

GameBench 2: measuring strategic problem solving through programs that compete

Data: GameBench 2 public release [gb2-20261020-7c5039941207](#), generated October 20, 2026, 12:36 UTC. This is a dated result, not a live ranking.

CONTENTS

Executive summary	6. Limitations and scope of claims
1. Motivation and related work	7. Roadmap
2. Benchmark design	References
3. Scale and integrity	Appendix A. Data sources, licences and credits
4. Results	Appendix B. Counting rated matches
5. How GameBench compares with popular benchmarks	Appendix C. Evidence and reproducibility

Executive summary

GameBench 2 asks AI models to write programs that play eight classic strategy games, then scores each model by how its programs fare against the programs of all the others. The 20 October 2026 release covers 85 models and the 1,855 programs they wrote, which played 428,924 rated matches between them.

From its lowest reasoning setting to its highest, the median model gains 15.3 points, and 52 of the 59 ranked models improve. Even so, the lowest setting gives the most points per dollar for 46 of 57 models (section 4.4).

In the per-game results, the best programs in the eight games come from 6 different models, and several games are near ties.

Placing the models we have evaluated at their release dates gives a cross-section of this one release, with few older models in it (section 4.7). Among models released by October 2025, the best of them scores 42.9 in this release. The best model now scores 89.6, a rise of 46.7 points over the year, and 34.2 of those points came from models released in the last six months.

Across eight comparisons with popular leaderboards, rank correlations run from 0.48 (LiveBench v1 Agentic Coding) to 0.86 (LMArena's Agent Arena), with the Epoch Capabilities Index at 0.84. A stricter test, written down before any result was seen, asked whether adding GameBench to LiveBench's own scores improves predictions of LiveBench's agentic coding results. On that main target neither GameBench nor anything else we tried beat a constant guess, so the test could not tell either way (section 5). In one of six secondary comparisons declared in advance, on LiveBench 2.0's coding tasks, which are built from real pull requests, GameBench reduced the prediction error by 24%, a lead that independent data will have to confirm.

1. Motivation and related work

Most language-model benchmarks are fixed question sets. Once a question set is published it can leak into training data, and as models improve, their scores bunch up near the ceiling.

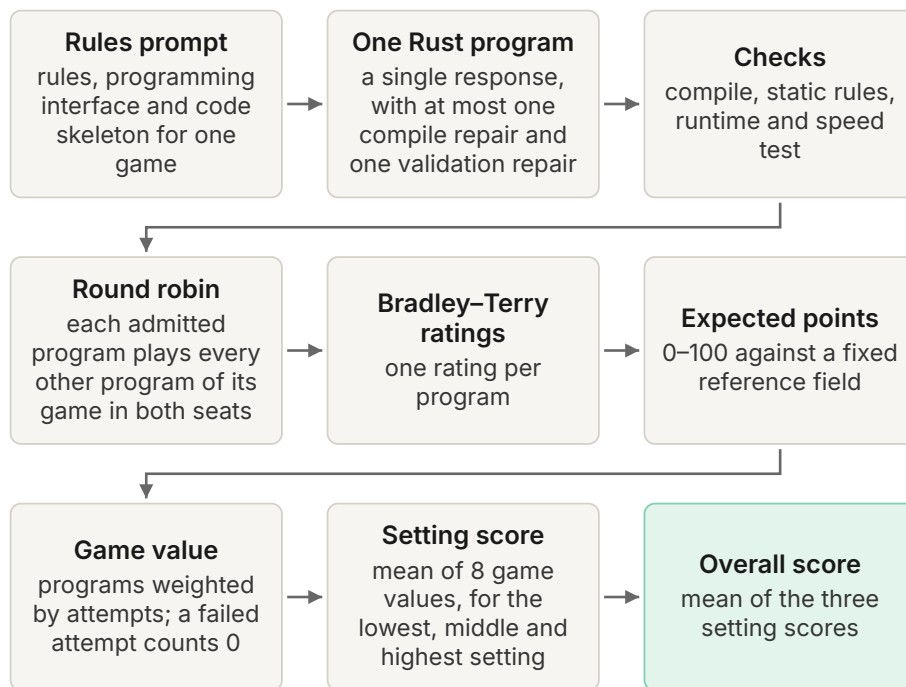
In GameBench 2 a model receives the rules of a game and a small programming interface, and writes a program that plays the game. The programs then meet in deterministic matches, and no human or AI judge decides any result. There is no fixed answer key to leak, although models can draw on published game-playing techniques. Ratings come from head-to-head play and have no fixed ceiling, so a stronger program can keep separating itself from the best existing ones as long as games stay decisive. The games leave plenty of room for that. None of the eight is solved at the board sizes we use: no perfect strategy is known for any of them. When the games are generalized to $n \times n$ boards, deciding who wins from a given position is proven to be hard. It is PSPACE-complete for Hex, Game of the Amazons and Tumbleweed, and EXPTIME-complete for Chess, Shogi and Xiangqi and for $N \times N$ checkers, a close relative of International Draughts. Go is EXPTIME-complete under Japanese ko rules and at least PSPACE-hard under the positional superko rule we use (see the references). These results describe the generalized games rather than our fixed boards, but they mean there is no known shortcut to perfect play.

Unlike the ratings, the published 0–100 score has a top, because it is a program's expected share of points against a fixed reference field (the set of programs that every program is scored against). It compresses near that top, and the highest reasoning settings already come close to it (section 4.2 and section 4.7); we plan to keep headroom with fixed reference players and new games (section 7).

CodeClash (Yang et al., 2025) and ProxyWar (Peng et al., 2026) also evaluate programs that compete. In CodeClash, agents maintain codebases through multi-round tournaments, and ProxyWar evaluates LLM-written game agents with testing and repair. We chose eight classic, deeply studied strategy games as the first set, for their strategic depth and relatively simple rules, and our game engine is built to add more. Every program plays every other in both seats under one recorded and verified execution configuration, and we report our methods and a pre-specified external test alongside the rankings.

2. Benchmark design

The benchmark uses eight games: Go, Chess, Shogi, Xiangqi, Hex, International Draughts, Game of the Amazons and Tumbleweed.



For each of the eight games and each of three reasoning settings, a model writes one program, which is checked, played against every other admitted program of that game and rated. The Overall score is the mean of the three setting scores, and each setting score is the mean of eight game values.

Programs. For each game, a model receives one prompt and answers it in a single response, with no tools, no code execution and no web access. The prompt gives the rules of the game, the programming interface, a minimal code skeleton and short strategy notes, and the model writes one Rust source file that chooses a move in any position. The host supplies the legal moves of each position, with facts such as captures and checks, so a program does not implement the rules. It supplies no search, simulation or evaluation helpers: a program that looks ahead must build that itself. Programs must be deterministic. They may use only the seeded random-number generator the host provides, and no clocks, threads or state kept between moves. The model does this at three of its own reasoning settings: a low one ("baseline"), a middle one ("balanced") and a high one ("intensive"). The intensive setting is XHigh where a model offers it, so a Max setting above XHigh is left out; section 4.3 lists the usual choices. In this report a model's "lowest" and "highest" setting mean its baseline and intensive settings.

If the code does not compile, the model gets one repair with the compiler's errors; if it fails the static checks or a short runtime and speed test, it gets one more with the failure details. It never sees match results. A program that cannot be read, checked, compiled, started or run within these limits is a failed attempt and counts 0. Working programs must also pass an admission speed check: a program whose median move takes more than 100 milliseconds is rejected, plays no matches and counts in no score. Most combinations of model, game and setting have one generation attempt. Where there are more, every admitted program counts, weighted by the attempts behind it.

Matches. Every admitted program is scheduled against every other program of its game in both seats, a double round robin. In this release every pairing has a counted result, as the published per-program match counts show, and in that sense the round robin is complete. Some of those results still come from an earlier engine version until they are replayed. Every counted match runs deterministically under the same execution configuration, and that configuration is recorded and verified for each match; we call this the attested sandbox policy. Besides the median-speed check at admission, every move has an absolute limit of 10 seconds. Each game also has a move cap, a maximum number of moves after which the game stops, counting both players'

moves: for example 400 moves in Chess and 900 in Go. The cap exists because a game could otherwise go on indefinitely, for example when both programs keep repeating the same moves, which many of these games would settle as a draw by repetition. A game that reaches the move cap is a draw, except that under the newest version a Tumbleweed game reaching its cap is left unscored. A crash or illegal move during a program's own move loses the match, while timeouts, other one-sided faults and infrastructure failures are left out of the match evidence.

Scores. Each program gets a Bradley–Terry rating fitted to convergence (prior precision 0.25 per logit, with a first-player term). We convert the rating to expected points on a 0–100 scale against the reference field, averaged over both seats. A model's value in a game weights its programs by the attempts that produced them, with failed attempts counting as zero. We call the results of one reasoning setting a bracket; its score is the mean of the eight game values, and a model's Overall score is the mean of its bracket scores. Only models with scores for all three reasoning settings receive an Overall rank.

Failures are treated according to who caused them. A failed generation attempt counts 0, because writing a working program is part of the task, and because leaving failures out of the mean would let a model gain from failing at the games it plays worst. A program rejected only by the speed check is disqualified but does not lower the model's score, because that check has also rejected working programs for reasons in our harness rather than in the model. In matches, timeouts are left out because they depend on machine load as well as on the program, and infrastructure failures never count against either side.

After a rules or engine correction, programs replay their matches under the new version. Only the newest version's result for each pairing counts, and older results stay in the data. Until every pairing has been replayed, a game mixes versions.

All GameBench numbers in this report come from public release [gb2-20261020-7c5039941207](#), scored under policy `attributed_fault_loss_official_qualification_v17` against reference field `model_balanced_common_field_v1`. The release is computed from results cut `20261020_123659`. A results cut is the set of result files built from one frozen copy of the benchmark store.

3. Scale and integrity

The release rates 85 models, each at up to three reasoning settings. We count a *program* as one distinct piece of admitted code for one model, game and setting. Identical code admitted twice counts as one program and plays its matches once, but in the model's score it still counts for both attempts. The release's 1,855 programs played 428,924 rated matches. A rated match is a game between two rated programs, played under the attested sandbox policy, in the newest benchmark version of its pairing. The benchmark store also holds older games, from pairings that were later replayed and from earlier sandbox policies. We keep them but do not count them (appendix B). An Overall rank goes to the 59 models, out of 85, that have three distinct reasoning settings scored. Among them, 52 are fully ranked and 7 are marked "Partial": they keep their rank, but part of their evidence is incomplete, and the site gives the reason beside each rank. In this release the reasons are a documented reasoning setting that was untested or ineligible, games whose only generation attempts failed for harness or provider reasons and cannot yet be repeated, generation quality below the publication threshold, and, for one model, a setting whose every attempt failed and so scores 0. The remaining models are listed without an Overall rank. Per-model uncertainty intervals for these ranks have not been validated yet, and this report makes no claim about how large a gap between two models must be to be reliable.

Hashes bind the results cut to the frozen copy of the store it was built from. The release's integrity card (its integrity summary file) records that the confidence gate and the redaction scan passed and lists no integrity incidents. An outside reader can recompute all 1,855 published per-game program scores to within 10^{-12} points

from the published ratings and reference-field files. Refitting the ratings themselves would need match-level data, which is not public.

We fixed several analyses in advance. After a pre-specified comparison in October 2026, the current rating method replaced its predecessor. For section 4, the summaries were committed before anyone read their numbers, and the external test in section 5 followed a protocol frozen before any analysis.

4. Results

4.1 The Overall leaderboard

The ten highest Overall scores (0–100) are:

Rank	Model	Overall
1	GPT-6 Astra	89.6
2	Claude Opus 5.5	88.6
3	Claude Fable 5.1	88.5
4	GPT-6.1 Sol	85.8
5	Claude Opus 5	78.1
6	GPT-6 Sol	74.5
7	GPT-5.6 Sol	68.3
8	Claude Fable 5	67.2
9	Muse Wonder	66.3
10	GLM-5.3	65.3

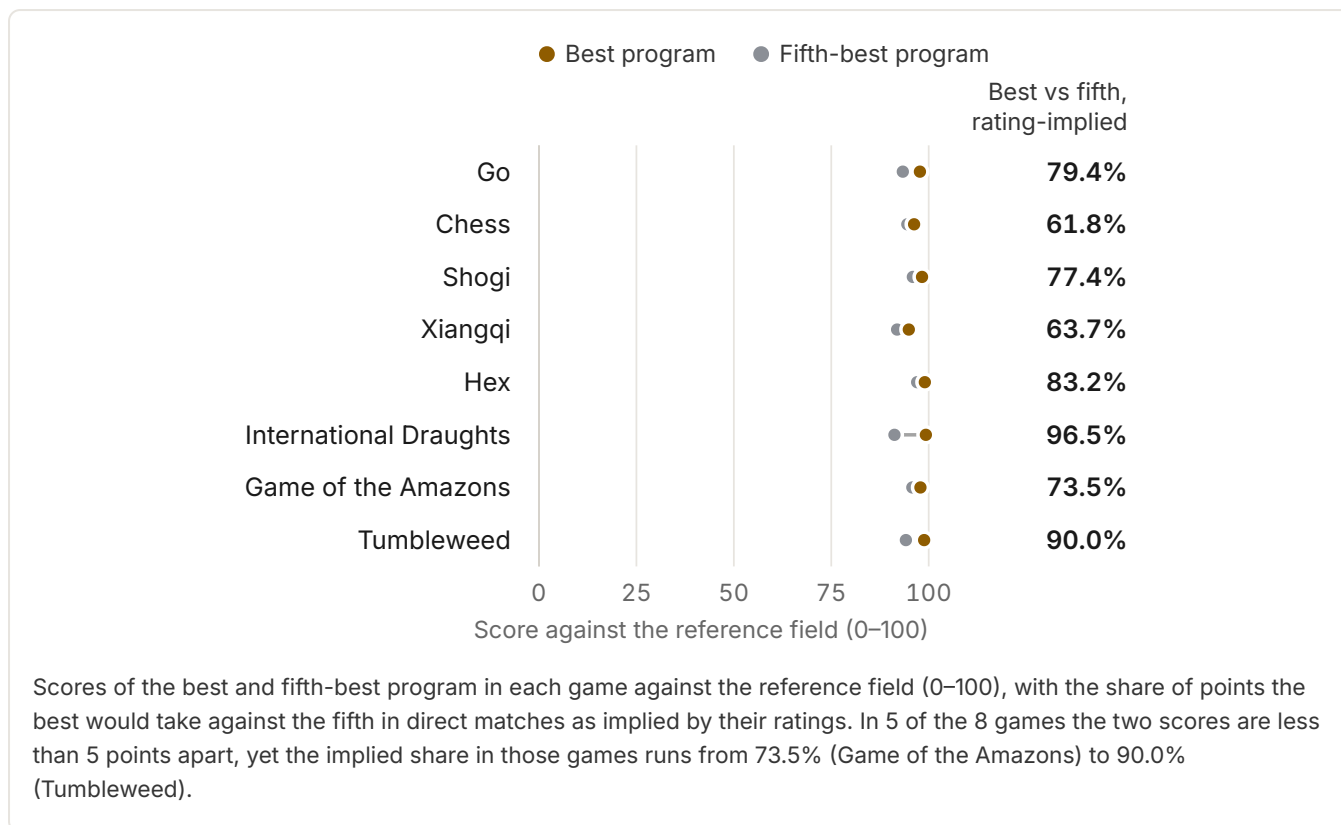
Close neighbours such as ranks 2 and 3 should be read as unresolved (section 3). The full table is published on the DuellLab site.

4.2 Separation among the leaders

The 0–100 score and the ratings measure separation differently: the score compares a program with a fixed set of reference opponents, while the ratings compare programs with each other.

The three leaders are close on the Overall score, at 89.6, 88.6 and 88.5, with the fourth at 85.8. Beyond the fourth model the gap widens sharply, to 78.1 for the fifth. A direct-match figure between two models would need a rule for weighting their games, settings and programs, so we compare models on the Overall score and compare programs head to head only within a game.

In 5 of the 8 games the best program scores within 5 points of the fifth-best against the reference field, yet its rating implies that it would take between 73.5% (Game of the Amazons) and 90.0% (Tumbleweed) of the points in direct matches against it. The 0–100 scale compresses these gaps because a program's score is its expected share of points against the fixed reference field. A program that already takes nearly all of those points has little room left on the scale, however much stronger it becomes. The ratings behind the scale have no such ceiling, and they still separate these programs. Two programs can both beat the reference opponents almost every time, and so both score close to 100, while one of them still clearly beats the other.



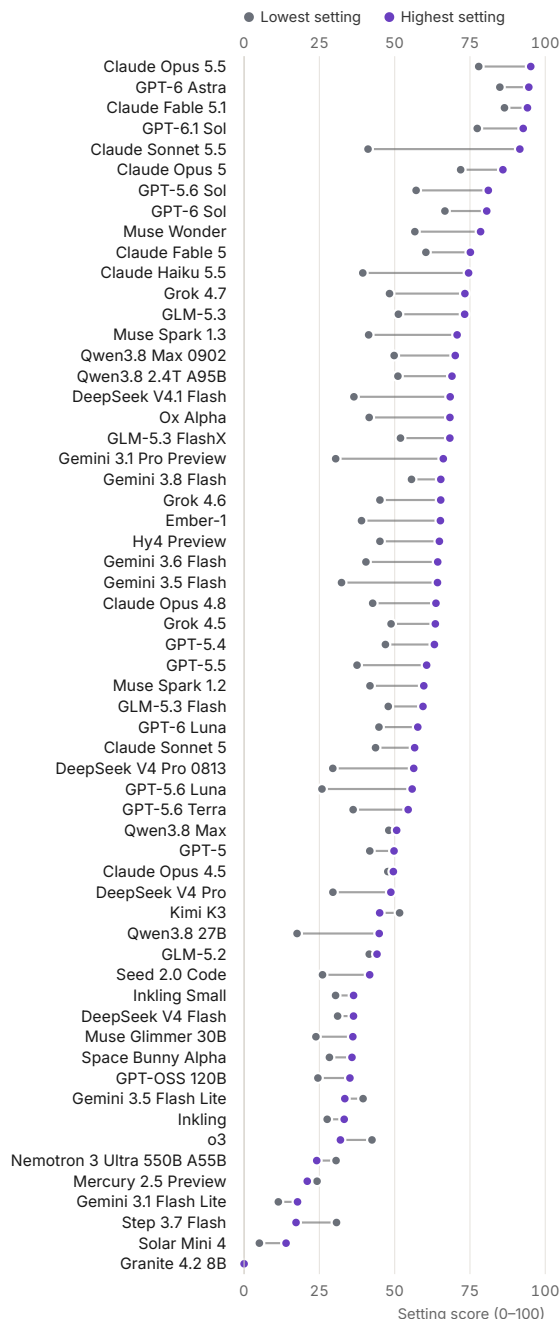
4.3 Effect of reasoning effort

The three settings come from each model's own options. Typically the baseline is no reasoning or the lowest level offered (None, Minimal or Low), the balanced setting is Medium, and the intensive setting is XHigh. Where a model offers both XHigh and Max, we use XHigh, the level users usually choose, and leave Max out; a model without XHigh uses Max if it has one, otherwise its highest level above Medium.

The table covers the 59 models with an Overall rank, each scored at all three settings. Because the settings are each model's own, "intensive" can change more than thinking length, and it differs from one provider to the next.

	Baseline	Balanced	Intensive
Median setting score	41.6	50.5	60.6
Generation attempts failing on the model's own code	9.9%	7.0%	9.7%

From its lowest setting to its highest, the median model gains 15.3 points; 52 of the 59 models gain at least a point and 6 lose at least a point. The strength of each model's working programs rises by a median 14.4 points over the same step, while the share of attempts that fail on the model's own code is about the same at both ends, so the gain comes mainly from stronger programs rather than fewer failures.



All 59 Overall-ranked models at their lowest and highest reasoning settings, sorted by the highest-setting score. From the lowest to the highest setting the median model gains 15.3 points; 52 gain at least a point and 6 lose at least a point.

4.4 Cost and strength

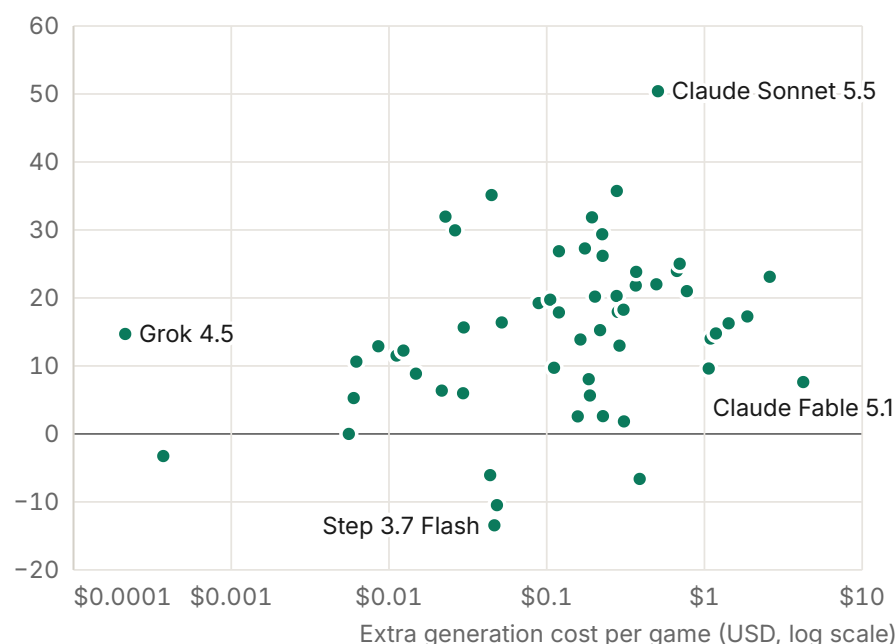
Across the 59 ranked models, generation cost per game and Overall score have a Spearman correlation of 0.64, and stronger models generally cost more. The correlation ranges from 0.46 with list-price estimates only to 0.69 with complete costs only. Costs are provider-reported amounts or list-price estimates for each generation run, not invoiced amounts. The cost–strength frontier, where no other model is both cheaper and stronger, has 9 members. Its cheapest point is a stealth preview (a model that a provider offers free under a code name before release). With a free model on the frontier, a single "cost per point" figure across models would mislead, and we do not report one.

The comparisons that follow stay within each model and leave free settings out. A setting's cost is its mean cost per game: the cost of one generation run for one game, with repairs and failed runs included. Its points per dollar are its score divided by that cost, which measures value against the reference field and nothing else. For

the 56 models whose highest setting cost more than their lowest, the step from lowest to highest gained a median 15.5 points for a median extra \$0.19 per game. Dividing each model's gain by its own extra cost gives a median of 71.6 points per extra dollar; as a median of ratios, it need not equal the median gain divided by the median extra cost. Because a ratio of two differences is sensitive to small differences in cost, we report only medians and counts. We left out 3 models: the two free stealth previews, recorded at \$0 for every setting, and one model whose highest setting cost no more than its lowest. All of these figures are per game; generating for all eight games costs eight times as much, whether or not every run yields a working program.

For most models, costs rise proportionally more than scores from one setting to the next, and that is why the lowest setting usually gives the most points per dollar. Among the 57 models with a recorded cost at every setting, it does so for 46, the middle setting for 11 and the highest for none. Returns per extra dollar also shrink from the first step up to the second. The first step yielded more for 34 of 47 models, with paired medians of 111 and 27.5 points per extra dollar; figures above 100 are possible because the extra costs are cents per game. Counting only complete costs puts the median for the full step at 108 points per extra dollar, and counting only provider-reported costs puts it at 140. On both bases the lowest setting still gives the most points per dollar.

Points gained, lowest to highest setting



Points gained from each model's lowest to its highest reasoning setting against the extra generation cost per game, for the 56 models whose highest setting cost more than their lowest. The median gain is 15.5 points and the median extra cost \$0.19 per game.

4.5 Reliability and failure modes

Most generation attempts (1,409 of 2,071) were admitted at the first try, and another 432 after repair. These attempts cover each model's three current settings. The 1,855 programs of section 3 also include 15 programs from an earlier, legacy reasoning configuration of Claude Opus 4.5, which play in the matches but are not in these attempt counts, and in one case two admitted attempts produced identical code, which counts as one program. The model's own code caused 212 failures (10.2%), at rates that were highest in Shogi (15.8%) and Chess (14.5%) and lowest in Hex (5.9%). The speed check rejected a further 18, which count in no score.

In matches, programs were charged with 215 faults. We report crashes and illegal moves together because the published files do not separate them. A few of the faults fell in games that have no result; appendix B counts these faulted games. The faults amount to 0.025% of program appearances and came from only 13 of the 1,855

programs. Draws at the move cap are far more common, ending 7.1% of program appearances overall, 18.4% in Go and 32.3% in Tumbleweed, the two games where such draws are most frequent. The Tumbleweed draws come from earlier engine versions, which stopped games at 500 moves.

4.6 Strongest program in each game

Game	Model	Setting	Field score
Go	GPT- 5.4	balanced	97.8
Chess	GPT- 6 Astra	intensive	96.3
Shogi	GPT- 6.1 Sol	intensive	98.3
Xiangqi	GPT- 6.1 Sol	intensive	94.9
Hex	Claude Opus 5	intensive	99.0
International Draughts	Claude Opus 5.5	intensive	99.3
Game of the Amazons	Claude Opus 5.5	intensive	97.9
Tumbleweed	Claude Sonnet 5.5	intensive	98.8

Six different models hold the eight top spots, and the strongest Go program came from a balanced setting. Several leads remain open. In four games the ratings imply that the top program would take under 56% of the points in direct matches against the runner-up: Chess (50.9%), Game of the Amazons (53.2%), Xiangqi (53.9%) and Hex (55.7%).

4.7 Progress over time

To see how quickly the best scores have improved, we placed each ranked model at its release date as recorded in Epoch AI's model data. That source has a reliable date for 45 of the 59 ranked models. The other 14, among them Muse Wonder (rank 9) and both stealth previews, are left out. Because every score comes from this one release, what follows is a cross-section by release date, not a history of past leaderboards.

Best Overall score among the dated models released by each date:

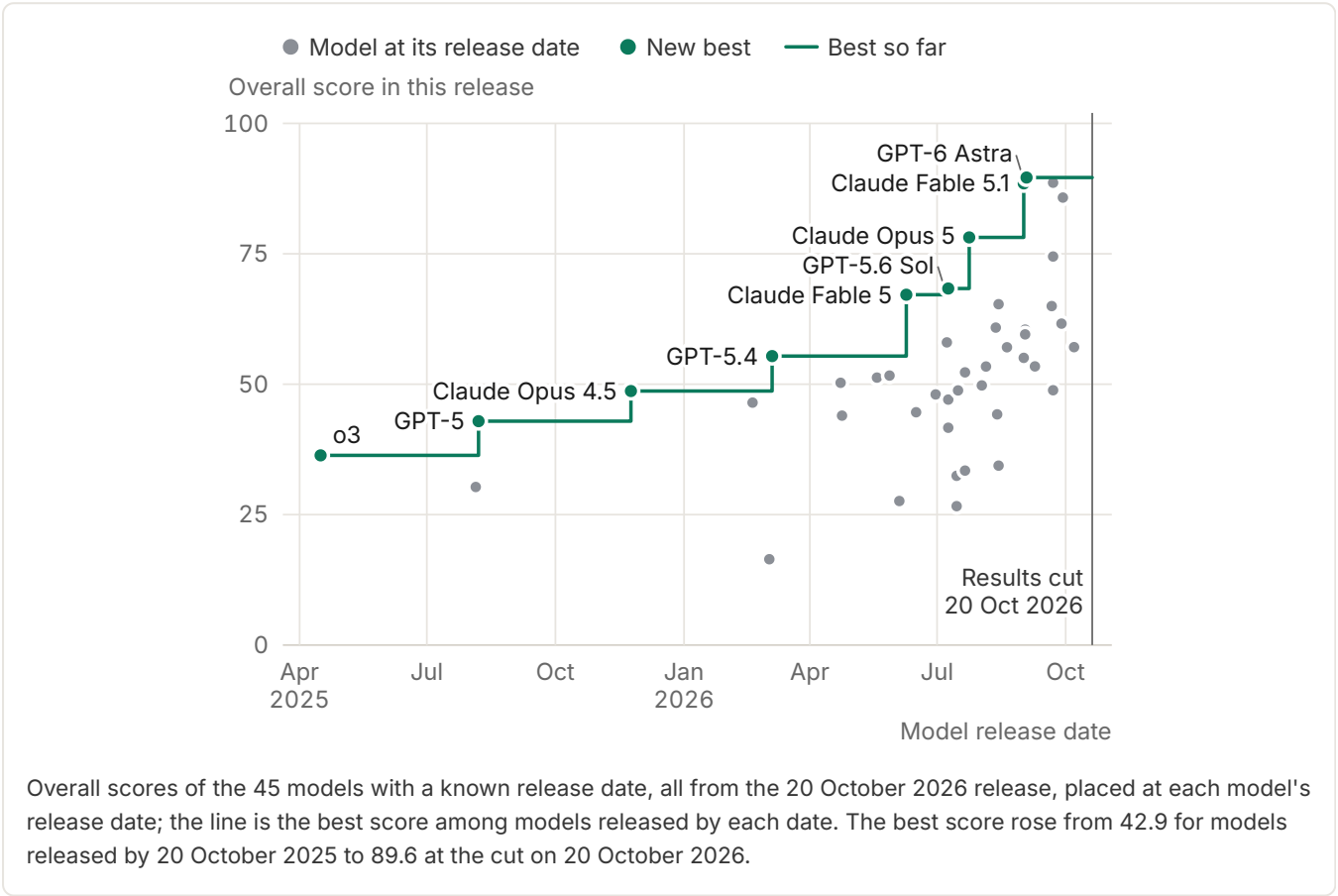
Released by	Dated models	Best Overall score	Model
2025-06-30	1	36.4	o3
2025-09-30	3	42.9	GPT- 5
2025-12-31	4	48.7	Claude Opus 4.5
2026-03-31	7	55.4	GPT- 5.4
2026-06-30	15	67.2	Claude Fable 5
2026-09-30	44	89.6	GPT- 6 Astra
2026-10-20 (this release)	45	89.6	GPT- 6 Astra

Among the models we evaluated, the best Overall score rose by 46.7 points in the 12 months before this release, from 42.9 for models released by 20 October 2025 to 89.6 for models released by 20 October 2026. Models released in the last six months account for 34.2 points of that rise. For 4 models the match to Epoch AI is looser, because their Epoch AI entry also pools results from other variants of the same model. Both rises still hold without these models, and the best score at this release is then 88.6.

Every new best came from two developers. The 9 models that beat every model released before them are 5 from OpenAI and 4 from Anthropic.

OpenAI also made all 3 dated models released by October 2025, while 29 of the 45 came out in July–September 2026. The early part of the curve rests on very few models. The size of the 12-month rise depends on which older models we chose to evaluate, so it should not be read as a yearly rate.

At each model's highest setting, the best score is already 95.2, 4.8 points below 100, and the last two records there added 0.5 and 0.6 points. The Overall score and the pace of new records show no saturation, but at the highest settings the 0–100 scale is close to its top, where it compresses (section 4.2). The ratings behind it still separate the top programs, and we plan to keep the scale informative (section 7).

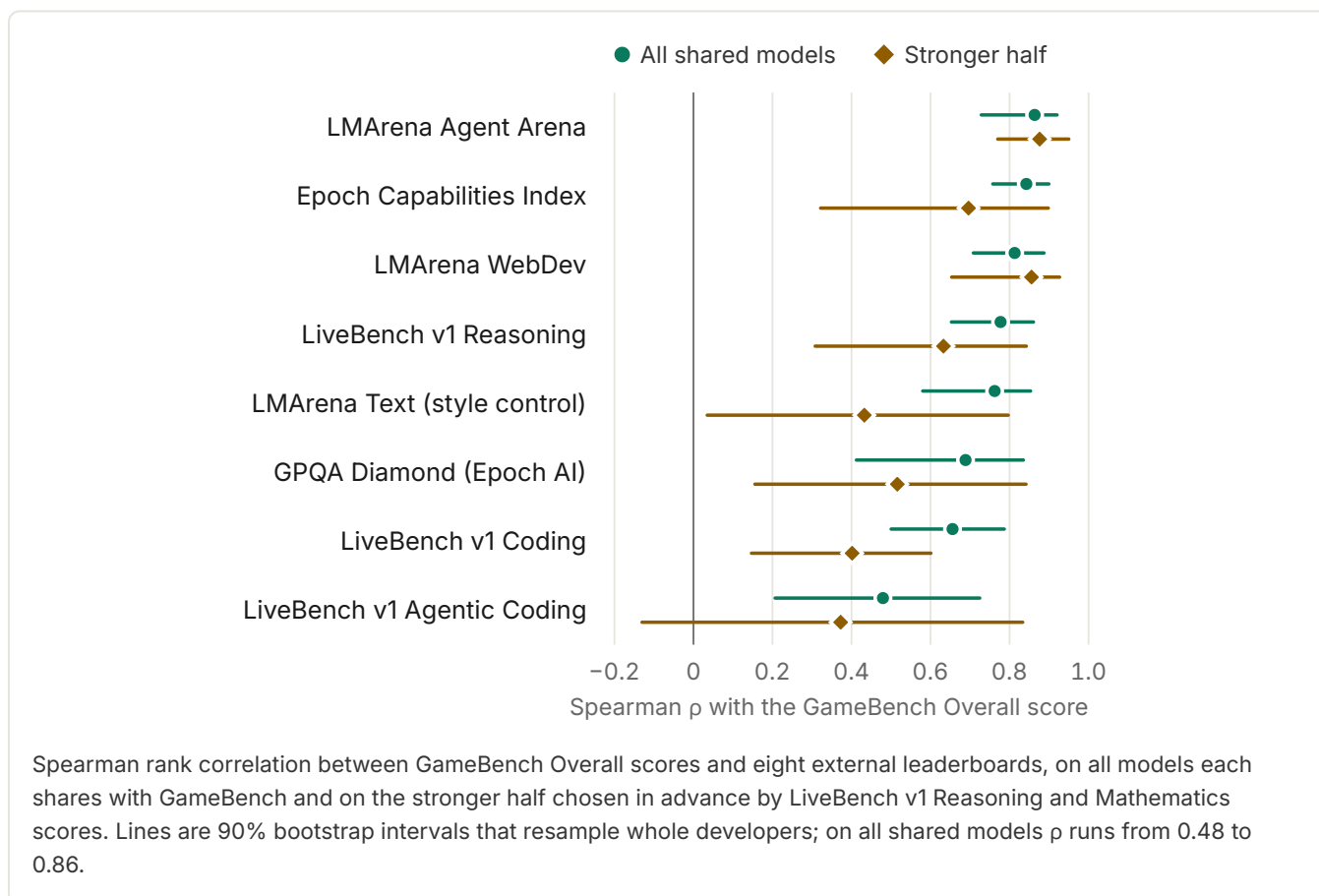


5. How GameBench compares with popular benchmarks

We compared GameBench Overall scores with well-known leaderboards, each time on the models that leaderboard shares with GameBench, matching models by exact identity and never by name similarity alone. The intervals are 90% bootstrap intervals that resample whole developers. In the last two columns each comparison is repeated on the stronger half of the shared models, chosen in advance by their LiveBench v1 (2026-06-25) Reasoning and Mathematics scores; these subsets are small, and restricting the range usually lowers correlations.

Compared with	Models / developers	Spearman ρ (90% interval)	Stronger half: models / developers	Stronger half: ρ (90% interval)
LMarena Agent Arena	43 / 16	0.86 (0.73–0.92)	18 / 5	0.88 (0.77–0.95)
Epoch Capabilities Index	52 / 13	0.84 (0.76–0.90)	20 / 7	0.70 (0.32–0.90)
LMarena WebDev	52 / 15	0.81 (0.71–0.89)	20 / 6	0.86 (0.65–0.93)
LiveBench v1 Reasoning	46 / 13	0.78 (0.65–0.86)	23 / 6	0.63 (0.31–0.84)
LMarena Text (style control)	58 / 17	0.76 (0.58–0.85)	21 / 6	0.43 (0.03–0.80)
GPQA Diamond (Epoch AI)	51 / 12	0.69 (0.41–0.84)	20 / 6	0.52 (0.16–0.84)
LiveBench v1 Coding	46 / 13	0.66 (0.50–0.79)	23 / 6	0.40 (0.15–0.60)
LiveBench v1 Agentic Coding (target of the test below)	34 / 12	0.48 (0.21–0.72)	17 / 4	0.37 (–0.13–0.83)

Across all shared models the agreement is broad, which is expected, since stronger models tend to be stronger everywhere. Within the stronger halves it stays high only for LMarena's Agent Arena and WebDev and falls for the other six comparisons, including LiveBench v1 Agentic Coding, the target of the test below, though these are small samples with wide, overlapping intervals. All LMarena leaderboards rank models by human pairwise preference votes rather than by task success. The Epoch Capabilities Index combines many benchmarks, among them GPQA Diamond and Chess Puzzles. Chess Puzzles is itself a game task and may add to the index's agreement with GameBench.



Agreement alone does not show that GameBench adds new information. Before seeing any results, we wrote down and froze a stricter test: with each model's whole developer held out, does adding GameBench to LiveBench's own scores improve predictions of LiveBench's agentic coding results? On that main target neither

GameBench nor anything else we tried beat a constant guess, so the test could not tell either way. In one of six secondary comparisons declared in advance, on LiveBench 2.0's coding tasks built from real pull requests, adding GameBench lowered the prediction error from 9.72 to 7.34 points, by 24%. We treat that as a lead to confirm on new data. The analysis ran once, on 34 models from 12 developers, and its protocol is kept internally.

6. Limitations and scope of claims

GameBench measures strategic problem solving through programs that models write for eight games. Each score rates the programs a model actually wrote. Most combinations of model, game and setting have one generation attempt, so a setting's score usually rests on eight programs, one per game, and another run could come out differently. Even with samples this small, GameBench ranks models much like well-established benchmarks (section 5). We have not measured the benchmark's resistance to training-data contamination.

The prediction test in section 5 rests on 12 developers, two of whom supply almost half the models, and its results can shift with a single developer. It is a retrospective held-out prediction, not a forecast of models released later.

Near its top the 0–100 score compresses (section 4.2), and at its highest setting the best model already scores 95.2 (section 4.7). The progress view in section 4.7 compares models released at different times within this one release and contains few older models. Per-game results can still change until every pairing has been replayed under the newest engine version, and without validated per-model intervals we cannot say how large a gap between two models must be to be reliable.

7. Roadmap

Each program today comes from a single model response, with a fixed budget for repair attempts. We plan benchmarks in which a model works on its program over a longer session: testing it, studying its losses and improving it against stronger opponents. We also plan new games, to keep the challenge growing as models improve, and fixed reference players (conventional engines) to anchor scores beyond the model field.

References

- **CodeClash.** Yang, J., Lieret, K., et al. "CodeClash: Benchmarking Goal-Oriented Software Engineering." 2025. arXiv 2511.00839. <https://arxiv.org/abs/2511.00839>
- **ProxyWar.** Peng, W., Wang, X., Wu, Q. "ProxyWar: Dynamic Assessment of LLM Code Generation in Game Arenas." ICSE 2026. arXiv 2602.04296. <https://arxiv.org/abs/2602.04296>
- **LiveBench.** White, C., Dooley, S., Roberts, M., et al. "LiveBench: A Challenging, Contamination-Limited LLM Benchmark." arXiv 2406.19314. <https://livebench.ai/>
- **Chatbot Arena / LMArena.** Chiang, W.-L., et al. "Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference." arXiv 2403.04132. <https://lmarena.ai/>
- **GPQA.** Rein, D., et al. "GPQA: A Graduate-Level Google-Proof Q&A Benchmark." arXiv 2311.12022.
- **Epoch AI.** "Capabilities & benchmarking." Published online at epoch.ai. Retrieved from <https://epoch.ai/benchmarks> on 2026-10-20.
- **Chess.** Fraenkel, A. S., Lichtenstein, D. "Computing a perfect strategy for $n \times n$ chess requires time exponential in n ." Journal of Combinatorial Theory, Series A, 1981.

- **Go.** Robson, J. M. "The complexity of Go." Proceedings of the IFIP 9th World Computer Congress, 1983. Lichtenstein, D., Sipser, M. "Go is polynomial-space hard." Journal of the ACM, 1980.
 - **Checkers.** Robson, J. M. "N by N checkers is Exptime complete." SIAM Journal on Computing, 1984.
 - **Shogi.** Adachi, H., Kamekawa, H., Iwata, S. "Shogi on $n \times n$ board is complete in exponential time." Transactions of the IEICE, 1987 (in Japanese).
 - **Xiangqi.** Gao, Q., Xu, X. "Research on the computational complexity of $n \times n$ Chinese chess." ICGA Journal 38(1), 47–53, 2015.
 - **Hex.** Reisch, S. "Hex ist PSPACE-vollständig." Acta Informatica, 1981.
 - **Game of the Amazons.** Hearn, R. A. "Amazons is PSPACE-complete." 2005. arXiv cs/0502013. <https://arxiv.org/abs/cs/0502013>
 - **Tumbleweed.** Bahack, L. "The game of Tumbleweed is PSPACE-complete." 2022. arXiv 2206.09476. <https://arxiv.org/abs/2206.09476>
-

Appendix A. Data sources, licences and credits

- **GameBench 2.** DuellLab public release [gb2-20261020-7c5039941207](https://duelllab.com/releases/gb2-20261020-7c5039941207).
 - **LiveBench.** livebench.ai, v1 release 2026-06-25 and LiveBench 2.0 release 2026-10-14. Files were retrieved on 2026-10-20 and cited as LiveBench above. The data files carry no licence of their own. This report publishes only rank statistics and prediction errors derived from them, never LiveBench's per-model values.
 - **LMarena.** Data from the LMarena leaderboard dataset (<https://huggingface.co/datasets/lmarena-ai/leaderboard-dataset>, dataset commit [8b599a3c](https://huggingface.co/datasets/lmarena-ai/leaderboard-dataset/commit/8b599a3c)). It uses the latest Text leaderboard (style control, published 2026-10-15) and the WebDev and Agent leaderboards (published 2026-10-19). Licensed under CC BY 4.0 (<https://creativecommons.org/licenses/by/4.0/>). Changes: models matched to GameBench models; only rank statistics and prediction errors are published.
 - **Epoch AI.** Data from the Epoch AI Benchmarking Hub (Epoch Capabilities Index, GPQA Diamond and model release dates; https://epoch.ai/data/benchmark_data.zip, retrieved 2026-10-20). Licensed under CC BY 4.0 (<https://creativecommons.org/licenses/by/4.0/>). Changes: models matched to GameBench models; only rank statistics, prediction errors and the best GameBench score by release date are published.
 - **Artificial Analysis.** Artificial Analysis data are not used.
-

Appendix B. Counting rated matches

The release's integrity report is a release-internal file. Its mixed track (where programs from all three settings play one another) counts 450,371 rows, which is 21,447 more than the 428,924 rated matches. The scoring code accounts for the difference. Earlier sandbox policies produced 21,434 of the extra rows; those policies are not attested, so their rows are not scored. The remaining 13 are faulted games that have no result. We derived the first figure by reading the code; no published file records it. No published number is affected.

Appendix C. Evidence and reproducibility

Every number in this report was checked against DuellLab's internal evidence records before publication. Those records, the frozen protocol of section 5 and its freeze manifest are kept internally and are not published. The scores, ratings and match counts behind sections 3 and 4 are in the public release linked in section 2.

Any published data package will leave out LiveBench per-model values until LiveBench's redistribution terms are confirmed.

Published October 22, 2026. Data: October 20, 2026, 12:36 UTC. [PDF version](#).

Evidence: [immutable release](#) · [leaderboard data](#) · [game-level data](#) · [reference field](#) · [methodology](#).