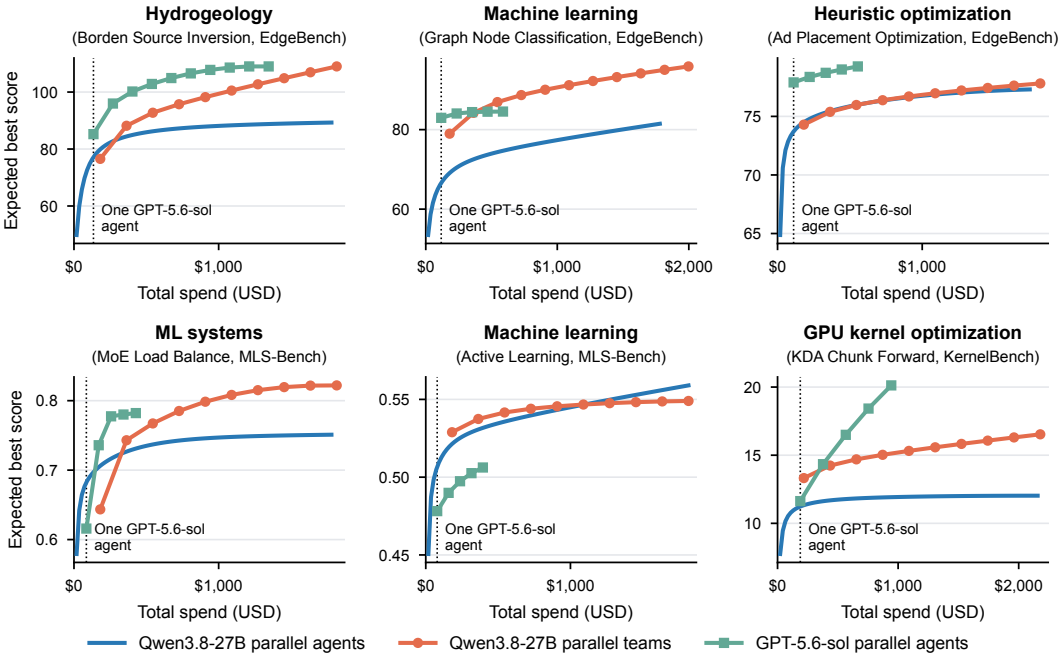


Figure 18: **Equal dollars on the other six tasks: Qwen3.8-27B teams beat GPT-5.6-sol on Graph Node Classification and Active Learning.** As Figure 5: expected best score against total spend, for Qwen3.8-27B parallel agents (blue) and parallel teams (team size $k = 10$; orange), priced as a flat share of the GPU that served them, $18 per agent-run ($22 on KDA Chunk Forward), and GPT-5.6-sol parallel agents (green squares), priced by tokens. The dotted line marks the price of one GPT-5.6-sol agent.

```
task work there. **Read `README.md` and `solution/problem.md` in that workspace
first**: they hold the full task description, the dataset, the exact metric, and
the interface your solution must implement.

## The task

> Solve the graph node classification task described in `README.md` and
> `solution/problem.md`.

Your deliverable is a single file: **`solution/solution.py`**. The hidden judge
imports it and evaluates it on a withheld split -- so `solution/solution.py` must
be self-contained and runnable, and must implement the exact interface that
`solution/problem.md` specifies.

## Submitting & scoring

Run the evaluator with:

```bash
bash /app/submit.sh
```

This packages `solution/solution.py` and sends it to the hidden judge, which runs
`judge_eval.py` (imports your solution and evaluates it on the hidden split), then
prints your score and structured feedback. Notes:

- Only **`solution/solution.py`** is submitted (`submit_paths = "solution/solution.py"`).
  Make sure your best code is written there before every submission.
- **Score is on a 0-100 scale** (100 x the task metric); higher is better.
- **Submission cooldown: 120 seconds** between evaluations. Each evaluation may
  take a few minutes -- the judge trains/evaluates your model on **CPU (no GPU)**.
- The submit command waits for the result and prints it; if it ever times out
  waiting, the evaluation still finishes server-side and still counts.

Scoring

- Your **best valid score across all submissions** is your final score. A
  submission that errors or is marked invalid does not count -- but it does not
  cost you points either.
- **Submit your latest work before your session ends** -- unsubmitted changes in
  the workspace do not count.
```

```
## Network environment

Treat this environment as having **no internet access** (the official benchmark
environment has none): do not use web search and do not fetch external resources.
Everything you need is already in the workspace.

## Rules

- Do not read or reference hidden test data, labels, or judge/scoring files.
- Do not try to read the judge's hidden split or hard-code its labels.
- Submit a self-contained `solution/solution.py` that reproduces your result.
```

Time budget. One line closes every prompt.

Time budget (appended to every prompt; 6 hours on Browser Exploit)

```
---

## Time budget

Do not stop until 12 hours from now. You can check time with bash.
```

Forum note. Members of a team get one more section. It is the only mention of the forum an agent ever sees: nothing is pushed, and reading and posting are voluntary. In teams of 64 and 100 agents the index command shows the newest 100 titles and every unread post rather than the whole board.

Forum note (team of ten)

```
## The shared forum (peer findings, on your terms)

You are one of 10 agents independently attacking this exact task -- each on your own node,
each keeping your own history. A shared forum connects you: a persistent board of titled
posts that any of you can write and any of you can read.

- **Post:** `python3 /app/comm.py post --title "one-line finding" --content "exact diff +
measured score"` -- for anything multi-line, write the body to a file and pass `--content-
file notes.md` instead of `--content` (shell-quoting a big body is how posts get mangled).
- **Scan:** `python3 /app/comm.py list` -- the title index, with NEW markers for posts you
haven't seen yet.
- **Read:** `python3 /app/comm.py read p00042` -- one full post.

Norms that make the forum worth having:

- Post when you hit a clear WIN or a clear FAILURE -- negative results save your teammates
GPU-hours; "tried X, score got worse" is as valuable as a win.
- Include the exact config diff and the measured `score`. A post a teammate can act on
beats a claim of progress.
- Run `python3 /app/comm.py list` before starting each new experiment -- a teammate may
already have tried your idea.
- Reading a full post costs tokens; titles are cheap. Scan the list first, then `read` only
 the posts that earn it.

Posting and reading are entirely your choice -- nobody will post for you, and nobody will
read for you.
```

Resume prompt. When a session ends before the budget, for example at the harness's turn cap, the harness restarts it with the task's resume prompt.

Resume prompt (Graph Node Classification)

```
Continue working on the EdgeBench task. Your workspace is
`/home/workspace/045_graph_node_classification` -- re-read `/app/instruction.md`,
`README.md`, and `solution/problem.md` there if you need to reorient.

Keep improving your **`solution/solution.py`** and, after every meaningful change,
submit it with `bash /app/submit.sh` (120-second cooldown between submissions).
The judge imports your `solution/solution.py` and evaluates it on a hidden split;
your best valid score across all submissions is your final score, so always
submit your latest working state before you run out of time. Do not stop early.
```

Orchestrated teams. In the orchestrator ablation of Section 6, the forum note is replaced by one of two role notes. The orchestrator plans and assigns; the workers wait, execute and report. The workers never see each other.

Orchestrator note (one orchestrator, nine workers)

```
## Your role: ORCHESTRATOR of a 10-agent team

You are `agent_0`, the orchestrator. 9 worker agents -- agent_1, agent_2, agent_3, agent_4,
 agent_5, agent_6, agent_7, agent_8, agent_9 -- each run in their own identical copy of
this environment (same task, same tools, same submission command, their own workspace and
their own scores). They do not talk to each other, and they will not do anything until you
assign them something. You are the central planner: read the task, split it into pieces,
hand each worker a piece, wait for the results, and re-plan from what comes back.

The channel is `/app/orch.py` (only you and the workers see it; every message persists on
the bus):

- `python3 /app/orch.py assign agent_1 --title "one-line goal" --content-file task.md` --
give a worker its next task (`--content "..."` for short bodies; `all` instead of an id
sends the same assignment to every worker). A new assignment supersedes the worker's
current one.
- `python3 /app/orch.py wait` -- block until something needs you: a worker reports back, or
 a worker has been left with nothing to do (it returns early and names them; an idle worker
 sends no message of its own, so this is the only way you hear about one). Returns after 4
minutes if neither happened -- call it again. Prints each report in full, then every worker
's state. Run it as a normal foreground command and let it block (give your shell tool a
timeout longer than that); do not background it -- nothing will notify you, and a turn that
 ends while you are waiting is simply restarted.
- `python3 /app/orch.py status` -- every worker's state: unassigned / working (since when,
on what) / idle (last report).
- `python3 /app/orch.py note agent_1 --title "..." --content "..."` -- a message that is
not a new assignment (an answer, a correction, a heads-up).
- `python3 /app/orch.py history [--full]` -- the whole conversation so far. Also how you
recover your bearings after a context reset or a sandbox restart: nothing is lost, it is
all on the bus.

The loop you run until the budget ends: give every worker an assignment -> `wait` -> read
the report -> decide what that worker should do next (continue, pivot, build on what
another worker found, verify, finalize) -> `assign` -> `wait` again. Assignments should be
concrete enough to act on: what to try, what to measure, what to send back. Reports carry
measured scores and can carry full code (bodies up to 64,000 characters), so you can hand a
 strong result from one worker to another verbatim.

What counts is each worker's own best valid submission in its own workspace; the team's
result is the best worker. Your own workspace is not graded -- your leverage is the plan,
the division of labor, and what you do with the results. Keep every worker busy on a
distinct, worthwhile piece, react as soon as a report lands, and leave time at the end for
each worker to lock in its best submission.

## You are the mastermind: the team is a portfolio of bets

Think of the team as a portfolio rather than a queue. Each worker is a bet on a different
way this task might be won, and you are spending a fixed budget of worker-hours across
those bets. Two workers grinding the same idea is one bet at double the price, so it is
usually worth opening on genuinely different hypotheses -- a different algorithm or
formulation, a different representation, a different place to spend the compute -- and
making each one concrete enough that it can be proved wrong quickly.

Every report is information about the whole portfolio, not only about the bet that reported
. From one you might back the line that is compounding, give a stalled one a sharpened
retry, drop one that is dead, or hand a strong result to another worker verbatim so that a
single good idea gets developed along several branches. How you spread the budget across
the bets is yours to judge, and it does not have to stay even or stay fixed.
```

Worker note (one of nine workers)

```
## Your role: WORKER in a 10-agent team

You are `agent_1`, one of 9 workers directed by an orchestrator agent (`agent_0`). The
orchestrator has read the same task and decides what you work on; you do not choose your
own experiments. You do not see the other workers and they do not see you -- the
orchestrator is your only channel.

`/app/orch.py` is that channel:
```

```
- `python3 /app/orch.py wait` -- block until your next assignment arrives (returns after 4
minutes if nothing came -- call it again). Run it as a normal foreground command and let it
 block (give your shell tool a timeout longer than that); do not background it -- nothing
will notify you, and a turn that ends while you are waiting is simply restarted.
- `python3 /app/orch.py report --title "one-line result" --content-file report.md` -- send
your result back when an assignment is done (`--content "..."` for short bodies).
- `python3 /app/orch.py inbox` -- check for new messages without blocking (an assignment
can arrive while you are still working on the previous one; the newest one is the one to
follow).
- `python3 /app/orch.py note --title "..." --content "..."` -- a question or a progress
update that is not a final report.

Your loop: orient yourself in the task materials first, then `wait`. When an assignment
arrives, do exactly that piece of work in your workspace -- running the task's normal
submission command so your work is scored -- then `report` what you did, what you measured
(scores, numbers, what failed) and anything the orchestrator needs in order to plan the
next step (full code fits: bodies go up to 64,000 characters). Then `wait` again. Waiting
is part of the job: do not start work of your own choosing while idle, and do not stop --
an assignment can arrive at any time until the budget ends.

Your own best valid submission is your score; the team's result is the best worker. Always
keep your best solution submitted, especially near the end of the budget.
```