SELF-EVOLVING AGENT TEAMS FOR OPEN-ENDED CREATION

Anonymous authors
Paper under double-blind review

ABSTRACT

One agent is too slow to build something large. A natural next step is a bigger team, yet we find that more agents, without structure, deliver less. They collide, repeat work, and leave parts of the job with no one responsible. In any task, new demands appear as work progresses, and the team has to reorganize itself to handle them. We present Self-Evolving Agent Team, a multi-agent system that reorganizes itself as it works. As the work progresses, a dedicated agent watches the team, finds what is falling behind, and reshapes structure and roles in response. Like human teamwork, the system rests on three backbones: a reporting tree for accountability, a messaging layer for coordination, and a shared codebase for integration. Across software builds, knowledge work, and standard agent benchmarks, ours outperforms fixed teams of the same size and improves as it grows.

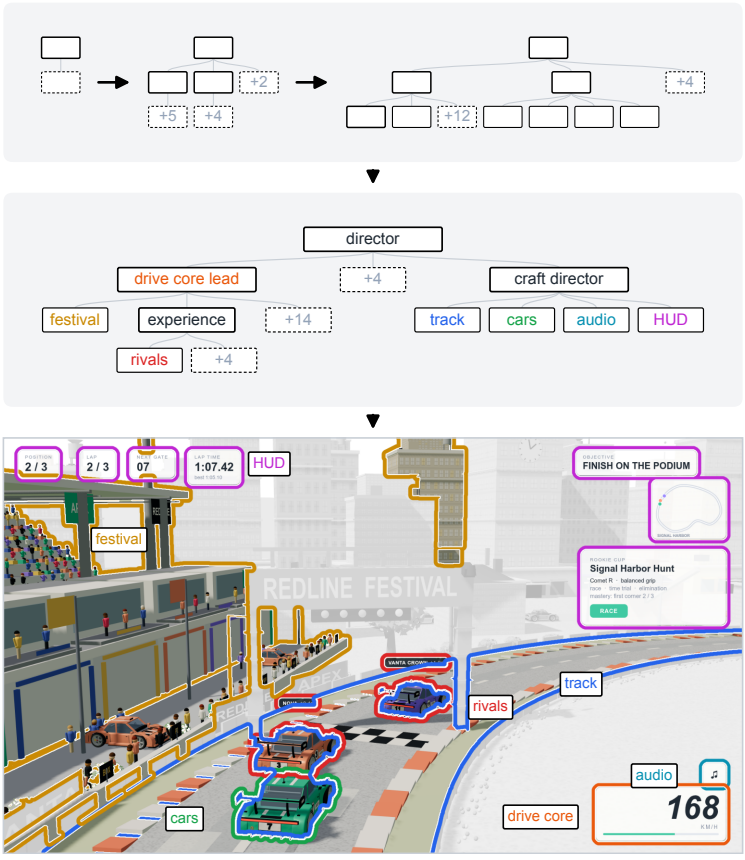


Figure 1: **A team that grows into its work.** In our Self-Evolving Agent Team, a run starts with one agent and ends with whatever team the work needed. *Top:* the team grows as the artifact shows what is missing. *Middle:* the final team; a separate agent, the Architect, watches it and reshapes it between rounds. *Bottom:* what it delivered; each outlined part is colored by the position that built it.

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.

1 INTRODUCTION

Ask thirty AI agents to build a racing game together. After some time, what they deliver looks real: a title screen, a car on a city track, a checkpoint ring ahead. Then the game denies the checkpoint while the car is sitting on it. It announces an impact with nothing nearby. Twenty seconds in, the car is stopped dead on a block in the middle of its own racing line, and there was never a rival to race (Figure 5). Now ask only two agents for the same game. They work longer, but they deliver a racer: three cars on a grid, standings, solid collisions, and a race that keeps going.

This is what we found in agentic open-ended creation, where someone gives a short request and expects a complete artifact in return. "Build me a racing game." "Make a fitness app." "Put together the quarterly report." The work behind each of these is too large for one agent to finish alone, and in a human team the natural answer would be to bring in more people.

We found that more agents deliver less. The thirty agents are not idle; they are busy in ways that cancel each other out. One agent builds a menu while another rebuilds it from scratch. They edit the same file and overwrite each other's changes. A car is drawn but nobody connects it to the physics, so it sits on the track and never moves. The pieces are all there, but nobody puts them together. Each agent, on its own, can write working code. What is missing is someone to divide the work, track what has been done, and make sure every part of the job has someone responsible for it.

Human teams solve this problem naturally. A manager watches the work, sees what is falling behind, and reassigns people or brings in new ones. The team at the end of a project rarely looks like the team at the start. We bring the same idea to multi-agent systems. We present Self-Evolving Agent Team, a system where a dedicated agent watches the team as it works, finds what is falling behind, and reshapes the team in response. It can add new agents, remove ones that are no longer needed, or change what each one does. The team is never designed; it is grown from the work.

The system rests on three backbones: structure, communication, and shared workspace. (1) The team is structured as a tree. Each spot holds one agent with a description of its responsibilities, and every spot reports to the one above it. (2) Agents communicate through direct messages and group channels, so agents working on related parts can coordinate directly. (3) All agents write to a single repository, each on its own branch. Their work merges into one artifact, and if two agents changed the same file, the conflict is caught instead of silently overwritten.

A run starts with one agent holding the whole job. It delivers a first version, and then the Architect, the agent in charge of the team, reviews what was built. If something is broken or missing, it figures out why and changes the team: it can bring in a new agent, remove one, move one to a different part of the tree, or rewrite what one is responsible for. This cycle repeats every round. No part of the team is decided before the work begins; the work itself decides what the team should look like.

We want every agent on the team to contribute something unique to the final artifact (Figure 2). In most existing systems, this is not what happens. Adding an agent means adding another attempt at the same task. Self-consistency (Wang et al., 2023), multi-agent debate (Du et al., 2024; Liang

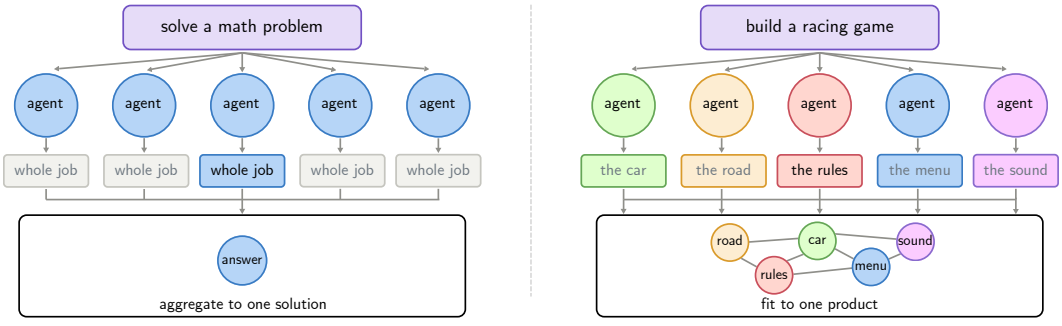


Figure 2: **More agents can mean more samples or more contributors.** Left: in self-consistency, debate, and mixture-of-agents, every agent attempts the whole job and the outputs are aggregated, so the agents are interchangeable. Right: what we want is that each agent actually contributes to something in the final result; that must fit into one artifact, so an added agent brings a new capability rather than another attempt.

et al., 2024) and mixture-of-agents (Wang et al., 2025a) all follow the logic of scaling test-time compute (Snell et al., 2025; Li et al., 2024): more samples, better odds. This works when the task has one answer to converge on, but when the task is to build something, there is no answer to vote on. The artifact is an assembly of parts that must fit together, and recent work confirms that sampling does not help here (Pappu et al., 2026; Bertalanič & Fortuna, 2026).

We show that there is a better way to use multiple agents. By letting the work itself shape the team, we turn the common finding that more agents make things worse into a setting where more agents make things better. Across software builds, knowledge work, and standard benchmarks, ours is the only method we test that improves as the team grows.

2 Do More Agents Help?

The case for a team of agents borrows an intuition from human organizations: work too large for one worker is divided among several. Open-ended creation looks like the perfect fit, outgrowing any single effort, and several goals matter at once. Before building on that intuition, we test it.

Setup. We collect a list of build tasks: small games and software artifacts, each specified by a one- or two-sentence request written the way a real user would write it (Figure 3). The request names user need and stops: no required feature, no binary acceptance criteria, no choice of technology or stack. Since nothing defines completion, a run ends when its budget runs out. We launched a separate agent that grades each delivered artifact: it receives the request and the artifact, nothing else, and returns a score from 0 to 100.

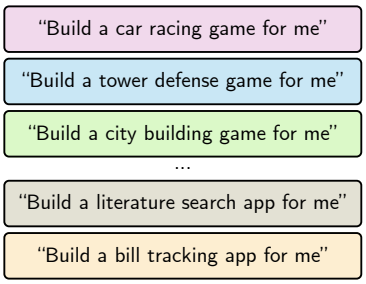


Figure 3: **Task examples.** Each names the artifact and stops; no feature list or no criteria.

More agents, worse result. We give the same prompt to teams of two to thirty agents, either *centralized*, one orchestrator dispatching a flat pool of workers, as most agent harnesses do, or *decentralized*, every agent given the prompt and the workspace with no orchestrator and free to message any other (Figure 6). Under both, the score falls as more agents are added (Figure 4). Table 1 breaks it down: from two agents to thirty, the score drops from 56 to 40 and the share of runs that ship nothing working rises from one in ten to two in five, while the team lands four times the commits and spends fourteen times the tokens. They work more and hand back a worse artifact. Figure 5 shows one pair.

Why teams fail. We use an agent to help judge the failure mode and found three repeated patterns (Figure 7), and none of them is a reasoning failure. *No one in charge*: every agent completes its own job, and nobody checks that what they do forms the artifact the user wants. *Duplication*: with no record of who is responsible for what, an idle agent picks up work another agent is halfway through, and the team pays twice for the same feature. *Collision*: two agents edit the same file against different assumptions, and whichever merge lands second quietly undoes the first (Conway, 1968; Cataldo et al., 2006; Xia et al., 2026). All three contribute to the final failure.

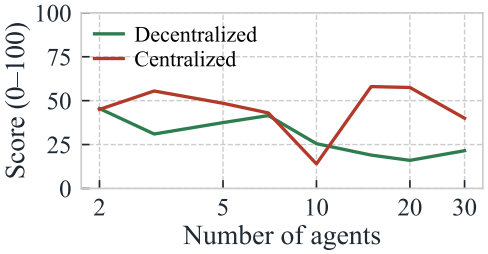


Figure 4: **Quality falls as teams grow.** Judge score against team size (log scale); both organizations trend downward.

Evaluation metric	2 agents	30 agents
Judge score ↑	**56**	**40**
Something broken ↓	**40%**	**80%**
No working artifact ↓	**10%**	**40%**
Tokens spent ↓	37M	531M
Commits landed	19	82

Table 1: **More agents, worse results.** Two agents against thirty on the same task. Middle rows give the share of runs; arrows mark the desirable direction.

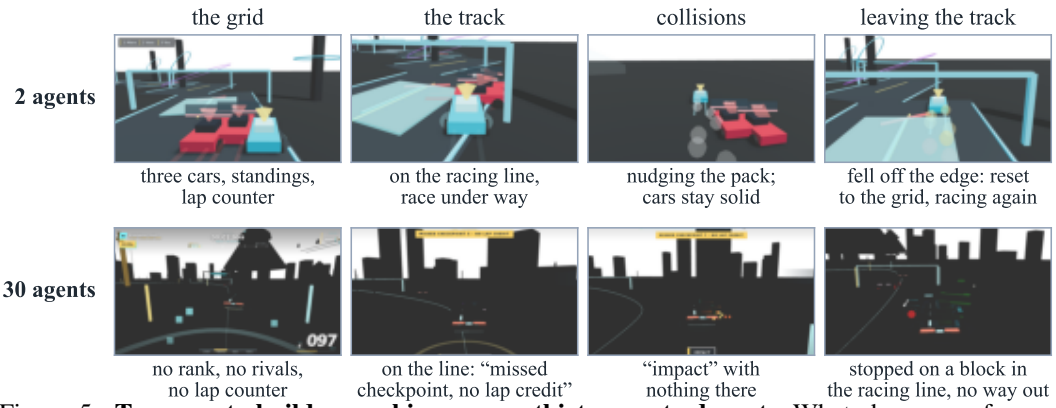


Figure 5: **Two agents build a working game; thirty agents do not.** What shown are frames from playing both builds (the prompt is "Build a car racing game for me"); each column tests one property. The two-agent can build a naive but functioning game. The thirty-agent team implement twenty-eight features in a few minutes. Each agent built in its own module; all landed in to the final repo (440 commits, six car racing tracks, eight different rivals cars), but they are not a single artifact, nothing is connected together: the rivals wait on a grid check that never passes, and the commits wiring the race modes together never reached main.

None of the three is a failure of ability. Each of the thirty agents can write working code by itself. What goes wrong is how they work together, a loss long measured on human teams (Ringelmann, 1913; Kravitz & Martin, 1986; Steiner, 1972; Latané et al., 1979; Staats et al., 2012; Brooks, 1995): no one is responsible for any single part, no one knows what the others are doing, and no one puts the pieces together. The natural fix is to organize the team the way a human team is organized (Simon, 1962; Galbraith, 1974; Mintzberg, 1979; Malone & Crowston, 1994), as classic multi-agent systems did (Fox, 1981; Smith, 1980; Durfee et al., 1987; Ishida et al., 1992; Tambe, 1997; Kamboj & Decker, 2007; Sims et al., 2008; Horling & Lesser, 2004): someone is responsible for each part, and someone decides who does what. But that plan cannot be written down at the start. Thirty agents cannot divide the responsibilities in the first minute, before anything exists; what a team needs only becomes clear while something is being built. So the question is:

"How should a team of agents organize itself,
so that adding an agent makes it better rather than worse?"

3 Self-Evolving Agent Team

We build a team that grows while it works. A run starts with a single agent, the director, holding the prompt. It ends with whatever team the work needed. Three backbones hold the team together (Figure 8): a *structure* that says who does what and who reports to whom, a *communication* layer for talking, and a shared *workspace* where all the work lands in one artifact. They answer the three failures of Figure 7 in order. A fourth part, the *Architect*, decides what the team looks like. It watches from outside and changes the team between rounds by adding more agents. A round is one delivery: the team works, the director ships what exists, and the Architect looks at it.

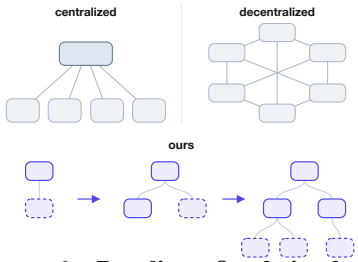
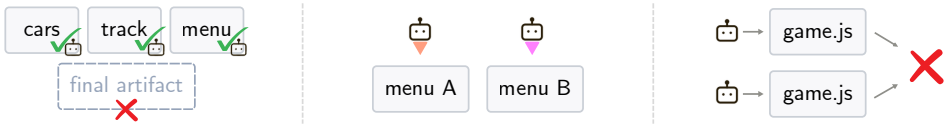


Figure 6: **Baselines fix their shape; ours grows one.** The centralized and decentralized organizations are set before the run; ours builds a tree during it.

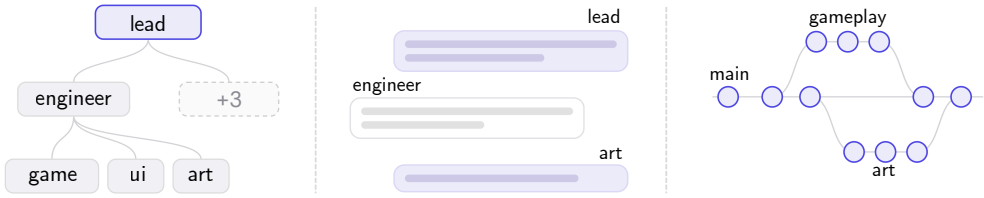
3.1 Three backbones

Structure. The team is a tree of positions. A position is a job, not an agent. It has a role, a short charter saying what it is responsible for, a parent it reports to, and powers, such as whether it may hire or dismiss the positions below it. An agent fills the position; if the agent is replaced, the position stays. The root is the director. It sets direction for the entire team, and is the only position that can decide when to stop, i.e. deliver the artifact. Each position keeps three notes across rounds: how the role works here, what to do on this task, and what went wrong last round.



(a) no one in charge (b) duplication (c) collision

Figure 7: **Unorganized teams fail in three ways.** (a) No one in charge: every component is finished and nobody puts them together into an artifact. (b) Duplication: an idle agent redoes work a teammate is halfway through. (c) Collision: two agents edit the same file against different assumptions.



(a) structure (b) communication (c) workspace

Figure 8: **Each backbone answers one failure mode.** (a) Structure: a tree of positions, each holding a charter and reporting to a parent. (b) Communication: direct messages and channels that skip the tree. (c) Workspace: one repository whose branches merge into a single mainline.

Communication. Any position can message any other, one to one or in a group channel. Nothing has to go through the tree. Every lead and its reports share a team channel, and that is where work is discussed; direct messages are for one-to-one matters, because work hidden in private messages is invisible to the team. When a position finishes or gets stuck, it posts one line in its team channel.

Workspace. All code lives in one git repository. Every position but the director has its own clone and its own branch. The director has no clone on purpose: it cannot write the artifact and has to get it built. Work enters the artifact only by merging into main, so two positions editing the same file get a conflict instead of a silent overwrite. Anything left on a branch is not in the final artifact, and the team has tool to know about it. The repository also holds the team's documents: what we are trying to build, live notes for each area, and what the testers actually reached in the shipped.

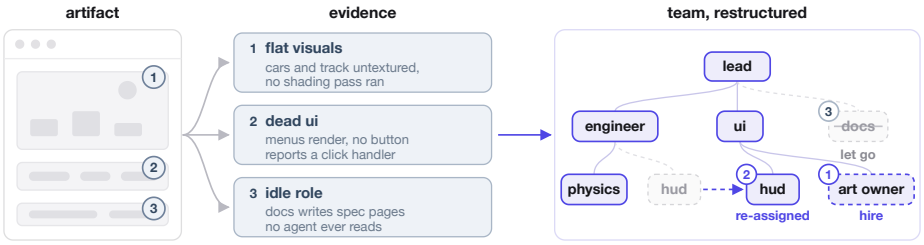
3.2 The Architect

The backbones let a team work. The Architect decides what team to have. It sits outside the tree and writes no code. It runs between rounds (Figure 9b). At each time it is woken up, it is told by simple prompting to review the team's work and their communications. It also reads the org chart, what each position did and cost, and the recent commits.

From this, it tries to find out the worst issue the team currently has; for example, one sub-team is too busy: they have too much work planned but not enough workers in it. From this, it decides which gap in the team let it happen, and fixes the gap: it can hire a position, let one go, re-assign one to another parent, or re-shape one by rewriting its charter, and it rewrites the notes each position keeps. It does not decide what the artifact should be; that stays with the director, who in turn reports stalled positions to the Architect instead of moving people. Figure 9a shows one pass: untextured cars mean nobody is responsible for how the game looks, so an art position is hired; menus that ignore clicks mean nobody is in charge of the interface, so the HUD position moves under the interface position; a docs position nobody reads is let go. The runs look the same. In one, the director began by hiring three directors before any code existed, and the Architect cut the team to seven positions; four rounds later the one engineer on the core was also managing every feature, so the Architect hired a lead for a new team and moved the feature positions under it.

3.3 Sub-teams

Teams are not declared; they form (Figure 10). When a position has two or more reports, it becomes a lead, its reports become its team, and a team channel opens. The lead names the team; in one run they were Studio Direction, Racing Craft, and Racing Experiences. A lead assigns, coordinates, and reviews; it no longer builds. The Architect can also form a team on purpose, by hiring a lead and moving positions under it, when several positions work in parallel with no one keeping aligned.



(a) one pass: what the Architect sees in the artifact, and what it changes in the team



(b) the outer loop: each change becomes part of what it reads next round

Figure 9: **The team evolves from the artifact.** (a) Three defects of the delivered game become three gaps in the team: HIRE adds an art position, RE-ASSIGN moves the HUD position under the interface position, and LET GO retires the idle docs position. (b) The Architect reads the delivered artifact, changes the team, and reads the result on the next pass.

Each team keeps its own documents in the repository. The Architect writes the team's purpose: what it exists to deliver and where its edge with other teams lies. The lead writes the plan: the pieces, who has each, and what done looks like. Each member keeps a checklist. When the Architect reorganizes, the documents move alongside.

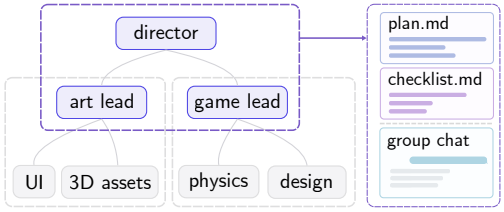


Figure 10: **Teams form around leads.** A position with two or more reports becomes a lead; its reports are its team, with their own channel and documents.

Inside a team, the lead assign work to its reports with the goal, and what is considered as done. Members work on their branches and post a message when they finish (or get stuck and need help); the lead reviews and merges each pull requested. Positions in different teams that share an chat channel and discuss their visions, plans, and problems. The What the team learned stays in its documents, so a new lead does not start over.

4 RESULTS

4.1 EXPERIMENTAL SETUP

Protocol. We give the same prompt to several multi-agent systems and compare what they deliver. All agents run Codex GPT-5.6 Sol at medium effort. Every system gets the same tools, token budget, and runtime; the only difference is how the agents are organized. Each system works in rounds: we count each round as when the team decide to stop. At the end of every round the team hands in what it has built so far. We loop it ten times for all. After the final round, the automated judge from Section 2 uses the delivered artifact blind and scores it out of 100.

Tasks. We evaluate on three kinds of work. First, eight browser games (Table 2), each specified by a one-line brief like "build a car racing game." Second, three existing benchmarks: GDPval (Patwardhan et al., 2026), which asks for knowledge-work deliverables across ten occupations, and ProgramBench (Yang et al., 2026) and NL2RepoBench (Ding et al., 2026) for coding (ten problems each, drawn with the official random selector). Third, three requests collected from real users who were not involved in the project: a deep literature survey rendered as an interactive HTML demo, a

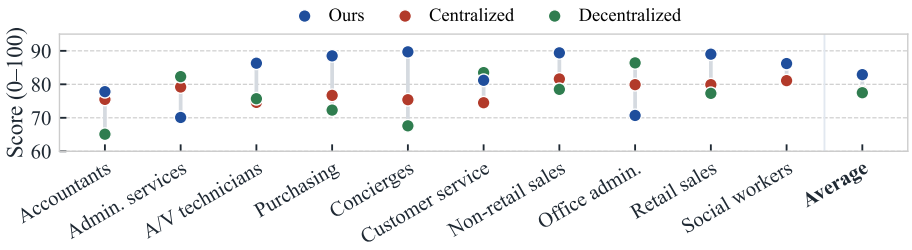


Figure 11: **GDPval scores by occupation.** Ours leads in six of ten.

Task	Ours	Cent.	Dec.	PE2	AV	MacN.	Meta.
Car Racing	**74**	40	22	59	49	43	26
City Builder 3D	**72**	59	37	68	64	69	65
Detective	**77**	32	54	52	54	64	41
Fishing	**75**	67	53	58	64	55	51
Rhythm	**68**	47	**68**	57	66	57	51
Shooter	57	**58**	36	51	44	41	42
Stealth	**79**	71	51	75	48	53	41
Tower Defense	59	**70**	56	61	56	57	46
Mean	**70.1**	55.5	47.1	60.1	55.6	54.9	45.4

Table 2: **Judge scores on eight games** (0–100) at thirty agents. Cent. = Centralized, Dec. = Decentralized, AV = AgentVerse, MacN. = MacNet, Meta. = MetaGPT.

Benchmark	Ours	Codex
GDPval	**82.9**	77.8
ProgramBench	**73**	26
NL2RepoBench	**53.3**	52.4

Table 3: **Benchmarks** (0–100).

User request	Ours	Codex
Lit. survey	**9**	7
Animation	**9**	3
Wardrobe app	**8**	6.5

Table 4: **User requests** (1–10).

duck-riding-a-bike animation with adjustable parameters, and a wardrobe app that records clothing and suggests outfits. Each requester graded the delivered artifacts blind.

Baselines. On the games we compare against two fixed team layouts (Figure 6), each at the same team size as ours: a *centralized* layout where one orchestrator assigns work to a flat pool of agents, and a *decentralized* layout where every agent gets the prompt and the workspace with nobody coordinating. To check whether the centralized team simply had a bad prompt, we also ran it with automatic prompt rewriting (Ye et al., 2024; Yang et al., 2024) (PE2), and we added three published systems: AgentVerse (Chen et al., 2024b), MacNet, and MetaGPT (Hong et al., 2024). On the benchmarks and real-user requests, the baseline is native Codex with a prompt that instructs it to launch as many agents as helpful and and we inject a strong prompt plus a `/goal` command.

4.2 Main Results

We win six of eight games and average 70.1 (Table 2). Prompt rewriting barely moves the centralized team (60.1 vs. 55.5), so a better prompt is not what it needs. On GDPval we lead in six of ten occupations (Figure 11), averaging 82.9 against 77.8 for Codex (Table 3). On ProgramBench we score 73 to Codex's 26; on NL2RepoBench the gap nearly vanishes (53.3 vs. 52.4), which tracks with ProgramBench requiring many files to work together while NL2RepoBench problems mostly do not. Real users scored blind (Table 4): the largest gap is the animation (9 vs. 3, where Codex produced a static page and ours delivered a working demo), with the literature survey and wardrobe app closer (9 vs. 7 and 8 vs. 6.5).

4.3 Scaling

Adding agents hurts every baseline but helps ours (Figure 12, top): at thirty agents we score 74, 34 points above the next best, while the centralized and decentralized teams both finish below where they started. Round by round (Figure 12, bottom), every other method plateaus after two or three rounds while ours reaches 74 by round ten. It takes longer (three hours versus under two) because the team grows, and the artifact shows it: Figure 12(c) counts the files reachable in the submitted artifact after each round, and under the same ten rounds and cap of thirty agents the baselines end with 53 and 30 files and ours with 291. Their counts flatten early, as their agents converge on repeating what

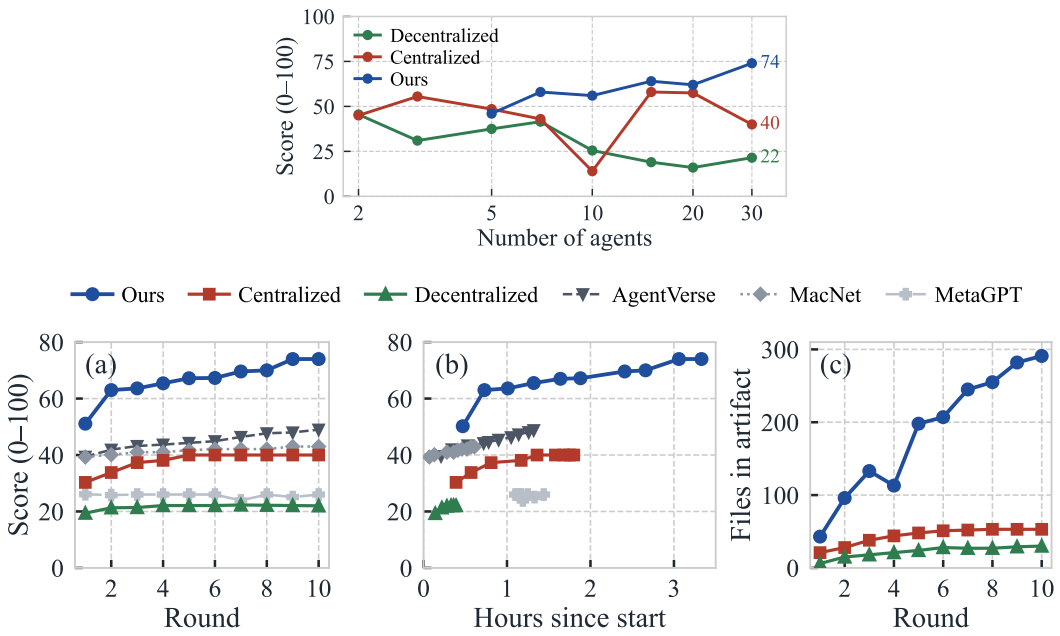


Figure 12: **Scaling behavior.** Top: score of the play judge, which plays each build, against team size (log scale); only ours improves with more agents. Bottom: one run of each method on the same prompt showing (a) score after each round, (b) the same against wall-clock time, (c) files in the delivered artifact.

Arm	Score	Delivered game
Full method	**74**	the run scored in Table 2
No diagnosis	46	menu only
On request	40	drives, lap 2 of 3
Frozen	30	car does not move
No charters	26	core never starts
Hired at once	21	renders, car does not move
No standard positions	14	route map, no race
Flat	8	title screen only

Table 5: **Every removal costs at least 28 points; removing depth costs nearly everything.** One run per arm on the car racing brief, scored by the play judge of Table 2 (0–100). Last column: what the play probe found in the build.

exists, the collisions and repeated work of Section 2; in ours the count climbs with the team, each new position bringing a part of the game the artifact did not yet have.

5 Taking Our Method Apart

We remove one commitment at a time on the prompt "build a car race game," one run per arm, and the judge scores the final build (Table 5). *No diagnosis* lets the Architect change the team, but never because of anything it saw in the artifact. *On request* makes it fill the director's staffing requests and decide nothing itself; *Hired at once* makes it design and hire the whole team up front instead of growing it; *Frozen* allows no change after the first hires. *No charters* leaves no position responsible for any part, so whoever is free takes the next piece of work; *No standard positions* gives the Architect no standard set of positions to fill and lets it invent every role. *Flat* allows no leads or teams. The full method scores 74 on this prompt (Table 2). The least damaging removal, changing the team blindly, still loses 28 points, and the order below it is the finding: the team survives changing blindly (46) better than not changing (30), having no responsibilities (26), being hired all at once (21), or having no standard positions (14), and it does not survive being flat (8).

The exception is *On request*: with six positions it delivers the only game that drives, a car in its second lap, and the judge puts it second at 40. We read it as a warning about small briefs, where each added position costs more coordination than it returns, not as evidence that the Architect's judgment is optional, and it is one run.

6 Related Work

Multi-Agent Systems. A multi-agent system runs several language model instances that exchange messages to solve one task. The earliest line uses the extra agents as extra samples of one problem, by voting over reasoning chains (Wang et al., 2023; Li et al., 2024), by debate (Du et al., 2024; Liang et al., 2024), or by aggregating and verifying many responses (Wang et al., 2025a; Snell et al., 2025). A second line gives agents distinct roles modeled on a human organization: CAMEL pairs a user with an assistant agent (Li et al., 2023), ChatDev and MetaGPT run a virtual software company through fixed phases and roles (Qian et al., 2024; Hong et al., 2024; Dong et al., 2024), and AutoGen lets developers script the conversation (Wu et al., 2024). A third line lets the team change while it works: DyLAN deactivates agents that contribute little (Liu et al., 2024), AgentVerse re-recruits experts from an evaluator's feedback (Chen et al., 2024b; Song et al., 2024; Guo et al., 2026), GPTSwarm and AgentDropout optimize or prune the communication graph (Zhuge et al., 2024; Wang et al., 2025c; Yang et al., 2025; Niu et al., 2025), and Self-Organized Agents and MegaAgent grow the team by decomposing the task (Ishibashi & Nishimura, 2024; Wang et al., 2025b). Across these systems, adding agents brings diminishing or negative returns (Chen et al., 2024a; Kim et al., 2025; Pappu et al., 2026; Bertalanič & Fortuna, 2026), mostly through specification and coordination failures rather than reasoning (Cemri et al., 2025). Ours belongs to the third line and differs in two ways: the change is driven by the goal, and the team grows into a hierarchy rather than a flat set.

Self-Evolving Agents. A self-evolving agent improves itself after deployment, and a recent survey sorts the field by what evolves: prompts, memory, tools, code or harness, model weights, or the whole workflow (Gao et al., 2026). Reflexion carries verbal reflections across attempts (Shinn et al., 2023), generative agents accumulate memory (Park et al., 2023), and Voyager grows a skill library (Wang et al., 2024). OPRO, PE2 and TextGrad optimize prompts from the model's own feedback (Yang et al., 2024; Ye et al., 2024; Yuksekgonul et al., 2025). ADAS and AFlow search over agent designs and workflows (Hu et al., 2025a; Zhang et al., 2025), and the Darwin Gödel Machine rewrites its own code and keeps versions that score higher (Zhang et al., 2026a). Multi-agent versions evolve the collaboration network from unit tests (Hu et al., 2025b) or a pool of agents toward open-ended discovery (Qu et al., 2026; Hughes et al., 2024). In each case what evolves belongs to the agent, and a score or test decides what stays. We evolve a different thing, the organization itself: which positions exist, who reports to whom and where the work sits, changed from the what built.

7 Conclusion

Thirty agents in a fixed organization delivered less than two. Nobody was in charge of the whole artifact, work got done twice, and one agent's edits broke another's. We built a team that reorganizes itself as it works: an Architect looks at what was delivered, not at a score, and adds, moves, or removes positions between rounds. In every experiment we ran, it is the only team that actually gets better when you give it more agents.

AI USE STATEMENT

The system this paper studies is made of AI agents, so we separate what the agents did from what we did. Every artifact scored in the paper was built by the agent teams under study; that is the object of study, not writing help. The judges that score the games and the GDPval deliverables are AI agents as well, from a different model family than the builders (Appendix D); ProgramBench is scored by the benchmark's test suite and the user requests by the people who asked. In preparing the paper we used AI coding assistants (Claude Code and Codex) to write the analysis and plotting scripts, to build the figures including the teaser, to check that every cited reference exists, and to edit prose we drafted. The research questions, the method, the experiment design and the interpretation of the results are ours, and the authors are responsible for every number in the paper.

REFERENCES

Blaž Bertalanič and Carolina Fortuna. The Ringelmann effect in multi-agent LLM systems: A scaling law for effective team size. *arXiv preprint arXiv:2606.02646*, 2026.

Frederick P. Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*. Addison-Wesley, 1995.

Marcelo Cataldo, Patrick A. Wagstrom, James D. Herbsleb, and Kathleen M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *CSCW*, 2006. doi: 10.1145/1180875.1180929.

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? In *NeurIPS*, 2025.

Lingjiao Chen, Jared Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. Are more LLM calls all you need? towards the scaling properties of compound AI systems. In *NeurIPS*, 2024a.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *ICLR*, 2024b.

Melvin E. Conway. How do committees invent? *Datamation*, 1968.

Jingzhe Ding, Shengda Long, Changxin Pu, Huan Zhou, Hongwan Gao, Xiang Gao, Chao He, Yue Hou, Fei Hu, Zhaojian Li, Weiran Shi, Zaiyuan Wang, Daoguang Zan, Chenchen Zhang, Xiaoxu Zhang, Qizhi Chen, Xianfu Cheng, Bo Deng, Qingshui Gu, Kai Hua, Juntao Lin, Pai Liu, Mingchen Li, Xuanguang Pan, Zifan Peng, Yujia Qin, Yong Shan, Zhewen Tan, Weihao Xie, Zihan Wang, Yishuo Yuan, Jiayu Zhang, Enduo Zhao, Yunfei Zhao, He Zhu, Liya Zhu, Chenyang Zou, Ming Ding, Jianpeng Jiao, Jiaheng Liu, Minghao Liu, Qian Liu, Chongyang Tao, Jian Yang, Tong Yang, Zhaoxiang Zhang, Xinjie Chen, Wenhao Huang, and Ge Zhang. NL2Repo-Bench: Towards long-horizon repository generation evaluation of coding agents. In *ICML*, 2026.

Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. Self-collaboration code generation via ChatGPT. *ACM Transactions on Software Engineering and Methodology*, 2024.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *ICML*, 2024.

Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, 1987.

Mark S. Fox. An organizational view of distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 1981.

Jay R. Galbraith. Organization design: An information processing view. *Interfaces*, 1974.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *TMLR*, 2026.

Xudong Guo, Kaixuan Huang, Jiale Liu, Wenhui Fan, Natalia Vélez, Qingyun Wu, Huazheng Wang, Thomas L. Griffiths, and Mengdi Wang. Embodied LLM agents learn to cooperate in organized teams. *IEEE Transactions on Computational Social Systems*, 2026.

Sirui Hong, Mingchen Zhuge, Jonathan 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 *ICLR*, 2024.

Bryan Horling and Victor Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 2004.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *ICLR*, 2025a.

Yue Hu, Yuzhu Cai, Yaxin Du, Xinyu Zhu, Xiangrui Liu, Zijie Yu, Yuchen Hou, Shuo Tang, and Siheng Chen. Self-evolving multi-agent collaboration networks for software development. In *ICLR*, 2025b.

Edward Hughes, Michael D. Dennis, Jack Parker-Holder, Feryal Behbahani, Aditi Mavalankar, Yuge Shi, Tom Schaul, and Tim Rocktäschel. Position: Open-endedness is essential for artificial superhuman intelligence. In *ICML*, 2024.

Yoichi Ishibashi and Yoshimasa Nishimura. Self-organized agents: A LLM multi-agent framework toward ultra large-scale code generation and optimization. *arXiv preprint arXiv:2404.02183*, 2024.

Toru Ishida, Les Gasser, and Makoto Yokoo. Organization self-design of distributed production systems. *IEEE Transactions on Knowledge and Data Engineering*, 1992. doi: 10.1109/69.134249.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *ICLR*, 2024. doi: 10.48550/arXiv.2310.06770.

Sachin Kamboj and Keith S. Decker. Organizational self-design in semi-dynamic environments. In *AAMAS*, 2007. doi: 10.1145/1329125.1329370.

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.

David A. Kravitz and Barbara Martin. Ringelmann rediscovered: The original article. *Journal of Personality and Social Psychology*, 1986.

Bibb Latané, Kipling Williams, and Stephen Harkins. Many hands make light the work: The causes and consequences of social loafing. *Journal of Personality and Social Psychology*, 1979.

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. *TMLR*, 2024.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *EMNLP*, 2024.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *COLM*, 2024.

Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 1994.

Henry Mintzberg. *The Structuring of Organizations: A Synthesis of the Research*. Prentice-Hall, 1979.

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: Modularized agentic workflow automation. In *ICLR*, 2025. doi: 10.48550/arXiv.2501.07834.

Aneesh Pappu, Batu El, Hancheng Cao, Carmelo di Nolfo, Yanchao Sun, Meng Cao, and James Zou. Multi-agent teams hold experts back. In *ICML*, 2026.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *UIST*, 2023. doi: 10.1145/3586183.3606763.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simón Posada Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, Natalie S. Kim, Samuel Miserendino, Gildas Chabot, David Li, Patrick Chao, Michael Sharman, Alexandra Barr, Amelia Glaese, and Jerry Tworek. GDPval: Evaluating AI model performance on real-world economically valuable tasks. In *ICLR*, 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 *ACL*, 2024.

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 *ICLR*, 2025.

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 *COLM*, 2026.

Max Ringelmann. Recherches sur les moteurs animés: Travail de l'homme. *Annales de l'Institut National Agronomique*, 1913.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

Herbert A. Simon. The architecture of complexity. *Proceedings of the American Philosophical Society*, 1962.

Mark Sims, Daniel Corkill, and Victor Lesser. Automated organization design for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 2008.

Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 1980.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *ICLR*, 2025.

Linxin Song, Jiale Liu, Jieyu Zhang, Shaokun Zhang, Ao Luo, Shijian Wang, Qingyun Wu, and Chi Wang. Adaptive in-conversation team building for language model agents. *arXiv preprint arXiv:2405.19425*, 2024.

Bradley R. Staats, Katherine L. Milkman, and Craig R. Fox. The team scaling fallacy: Underestimating the declining efficiency of larger teams. *Organizational Behavior and Human Decision Processes*, 2012.

Ivan D. Steiner. *Group Process and Productivity*. Academic Press, 1972.

Milind Tambe. Towards flexible teamwork. *JAIR*, 1997.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *TMLR*, 2024.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In *ICLR*, 2025a.

Qian Wang, Tianyu Wang, Zhenheng Tang, Qinbin Li, Nuo Chen, Jingsheng Liang, and Bingsheng He. MegaAgent: A large-scale autonomous LLM-based multi-agent system without predefined SOPs. In *Findings of ACL*, 2025b. doi: 10.18653/v1/2025.findings-acl.259.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *ICLR*, 2023.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. AgentDropout: Dynamic agent elimination for token-efficient and high-performance LLM-based multi-agent collaboration. In *ACL*, 2025c. doi: 10.18653/v1/2025.acl-long.1170.

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 conversations. In *COLM*, 2024.

Haocheng Xia, Eugene Wu, and Yongjoo Park. Passes alone, fails together: Benchmarking semantic coordination in parallel LLM-agent development. In *EXPRESS Workshop*, 2026.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Melroy Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. In *NeurIPS*, 2025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *ICLR*, 2024.

John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. Program-Bench: Can language models rebuild programs from scratch? *arXiv preprint arXiv:2605.03546*, 2026.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. AgentNet: Decentralized evolutionary coordination for LLM-based multi-agent systems. In *NeurIPS*, 2025.

Qinyuan Ye, Maxamed Axmed, Reid Pryzant, and Fereshte Khani. Prompt engineering a prompt engineer. In *Findings of ACL*, 2024.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents. In *ICLR*, 2026a.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *ICLR*, 2025.

Wei Zhang, Jian Yang, Renshuai Tao, Linzheng Chai, Shuyue Guo, Jiajun Wu, Xiaoming Chen, Ganqu Cui, Ning Ding, Xander Xu, Hu Wei, and Bowen Zhou. V-GameGym: Visual game generation for code large language models. In *Findings of ACL*, 2026b. doi: 10.18653/v1/2026.findings-acl.276.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *NeurIPS*, 2023. doi: 10.48550/arXiv.2306.05685.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *ICML*, 2024.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-Judge: Evaluate agents with agents. In *ICML*, 2025.

APPENDIX

A RUNTIME AND ROUNDS

Every agent is one Codex CLI session running GPT-5.6 Sol at medium effort, the Architect included, and no run has network access. Messaging, delegation, notes, hiring, advice and reports are all runtime tools (Section 3).

Every game run is set to ten rounds, except the ablation runs made under an earlier run script. A round closes once the team has delivered and then goes idle. Three clocks bound it: a delivered round closes after fifteen minutes if the team never idles, which is how most rounds end at thirty positions, a round with no delivery closes after sixty minutes, and a team idle for about fifteen minutes without a new delivery has its round closed anyway. The artifact scored is the last delivery of round ten. Rounds, not tokens or time, are the budget.

Our method adds a round zero in which the director and the Architect stand up the starting team, and messages to positions not yet started wait until it ends. The Architect makes its first pass ten minutes after the start, then runs pass after pass with a thirty-second pause, also wakes when the team goes idle, and files at least one report per round. Each position keeps three notes, on the role, the task and last round's mistakes, that a new occupant reads first.

B TASKS

B.1 GAME BRIEFS

The eight briefs (car racing, city builder 3D, detective, fishing, rhythm, shooter, stealth and tower defense) are page-long requests of roughly 900 to 1,150 words, the same text for every arm. Each asks for a large, multi-file browser game at shipping quality on a vendored production web engine and names the one moment of play that must work first. They describe the experience wanted, with no feature checklist or acceptance tests. Table 6 gives one sentence per brief, and Figure 13 shows screens from delivered artifacts.

B.2 BENCHMARKS

GDPval (Patwardhan et al., 2026) asks for knowledge-work deliverables. We take ten occupations, one task each, from the public release with the task's reference files: computer and information systems managers, concierges, editors, financial and investment analysts, first-line supervisors of office and administrative support workers, industrial engineers, mechanical engineers, nurse practitioners, wholesale and manufacturing sales representatives, and software developers. The task text is the benchmark's own, about 570 words on average, and the deliverables are documents, spreadsheets and presentations (while our system is codebase-based, such form of presentations can also be version controled as code workspaces). ProgramBench (Yang et al., 2026) and NL2RepoBench (Ding et al., 2026) have far more problems than we could run, so we use ten problems from each, chosen by the benchmark's own random selection script, and score them with the benchmarks' official evaluations. A ProgramBench problem rebuilds a real program from its specification and is scored by the share of hidden tests passed. An NL2RepoBench problem builds a library or program from its documentation in an empty workspace. Unlike patch benchmarks (Jimenez et al., 2024) or simulated-company tasks (Xu et al., 2025), both ask for a whole program.

B.3 USER REQUESTS

Three requests came from people outside the project (Table 7): a literature survey as an interactive HTML demo, a duck-riding-a-bike animation, and a wardrobe app that suggests outfits. Each requester graded both deliverables blind from one to ten (Table 4). Three raters show that the difference is visible to the person who asked, not how large it is.

Brief	What it asks for
Car racing	A car racing game the player is driving one click after opening it: responsive keyboard driving with a real sense of speed, a track worth being on, opponents that race on the same physics, and engine, skid and impact sound.
City builder 3D	A real-time 3D city-building simulation with terrain, buildings that rise as zones develop, traffic and citizens in the streets, a free camera, and many independent systems that interact.
Detective	A detective game in which the player examines a scene at once: evidence worth a close look, testimony that contradicts other testimony, a way to connect findings into a claim, and an accusation the game judges with reasons.
Fishing	A fishing game in which the player is casting at once: aiming and casting with real feel, a readable bite, a hook-and-reel fight the player can lose, and water and sound that sell it.
Rhythm	A rhythm game in which the player hits notes in time with audible music: tight timing judgment, immediate feedback on every hit and miss, a combo that builds, and fair difficulty.
Shooter	A 3D arena or wave-survival shooter: smooth movement and mouse look, a weapon that fires where the crosshair points, and enemies that see, approach, attack and visibly react to hits.
Stealth	A stealth game in which the player is sneaking past a guard at once: a real difference between fast and quiet movement, guards with vision cones, hearing, patrols and alert states, and clear feedback on exposure.
Tower defense	A tower-defense game in which the player places and upgrades towers against waves advancing along a path toward something to protect, with every kill paying out and waves that escalate.

Table 6: **The eight game briefs.** One sentence on the core each brief asks for first; every brief then asks for the same things around it (Appendix B).

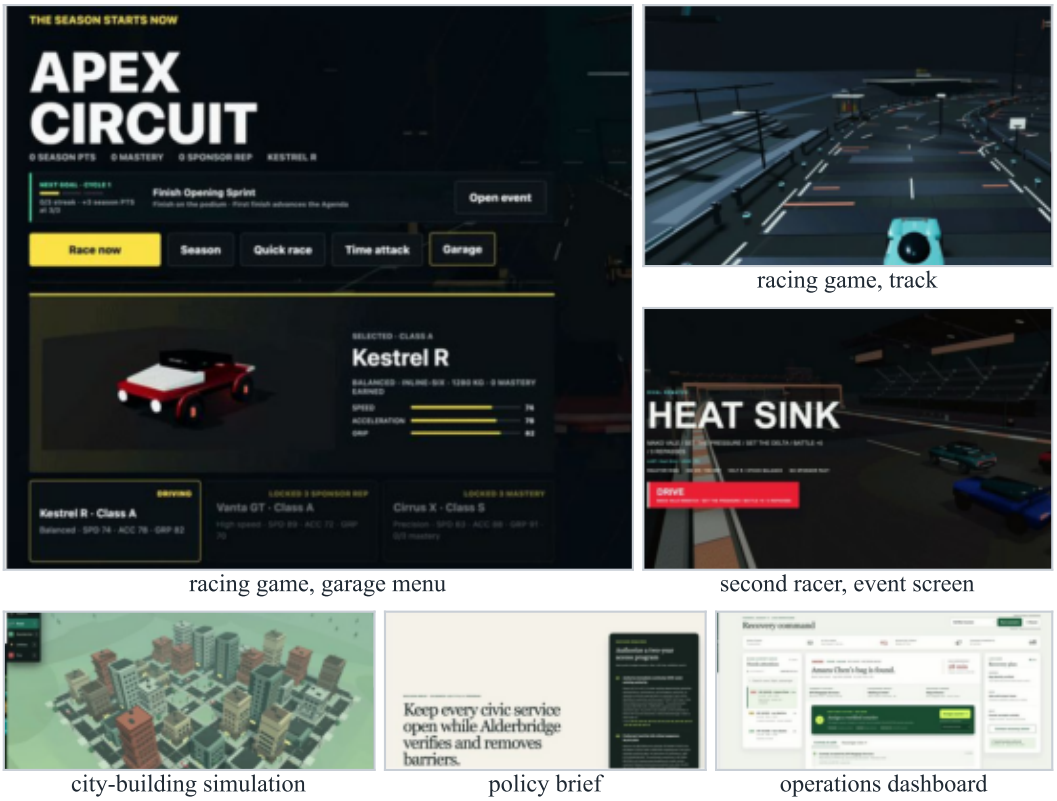
Request	What was asked
Literature survey	A deep survey of a research literature, delivered as an interactive HTML demo rather than a document.
Animation	An animation of a duck riding a bicycle, with parameters the viewer can adjust.
Wardrobe app	An app that records the clothes a person owns and suggests outfits from them.

Table 7: **The three user requests.** Each came from a person outside the project, who graded both deliverables blind (Table 4).

C BASELINES

The game baselines run inside our runtime with the same model, tools, briefs, ten rounds and cap of thirty, differing only in the organization prompt. Table 2 rests on one run per system per game at thirty positions, with repeats for ours on car racing. The centralized layout is one lead over twenty-nine workers with no Architect, and since workers do not message each other all coordination goes through the lead. The decentralized layout is thirty equal peers with no leader, free to message anyone, and no one assigns work. The sweep of Section 2 runs both on car racing at 2, 3, 5, 7, 10, 15, 20 and 30 positions, 40 and 46 runs including repeats. PE2 is the centralized layout with its prompt rewritten in the style of PE2 and OPRO (Ye et al., 2024; Yang et al., 2024): a model diagnoses the prompt's contradictions and ambiguities and rewrites it, and the result, about 32,000 characters, replaces the original with nothing else changed.

AgentVerse (Chen et al., 2024b), MacNet (Qian et al., 2025) and MetaGPT (Hong et al., 2024) are re-implemented inside the same runtime, copy code from their repos, but not directly run from their code bases, otherwise we cannot ensure it is fair.



racing game, garage menu | racing game, track | second racer, event screen | city-building simulation | policy brief | operations dashboard

Figure 13: **Screens from artifacts delivered by our method.** A racing game's garage menu and track, a second racing game's event screen, a city-building simulation, a policy brief and an operations dashboard.

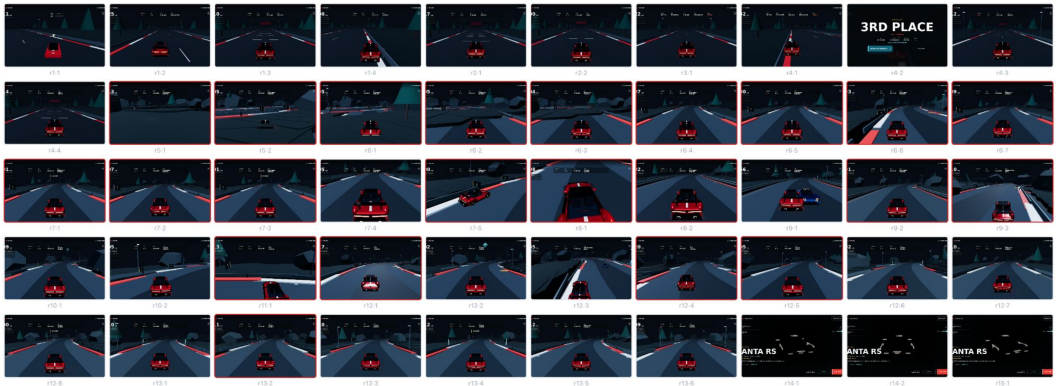


Figure 14: **The judge plays the build.** Frames captured while the judge plays car racing builds in a browser window, one frame per step: it starts the game itself, drives, and reads the result screens before it scores what it played.

D GRADING

The judge plays the game in a browser window (Figure 14), as in Agent-as-a-Judge (Zhuge et al., 2025) and V-GameGym (Zhang et al., 2026b), mindful of judge biases (Zheng et al., 2023). One agent session is given the delivered build served over a local server and a one-line description of the genre, nothing else, and it never reads the code. At each step it sees a screenshot with a coordinate grid drawn over it and answers with one action: click a point, press a key, hold a key for a moment, or wait. The game clock is paused while it thinks and advanced by a fixed slice per action, so it plays turn by turn. It starts the game itself, plays twelve steps, and then grades what it played from

the clean frames. In short, the prompt says: *you are a player trying this game for the first time; each turn you get a screenshot with a grid; reply with one action; start the game yourself; after the last step, rate what you played from 0 to 100 overall, with sub-scores for fun, controls, visuals and brokenness, and let the overall be your own judgment rather than a formula over the sub-scores.* Each build is played three times and the scores are averaged. The judge is the Claude CLI, a different model family from the builders since same-family favoritism has been reported (Patwardhan et al., 2026), and it is blind to the arm. It scores the sweep (Figure 4, Table 1), the car racing row of Table 2 and the ablation (Table 5).

ProgramBench and NL2RepoBench are scored by their official evaluations. GDPval does not release its grader, so we prompt Claude to compare each deliverable with the human expert's deliverable for the same task, blind and with sides balanced, and report the result on a 0 to 100 scale (Table 3, Figure 11).

The deterministic probe is not a judge of quality. A browser harness checks whether the build opens, console errors, reachable pages, survival of an injected infinite loop, code size and commits, nine dimensions summing to 100. It separates broken from working but not working from good, and its ceiling for a game is about 72. The share of runs with something broken or no working artifact in Table 1 and the last column of Table 5 come from it.