DELEGATING AI EVALUATION TO AI-GENERATED AUTOMATA IN GAMES

Anonymous authors
Paper under double-blind review

ABSTRACT

AI benchmarks predominantly evaluate models based on their direct, interactive capabilities. However, a principled evaluation of AI usefulness should recognize that true intelligence externalizes cognition, creating durable artifacts that make future work reliable without continuous intervention. The central measure of an AI is shifting from what it can do through regular interaction in a live setting to the utility of the autonomous tools it can build to accomplish goals. We introduce STRAUTOMATA, a framework that evaluates models not as interactive players, but as builders of autonomous artifacts (executable policies or automata). We measure this artifact-generating utility within the space of well-known strategic board games. Across 20 distinct games, we sample 5,200 Python-based automata generated by 52 Large Language Models and execute over 2,500,000 offline, head-to-head matches to establish comprehensive Elo ratings. By separating code generation from execution, we ensure highly cost-efficient evaluation while testing vital LLM capabilities, placing our benchmark on the cost–alignment Pareto frontier.

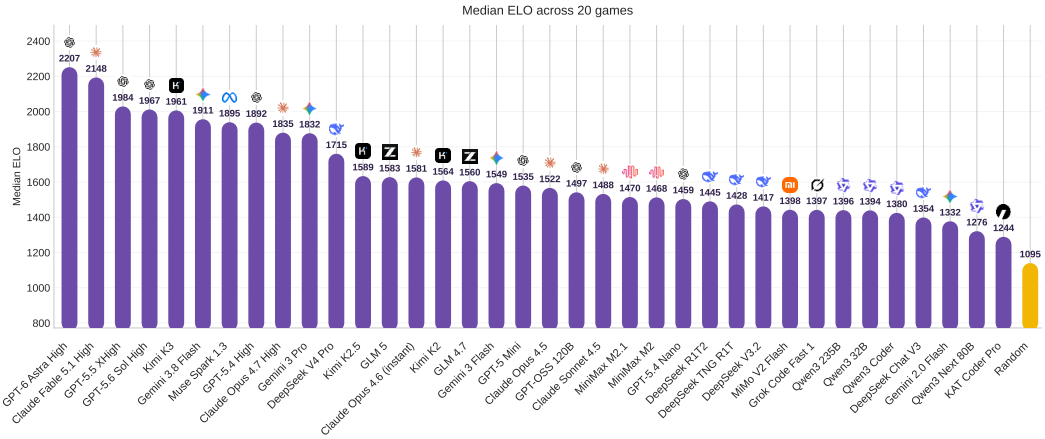


Figure 1: The median Elo across the 20 games $\mathcal{G}$ for a sample of models (All models are included in Section D). Elo scores for $\mathcal{M}$ models across $\mathcal{G}$ games (the full breakdown of the scores is in Table 4 and Figure 13). For each $(m, g) \in \mathcal{M} \times \mathcal{G}$ pair, only the qualified automata $(a \in \mathcal{A}_{g,m})$ are used to compute the mean Elo score of model m in game g ($Elo_{m,g}$). For each game g, 109,200 matches were played in a round-robin tournament (20 matches per pair of automata, not counting the qualifiers). The final median Elo score for a model m is obtained by taking the median of the 20 scores $\{Elo_{m,1}, \ldots, Elo_{m,20}\}$.
Note: the *Random* policy baseline is non-trivial to surpass, more details are provided in Section D.1.

We call a benchmark *saturated* when its top five models all score above 90%. We emphasize that STRAUTOMATA remains on the cost–alignment Pareto frontier whether *saturated* benchmarks are included

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.

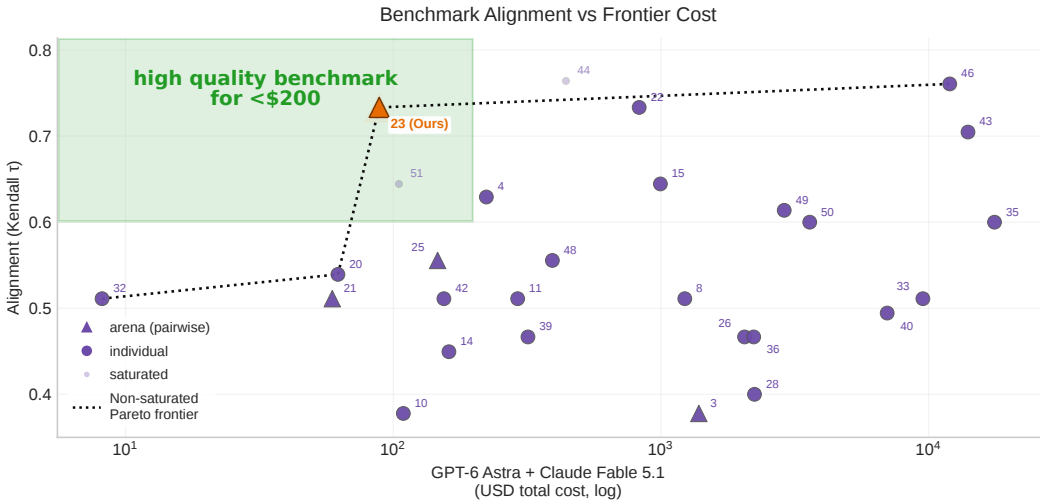


Figure 2: **Benchmark Alignment vs. Frontier Inference Cost.** Each point represents a benchmark. The x-axis shows the total inference cost, in USD, of evaluating the two frontier models (GPT-6 Astra and Claude Fable 5.1). The y-axis shows the alignment (Kendall's τ) between a given benchmark's ranking and the mean ranking across all other benchmarks in our collected benchmark pool (Perlitz et al., 2024b). Arena benchmarks have models competing directly against each other as opposed to most benchmarks where LLMs accomplish tasks individually. *Saturated* benchmarks have their top five models all achieving $\geq 90\%$. Approximately one quarter of benchmarks achieve an alignment score of ≥ 0.6. The Pareto frontier highlights the highest alignment achievable at a given LLM evaluation cost. STRAUTOMATA substantially advances the Pareto frontier at a cost of approximately $\$100$. Additional details are provided in Section C.

1 INTRODUCTION

As foundation models become more capable, the central question for both research and deployment is no longer '*how smart is the model?*' but '*how useful is the artifact it produces?*' The distinction matters: the most knowledgeable or benchmark-dominant agent is not necessarily the one that reliably produces value in the real world. In practice, usefulness is mediated by what persists after the interaction: code patches, plans, scripts, designs, checklists, proofs, configuration files, and other *artifacts* that can be executed, verified, reused, and composed. Increasingly, the "work product" of an AI system is not a sequence of tokens exchanged in a dialogue, but a set of artifacts that enable downstream progress at low marginal cost (Liang et al., 2023; Bommasani et al., 2022; Jimenez et al., 2024a).

Most current evaluation paradigms, however, still treat models as if they act directly in an environment: answer a question, choose an action, predict the next state, or maximize reward under a fixed interaction protocol. This framing has produced valuable benchmarks and rapid iteration, but it also encourages a narrow view of capability and is costly (Chen et al., 2025).

Dataset scores and human judgments reward systems that look competent in the moment while overlooking whether the system can produce durable, high-leverage outputs. Moreover, average-case benchmark performance can be a poor proxy for impact on the particular problems we actually care about: in many domains, the value distribution is heavy-tailed, where solving a small number of critical instances dominates the utility of solving many routine ones (Silverberg & Verspagen, 2007). In such settings, it is not enough to know that a model reaches a certain accuracy on a curated set; what matters is whether it can produce the artifacts that make progress on consequential goals.

Useful intelligence externalizes cognition, creating scaffolding that makes future work cheaper and more reliable (Wölflein et al., 2025). Modern AI systems are beginning to exhibit the same pattern. When a model writes code that calls APIs, generates a tool to transform data, drafts a reusable plan template, or composes a suite of tests, it is *augmenting itself*, not by changing its weights, but by extending the system around it.

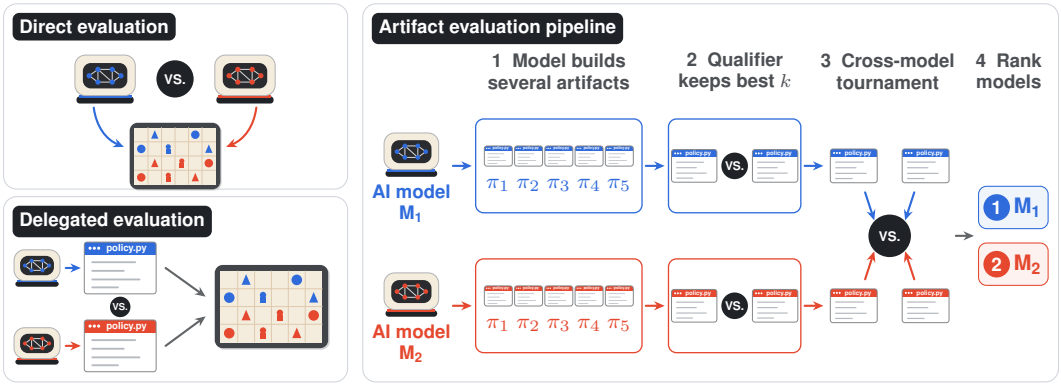


Figure 3: **Direct Evaluation**: AI models are evaluated by competing directly (typical benchmarks). **Delegated Evaluation**: AI models are evaluated based on the automata they produce in executable code (STRAUTOMATA). **Artifact evaluation pipeline for STRAUTOMATA**: LLMs generate artifacts (automata), which compete in intra-model qualifiers, followed by a cross-model tournament, and final ranking via Elo-based performance.

In this paper, we argue that **a principled evaluation of AI usefulness should center on the utility of the automaton a model can generate to accomplish goals**, rather than solely on direct interaction behavior. Concretely, we study settings in which an agent is tasked with achieving objectives, but is required to do so by producing automata that are then used by an execution process.

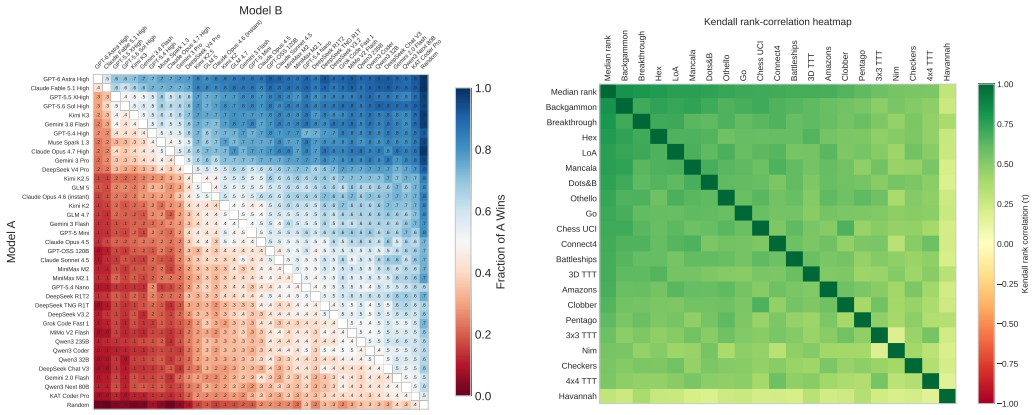
To operationalize this, we present an open-ended evaluation framework centered on artifact generation within game environments. Unlike traditional environments where agents are scored on direct interactions (e.g., achieving a high score through reflexes), our framework evaluates the agents based on the tools and structures they build. By shifting focus from intrinsic reasoning to externalized utility, we demonstrate that artifact generation differentiates models in a scalable and cost-effective manner which traditional static benchmarks struggle with. This approach moves us closer to an evaluation standard that mirrors real-world deployment, where the most valuable models are those that can effectively "harness" their environment to solve problems.

This separation between *artifact generation* and *artifact execution* provides three benefits. First, it makes the target of evaluation explicit: the object we score is the artifact's downstream contribution to goal attainment, not the model's ability to "talk through" a solution. Second, it encourages durability and compositionality: artifacts can be reused across episodes and combined with other components, mirroring real workflows where value accrues through accumulation of tools and infrastructure. Third, it offers a path to scalable assessment: by leveraging environments with programmatic interfaces and measurable objectives, we can evaluate outputs without reducing them to brittle reference answers.

2 RELATED WORKS

LLM Evaluation Benchmarks. The dominant paradigm for evaluating large language models relies on direct interaction: question answering (Hendrycks et al., 2021), mathematical reasoning (Cobbe et al., 2021), commonsense inference (Zellers et al., 2019), and multi-task benchmarks such as MMLU (Hendrycks et al., 2021) and BIG-Bench (Srivastava et al., 2022). While these benchmarks have driven rapid progress, they primarily measure correctness in answering a single specific question rather than the ability to produce functional, reusable artifacts. Benchmark agreement testing validates a benchmark by measuring how closely its rankings match those of established benchmarks. Perlitz et al. (2024b) show how such measurements should be made using rank correlation (Kendall τ) against an aggregate of many benchmarks. Under this view, a benchmark that agrees with the consensus at low cost is a good proxy for the field's overall judgment of model capability.

Code Generation Benchmarks. Evaluating code synthesis capabilities has received substantial attention, with benchmarks such as HumanEval (Chen, 2021), MBPP (Austin et al., 2021), and SWE-Bench (Jimenez et al., 2024b) measuring whether models can produce functionally correct programs. These benchmarks typically assess correctness via unit tests on isolated functions. STRAUTOMATA dif-



(a) Proportion of wins in all 20 games between models. The axes are sorted by median Elo (Figure 1).

(b) Inter-game ranking correlation. $A_{(i,j)}$ is the correlation between model rankings from game i and j.

Figure 4: (a) Proportion of wins across games and (b) correlation of model rankings between games.

fers by evaluating generated code in a competitive, adversarial setting where policies must generalize across diverse opponents rather than pass predetermined test cases.

Games as AI Testbeds. Strategic games have long served as benchmarks for artificial intelligence, from early work on checkers (Samuel, 1959) and chess (Campbell et al., 2002) to Go (Silver et al., 2016) and poker (Brown & Sandholm, 2019). These typically involve training specialized agents through reinforcement learning or search. A separate line of work evaluates LLMs as direct game players (Light et al., 2025), prompting models to select moves each turn. GTBench (Duan et al., 2024) and GameBench (Costarelli et al., 2024b) assess strategic reasoning through game-theoretic interactions, while GameBot (Lin et al., 2025) provides transparent evaluation of LLM reasoning in games. Chess has emerged as a particularly popular testbed, with ChessGPT (Feng et al., 2023) bridging policy learning and language modeling, and subsequent work (Kolasani et al., 2025) benchmarking reasoning through chess play. Grid-based game competitions (Topsakal et al., 2024) offer extensible leaderboards for LLM evaluation. Our framework differs fundamentally: rather than querying models at a decision point, we evaluate the executable policies that LLMs synthesize in a single generation step, shifting the focus from online deliberation to artifact quality.

Toward open-ended evaluation via automata. Across both code and game benchmarks, a common limitation is that evaluation protocols are often *closed*: either correctness is checked against fixed unit tests (Jimenez et al., 2024b), or performance is measured in a bounded set of prompts and opponents (Yu et al., 2025). As LLM capabilities advance, many static benchmark suites exhibit rapid saturation (Chen et al., 2025; Ghosh et al., 2025; Zhu et al., 2026; Akhtar et al., 2026), motivating an ongoing cycle of releasing harder variants and new datasets. In contrast, evaluating *automata* (executable policies produced by a model and then run autonomously) naturally supports open-ended, continuously extensible assessment. One can add new opponents, tournaments, games, or even entirely new environments without changing the core evaluation mechanism. STRAUTOMATA follows this direction by shifting from *LLM-as-player* (live, step-by-step querying (Costarelli et al., 2024a)) to *LLM-as-builder*, where the object of evaluation is a durable artifact whose quality is revealed through competitive interaction over time.

3 METHODS

Our framework centers on evaluating LLMs based on their capability to generate automata that autonomously behave in dynamic environments. In this case, the dynamic environments are board games, while the automata are Python programs that can output actions given any valid state in a particular game.

In this framework, once an automaton has been generated by an LLM, it is set in stone and must now compete against a wide variety of other automata in the environment, each of which has its own

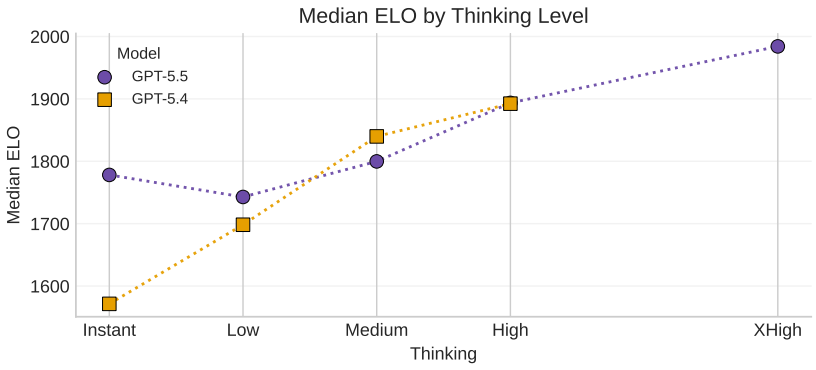


Figure 5: Relationship between thinking levels and performance. Higher levels of reasoning consistently improve performance, particularly in complex environments such as Go or Chess (full breakdown in Section D).

strategies and behaviors. Thus, for a single automaton to dominate all others within an environment, it must either employ a new strategy that surpasses all other automata or autonomously adjust its behavior to outperform each individual opponent. Both of these scenarios suggest that the LLM has higher capability.

Formally, let ($\mathcal{G} = g_1, g_2, \ldots, g_{20}$) denote the set of 20 board games we implement using the OpenSpiel library (Lanctot et al., 2019) as the simulator. For each game ($g \in \mathcal{G}$), we define a prompt that defines the rules of the game, the exact structure of the inputs, and the expected output format that the automata must follow. We also add environment meta-information to the prompt, such as the computational capabilities of the system executing the automata as well as a warning about the fixed time limit ($\tau = 1$ second) to prevent non-halting automata. An LLM deviating from these specifications, such as producing an automaton that returns malformed actions, illegal actions, or missing input fields, renders the automaton invalid and thus yields the lowest possible ranking for that automaton.

Let ($\mathcal{M} = m_1, m_2, \ldots, m_{52}$) be the set of LLMs evaluated. For each pair ($(m, g) \in \mathcal{M} \times \mathcal{G}$), we sample $\mathcal{A}_{g,m} = 5$ independent automata from the LLM m, yielding a total population

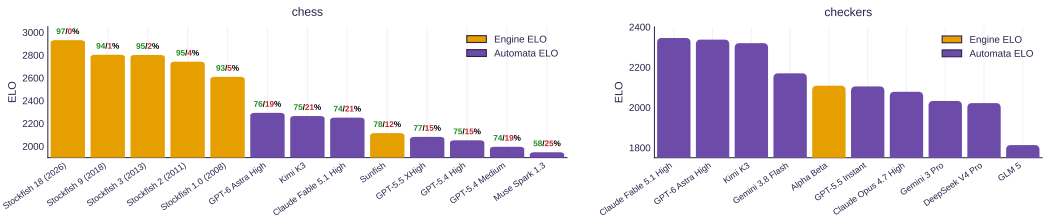
$$\mathcal{A} = \bigcup_{g \in \mathcal{G}} \mathcal{A}_g, \quad \mathcal{A}_g = \bigcup_{m \in \mathcal{M}} \mathcal{A}_{g,m}. \tag{1}$$

We have $|\mathcal{A}_g| = 260$ and $|\mathcal{A}| = 5200$. Each automaton ($a \in \mathcal{A}_g$) is a policy ($\pi_a : \mathcal{S}_g \to \mathcal{A}ct_g$) that maps any valid game state to a (possibly) legal action. Where both $\mathcal{S}_g$ and $\mathcal{A}ct_g$ are defined per game.

$$\pi_a : (\mathcal{S}_g, M_i) \to (\mathcal{A}ct_g, M_{i+1}) \tag{2}$$

For each game g, we evaluate all automata in $\mathcal{A}_g$ using a round-robin tournament. Let ($N_g = |\mathcal{A}_g| = 260$). For every unordered pair (a_i, a_j), we conduct two matches to eliminate first-move bias: one with (a_i) playing first and one with (a_j) playing first. The theoretical total number of matches per game is $r \cdot \binom{N_g}{2}$ for the full round-robin tournament before introducing any kind of qualifier phase. Here, r is the number of repeated matches per automaton pair.

Each match produces an outcome ($o \in \{-1, 0, +1\}$), corresponding to a win for a_j, a draw, or a win for a_i. Aggregating over all matches yields ordinal variables between every pair of automata. We then use the classical **Elo** rating system for estimating the relative skills of each automaton in g by fitting Bradley–Terry parameters to the pairwise outcomes similar to Chiang et al. (2024). Internally, each match can result in one of seven possible outcomes: a completed match ending in a win for one of the two competitors or a draw, an error attributable to either competitor (due to a runtime failure or an attempted illegal move), or a timeout incurred by either competitor for exceeding the time-limit hyperparameter (τ). Other than a draw, the remaining six outcomes yield a well-defined winner and loser. More details are given in Section A.



(a) Chess Elo: Top-7 Automata and multiple chess engines. Numbers represent the percentage of games the automata won and lost (not including draws). Exact matchup results for any two top automata included in Section G.

(b) Checkers Elo: Top-9 Automata and a checkers engine.

Figure 6: Elo ratings in the offline automata tournament for (a) Chess and (b) Checkers, comparing the top-n generated automata (purple) against strong hand-written engine baselines (orange). In both games the engine is an upper bound: it almost always wins, while the best automata cluster well below it. The gap is tighter for certain automata that can manage to draw against the engine.

Due to the combinatorial cost of the full round-robin, we introduce a qualifier phase. For each game g and model $m \in \mathcal{M}$, we run an intra-model qualifier over the candidate pool $\mathcal{A}_{g,m}$ to select the strongest k automata. Each unordered pair of automata from the same model continually plays against each other; we then estimate their Elo ratings with a bootstrap confidence interval with a conservative CI-separated ranking rule (Equation (11)). We repeat this process until exactly $k = 2$ automata satisfy the CI criterion. Implementation details of this qualifier phase are given in Section B and the exact match counts for each game and phase are reported in Table 11.

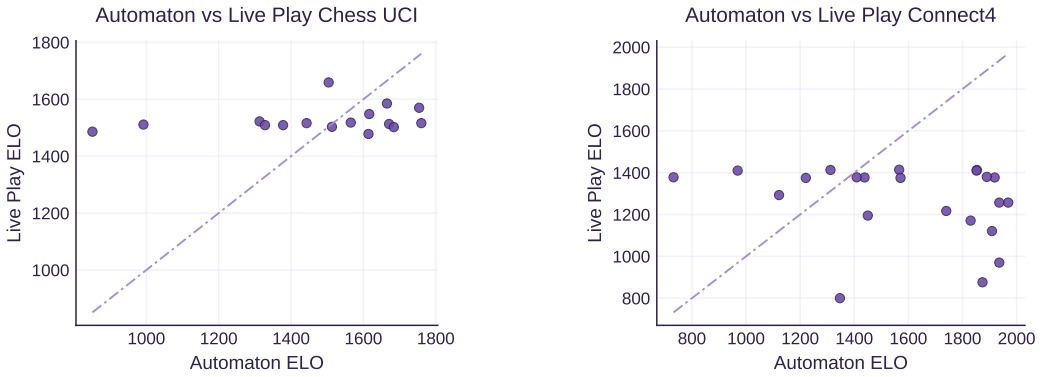
Once the qualifier phase has concluded, we obtain $k = 2$ automata from each LLM as well as the *Random* policy baseline for a total of 105 automata. Each unordered pair plays $r = 20$ matches with move order swapped, to eliminate first-move bias, for a total of $20 \cdot \binom{52 \cdot 2+1}{2} = 109{,}200$ matches per game. Once all matches have concluded in the round-robin tournament, for each $(m, g) \in \mathcal{M} \times \mathcal{G}$ pair we take the mean Elo of qualified automata from model m and consider it to represent $Elo_{m,g}$, the performance of model m in game g. Adding the qualifier round and reporting the mean automata Elo score greatly reduce noise and more accurately measure the model's strength in creating automata. The final reported median Elo score for a model m is obtained by taking the median of the 20 scores $\{Elo_{m,1}, \ldots, Elo_{m,20}\}$.

We report the median Elo across the 20 games for a sample of models in Figure 1 and list details for all models in Section D. Additionally, we highlight the consistent transitive property across models in Figure 4a where averaging across all games yields a consistent pattern where $a_{ij} > 0$ for $i < j$. Another important property is the correlation between the rankings of different games where all rankings are correlated, with most correlations being high (Figure 4b). We additionally report the change in Elo across different thinking levels in Figure 5 and include a correlation between our rankings and other individual popular benchmarks in Section F which shows a high correlation $(r > 0.70)$. Finally, we report a Perlitz et al. (2024b) style alignment metric vs cost of all benchmarks that provide the cost of frontier models in Figure 2. We show that STRAUTOMATA substantially advances the Pareto frontier. Additional details are provided in Section C including Table 2 of all included benchmarks and demonstrate that STRAUTOMATA advances the Pareto frontier among various different calculations of alignment.

4 LLMs Versus Human-Written Engines

Under this framework, a comparison between LLMs and humans would be to evaluate the model's automata against a human-written automaton. The human automaton is essentially a program to play the game, commonly known as an "engine". We compare the LLM automata to human engines for two games: Chess and Checkers. We show the results in Figures 6a and 6b.

Across both games, the human engines are amongst the top 10 of all sampled automata. Interestingly, in both Chess and Checkers, frontier LLMs manage to generate automata that are stronger than Sunfish in Chess and an Alpha Beta strategy in Checkers. No automata from an LLM released prior



(a) Live play Chess

(b) Live play Connect 4

Figure 7: **Automata vs. live-play Elo.** Each point is a model: the x-axis is the Elo of its best automaton (LLM-as-builder) and the y-axis is the Elo of the same underlying model when queried at every move during live play (LLM-as-player). In Connect4, most models fall below the diagonal, consistent with minimax-style compiled policies outperforming per-move prompting. In Chess, results are mixed, some models perform better live while others do better as automata.

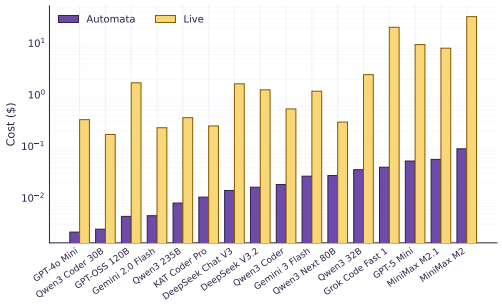
to June 2026 could beat these two human-written engines, which a small demonstration the pace of progress among frontier LLMs. In Chess, several frontier automata are now able to tie with Stockfish. However, no automaton has been able to win against any version of Stockfish, yet.

5 Live Play Performance

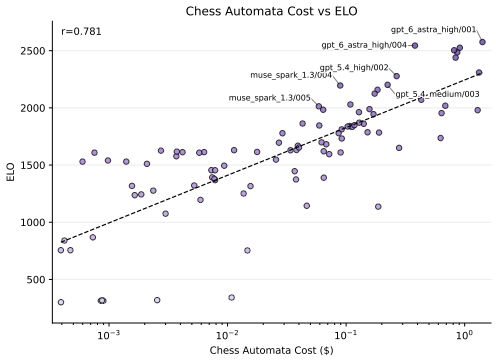
A core motivation for STRAUTOMATA is to evaluate models on the strategic strength of a *persistent artifact* (executable automata) rather than on transient, token-by-token interaction. To connect this artifact-centric view to the standard framework of LLMs directly accomplishing a task, we additionally measure *live play* performance: the same underlying model is placed directly in the game loop and must choose an action at each turn. This creates a clean contrast between (i) *LLM-as-player*, reacting online with no compiled policy and (ii) *LLM-as-builder*, producing a reusable automaton that then competes in head-to-head matches. We focus on chess and Connect 4 as canonical settings where strong baselines and well-understood notions of skill make the comparison interpretable.

Figure 7 summarizes the resulting picture. In the simpler game Connect 4, depth-limited minimax-style search is already highly effective, and the resulting automata strongly outperform live play. In the more difficult game chess, the comparison is less uniform: some models achieve higher Elo in live play than with their own generated automata, while other models cannot, and their automata remain stronger and more stable. Overall, the live-versus-automata gap depends on both game difficulty and on how effectively a given model can translate its online reasoning into a high-quality executable policy.

These results give a framing for agent evaluation; the same model can look markedly different depending on whether it is judged as an *interactive agent* or as a *tool-building system*. Live play emphasizes online deliberation and is sensitive to compounding errors. Automata emphasize design of a decision procedure which can effectively create an internal simulation of the environment and evaluate actions within. This shifts computation to an upfront "compile" step, yielding more stable play and, in some cases, higher overall strength. Practically, this indicates that benchmarking only live play can understate the capabilities that matter for real-world deployment, where the dominant value may come from producing durable artifacts that can be executed repeatedly and audited, rather than from reproducing reasoning from scratch at every step.



(a) Cost comparison of API cost to generate **five automata** generations vs. the *average* estimated API cost for the model to play **60 games** live (details in Section O). Unlike automata generation, which has a fixed cost, live-play cost grows *quadratically* with the participants.



(b) **Cost vs Elo for Chess.** The cost of a single automaton in Chess compared to its Elo in the Chess tournament.

Figure 8: **Chess cost analysis.** Left: API cost of generating five automata for a model compared with the average cost of playing 60 games. Right: per-automaton generation cost versus Chess Elo.

6 Cost Comparison

A key practical advantage of STRAUTOMATA is the cost efficiency where model usage is amortized into a one-time *compile step*. In the automata setting, for each model we pay a fixed API cost per game to generate a small set of executable automata programs (we sample $A = 5$ policies). After this generation phase, the LLM is no longer needed: the full round-robin tournament is run entirely offline by executing the produced code. Consequently, the marginal API cost of increasing the number of opponents is essentially zero once policies have been generated. One can evaluate against more competitors (or re-run tournaments under different schedules) without additional model queries. This also implies that a deprecated closed-source LLM can still participate in the arena and continuously have it face off against the newest available LLMs as the automata can be rerun without any additional access to the source LLM.

Live play, by contrast, is inherently interaction-costly. The model must be queried at *every* decision point in *every* match, so evaluation cost scales with both game length and the number of pairings. Under a round-robin protocol with N participants, the number of pairings grows as $O(N^2)$, and adding a competitor requires running a substantial fraction of matches to obtain updated Elo ratings. This creates a sharp cost asymmetry: automata evaluation is dominated by a fixed per-(model,game) generation budget, whereas live-play evaluation grows superlinearly with the tournament size.

Figure 8a illustrates this effect for Chess. "Automata" reports the API cost to generate the five automata required for a model to enter the automata tournament for a single game. "Live" reports the average estimated API cost for the same model to play a 60-game live round-robin schedule (two games against each opponent). Across models, offline automata generation is consistently cheaper, often by orders of magnitude, because it replaces repeated per-move querying with a single up-front compilation of a reusable decision procedure. More details and a comparison with simpler games are provided in Section O.

7 CONCLUSION

By decoupling artifact generation from execution, STRAUTOMATA establishes a highly cost-efficient paradigm for evaluating strategic reasoning. Unlike traditional live-play evaluations, where API costs scale quadratically with the number of tournament participants, our framework amortizes the computational expense into a single, fixed-cost compilation step. Once generated, these automata are evaluated in extensive offline tournaments without incurring additional model inference costs, allowing for scalable comparisons against thousands of opponents.

Furthermore, this methodology yields significantly richer signal than transient, token-by-token interaction. Because automata are subjected to a round-robin tournament, they are forced to demonstrate robustness against a wide variety of opposing strategies and behaviors rather than a context-specific strategy within a particular game instance. This shifts the evaluation focus from momentary performance to the model's ability to produce durable artifacts that function autonomously in dynamic environments.

8 LIMITATIONS

A limitation of STRAUTOMATA is its strictly "open-loop" generation protocol. The LLM interacts with the environment only once: it produces an automaton in a single "compile" step, after which the artifact is executed offline without further model intervention. We make this design choice intentionally to advance and lie on the benchmark cost-alignment pareto frontier and to also avoid introducing an additional confound—the choice of agent harness. Different LLMs may be optimized for different harnesses, making it difficult to attribute performance differences solely to the generated automata, to ensure a fair comparison across models, or to utilize harnesses in a cost-effective manner.

This design constrains the evaluation to a "zero-shot" coding capability, which may not fully reflect a model's potential for software engineering or strategic reasoning. In realistic deployment settings where cost isn't tightly constrained, useful intelligence involves an iterative cycle: creating a prototype, observing its behavior in the environment, and refining the code based on outcomes. By denying the model access to match results or execution traces, we prevent it from debugging malformed actions or adapting its strategy to specific opponents. Consequently, our results serve as a lower bound to the capabilities of models that could significantly improve their automata given thorough performance feedback.

9 IMPACT STATEMENT

This paper introduces a *creative automata* evaluation framework in which LLMs generate autonomous agents ("automata") that are executed in a controlled competitive environment. In this work, we restrict the framework to strategic games, which are well-scoped domains with low direct real-world externalities. The primary positive impact is scientific: STRAUTOMATA enables reproducible, scalable benchmarking of models' ability to produce durable decision procedures rather than only interactive responses. It also provides a testbed for studying agentic coding and generalization under competition.

At the same time, the broader framing could be applied beyond games. In particular, competitive settings can create strong optimization pressures. Such a framework could be misused to incentivize models to generate automata for harmful objectives and "racing" systems to perform harmful actions more effectively. While our current instantiation does not pursue such domains, this possibility is a foreseeable risk of generalizing the approach.

To mitigate these concerns, we emphasize (i) maintaining a narrow default scope (strategic games) for public releases and demonstrations and (ii) subjecting any extension to real-world domains—especially those with dual-use potential—to explicit risk assessment and responsible disclosure practices. Overall, we expect the primary societal effect of this work, as presented, to be improved evaluation methodology for machine learning systems, while noting that care is required to prevent competitive automata-generation benchmarks from being repurposed to advance harmful capabilities.

AI USE STATEMENT

In this work, we used generative AI tools to support qualitative and thematic data analysis by improving the styles of plots. We also used generative AI for running some of our experiments, implementing methods in code, and to give feedback on our presentation and research.

We have not used generative AI tools for generating synthetic datasets, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, assisting with translation, or interpreting results.

Additionally, we used generative AI tools for creating and modifying scientific figures, for brainstorming, and to suggest and make edits to the paper to improve readability. We have reviewed all AI-assisted work. All text outputs used in the paper were read by the authors and AI-generated code was verified and tested by the authors responsible for maintaining each section of the code. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide an anonymized code repository at:

`https://anonymous.4open.science/r/strautomata-codebase`

We note that all sampled policies from all LLMs in our experiments are contained in this repository along with all prompts used. All experiments in this paper utilized OpenRouter as the infrastructure provider for LLM access [1].

REFERENCES

Mubashara Akhtar, Anka Reuel, Prajna Soni, Sanchit Ahuja, Pawan Sasanka Ammanamanchi, Ruchit Rawal, Vilém Zouhar, Srishti Yadav, Chenxi Whitehouse, Dayeon Ki, et al. When ai benchmarks plateau: A systematic study of benchmark saturation. *arXiv preprint arXiv:2602.16763*, 2026.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. 2021. URL `https://arxiv.org/abs/2108.07732`.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models, 2022. URL `https://arxiv.org/abs/2108.07258`.

[1] https://openrouter.ai/

Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456): 885–890, 2019.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134 (1-2):57–83, 2002.

Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Simin Chen, Yiming Chen, Zexin Li, Yifan Jiang, Zhongwei Wan, Yixin He, Dezhi Ran, Tianle Gu, Haizhou Li, Tao Xie, and Baishakhi Ray. Benchmarking large language models under data contamination: A survey from static to dynamic evaluation. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10080–10098, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.511. URL https://aclanthology.org/2025.emnlp-main.511/.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*, 2024.

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

Anthony Costarelli, Mat Allen, Roman Hauksson, Grace Sodunke, Suhas Hariharan, Carlson Cheng, Wenjie Li, Joshua Clymer, and Arjun Yadav. Gamebench: Evaluating strategic reasoning abilities of llm agents, 2024a. URL https://arxiv.org/abs/2406.06613.

Anthony Costarelli, Mat Allen, Roman Hauksson, Grace Sodunke, Suhas Hariharan, Carlson Cheng, Wenjie Li, Joshua Clymer, and Arjun Yadav. Gamebench: Evaluating strategic reasoning abilities of llm agents. *arXiv preprint arXiv:2406.06613*, 2024b.

Jinhao Duan, Renming Zhang, James Diffenderfer, Bhavya Kailkhura, Lichao Sun, Elias Stengel-Eskin, Mohit Bansal, Tianlong Chen, and Kaidi Xu. Gtbench: Uncovering the strategic reasoning capabilities of llms via game-theoretic evaluations. *Advances in Neural Information Processing Systems*, 37:28219–28253, 2024.

Xidong Feng, Yicheng Luo, Ziyan Wang, Hongrui Tang, Mengyue Yang, Kun Shao, David Mguni, Yali Du, and Jun Wang. Chessgpt: Bridging policy learning and language modeling. *Advances in Neural Information Processing Systems*, 36:7216–7262, 2023.

Adhiraj Ghosh, Sebastian Dziadzio, Ameya Prabhu, Vishaal Udandarao, Samuel Albanie, and Matthias Bethge. ONEBench to test them all: Sample-level benchmarking over open-ended capabilities. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32445–32481, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1560. URL https://aclanthology.org/2025.acl-long.1560/.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

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?, 2024a. URL https://arxiv.org/abs/2310.06770.

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? 2024b. URL https://arxiv.org/abs/2310.06770.

Sai Kolasani, Maxim Saplin, Nicholas Crispino, Kyle Montgomery, Jared Quincy Davis, Matei Zaharia, Chi Wang, and Chenguang Wang. Llm chess: Benchmarking reasoning and instruction-following in llms through chess, 2025. URL https://arxiv.org/abs/2512.01992.

Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. OpenSpiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019. URL http://arxiv.org/abs/1908.09453.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic evaluation of language models, 2023. URL https://arxiv.org/abs/2211.09110.

Jonathan Light, Min Cai, Weiqin Chen, Guanzhi Wang, Xiusi Chen, Wei Cheng, Yisong Yue, and Ziniu Hu. Strategist: Self-improvement of llm decision making via bi-level tree search, 2025. URL https://arxiv.org/abs/2408.10635.

Wenye Lin, Jonathan Roberts, Yunhan Yang, Samuel Albanie, Zongqing Lu, and Kai Han. GAMEBoT: Transparent assessment of LLM reasoning in games. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7656–7682, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.378. URL https://aclanthology.org/2025.acl-long.378/.

Yotam Perlitz, Elron Bandel, Ariel Gera, Ofir Arviv, Liat Ein-Dor, Eyal Shnarch, Noam Slonim, Michal Shmueli-Scheuer, and Leshem Choshen. Efficient benchmarking of language models, 2024a. URL https://arxiv.org/abs/2308.11696.

Yotam Perlitz, Ariel Gera, Ofir Arviv, Asaf Yehudai, Elron Bandel, Eyal Shnarch, Michal Shmueli-Scheuer, and Leshem Choshen. Do these llm benchmarks agree? fixing benchmark evaluation with benchbench, 2024b. URL https://arxiv.org/abs/2407.13696.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gong-Jun Xu, and Mikhail Yurochkin. tinybenchmarks: evaluating llms with fewer examples. *ArXiv*, abs/2402.14992, 2024. URL https://api.semanticscholar.org/CorpusID:267897919.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Gerald Silverberg and Bart Verspagen. The size distribution of innovations revisited: an application of extreme value statistics to citation and value measures of patent significance. *Journal of Econometrics*, 139(2):318–339, 2007.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *TRANSACTIONS ON MACHINE LEARNING RESEARCH*, 2022.

O Topsakal, CJ Edell, and JB Harper. Evaluating large language models with grid-based game competitions: An extensible llm benchmark and leaderboard. arxiv, 2024.

Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelović, and Jakob Nikolas Kather. Llm agents making agent tools, 2025. URL https://arxiv.org/abs/2502.11705.

Boxi Yu, Yuxuan Zhu, Pinjia He, and Daniel Kang. Utboost: Rigorous evaluation of coding agents on swe-bench, 2025. URL https://arxiv.org/abs/2506.09289.

Moy Yuan, Chenxi Whitehouse, Eric Chamoun, Rami Aly, and Andreas Vlachos. Probelm: Plausibility ranking evaluation for language models. *ArXiv*, abs/2404.03818, 2024. URL https://api.semanticscholar.org/CorpusID:268987420.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

Longyuan Zhu, Hairan Hua, Linlin Miao, and Bing Zhao. Benchmark health index: A systematic framework for benchmarking the benchmarks of llms. *arXiv preprint arXiv:2602.11674*, 2026.

A ELO ESTIMATION AND CONFIDENCE INTERVALS

In the following section, we use similar calculations to Chiang et al. (2024).

For a fixed game g, let $\mathcal{D}$ be a multiset of recorded matches. Each match is an ordered pair of automata (i, j) and an outcome $s \in \{1, \frac{1}{2}, 0\}$, corresponding to win/draw/loss for the first player i. Let $N = |\mathcal{D}|$ denote the total number of matches. We fit Bradley–Terry parameters, with draws encoded as 0.5 and errors/timeouts as losses. Without loss of generality, we take $\theta_1 = 0$.

Bradley–Terry (BT). We assign each automaton i a latent skill $\theta_i \in \mathbb{R}$ and model

$$p_{ij} \;=\; \Pr(i \text{ beats } j) \;=\; \sigma\big(\alpha(\theta_i - \theta_j)\big), \qquad \sigma(x) = \frac{1}{1 + e^{-x}}, \qquad \alpha = \ln(b) \tag{3}$$

where $b > 1$ is a base hyperparameter. We aggregate $\mathcal{D}$ into T distinct outcome types indexed by $t = 1, \ldots, T$, where type t corresponds to an ordered matchup (i_t, j_t) with label $y_t \in \{1, \frac{1}{2}, 0\}$ and count c_t, with $\sum_{t=1}^{T} c_t = N$. The BT parameters are fitted by minimizing the weighted logistic cross-entropy:

$$\widehat{\theta} = \arg\min_{\theta} \sum_{t=1}^{T} c_t \Big(- y_t \log p_{i_t j_t} - (1 - y_t) \log(1 - p_{i_t j_t}) \Big) \tag{4}$$

We center and map skills to Elo via an affine transform

$$\bar{\theta} = \frac{1}{K} \sum_{i=1}^{K} \hat{\theta}_i \tag{5}$$

$$\widehat{E}_i = E_0 + S\,(\hat{\theta}_i - \bar{\theta}) \tag{6}$$

with baseline E_0 and scale S, where K is the number of automata included in the fit.

Bootstrap confidence intervals. For $r = 1, \ldots, R$, we generate bootstrap counts by

$$(c_1^{(r)}, \ldots, c_T^{(r)}) \;\sim\; \text{Multinomial}\Big(N,\; \Big(\frac{c_1}{N}, \ldots, \frac{c_T}{N}\Big)\Big) \tag{7}$$

refit BT to obtain $\widehat{\theta}^{(r)}$, and convert to Elo

$$\bar{\theta}^{(r)} = \frac{1}{K} \sum_{i=1}^{K} \hat{\theta}_i^{(r)} \tag{8}$$

$$\widehat{E}_i^{(r)} \;=\; E_0 + S\,(\widehat{\theta}_i^{(r)} - \bar{\theta}^{(r)}) \tag{9}$$

Given confidence level δ, we compute percentile intervals

$$L_i = \mathrm{Quantile}_{(1-\delta)/2}\Big(\{\widehat{E}_i^{(r)}\}_{r=1}^{R}\Big), \qquad U_i = \mathrm{Quantile}_{(1+\delta)/2}\Big(\{\widehat{E}_i^{(r)}\}_{r=1}^{R}\Big) \tag{10}$$

Using $[L_i, U_i]$, we define a conservative rank

$$r(i) \;=\; 1 + |\{j \neq i \;:\; L_j > U_i\}| \tag{11}$$

which permits ties when intervals overlap.

We use $E_0 = 1500$, $S = 400$, $\delta = 0.95$, and $R = 100$.

B QUALIFIER PHASE

To avoid the $\mathcal{O}(n^2)$ growth of the number of matches needed for a round-robin with n competitors, we instead employ a Qualifier Phase which filters the automata generated by a single model in a game from $\mathcal{A}_{g,m}$ to k (we have $k = 2$). Below we describe this qualifier phase.

For each model-game pair (m, g), we generate a candidate pool $\mathcal{A}_{g,m}$ of n automata and run a qualification procedure to select k representatives for the subsequent inter-model evaluation. The qualifier is based on repeated round-robin play within $\mathcal{A}_{g,m}$ followed by calculating the confidence intervals of the Elo from the matches so far to make a ranking. If the confidence intervals overlap too much (no clear top-k), then keep repeatedly playing until a clear separation forms.

Algorithm 1 details the procedure. In each iteration, every unordered pair (a_i, a_j) plays two matches, then we fit BT/Elo ratings (Equation (6)) and compute confidence intervals (Equation (10)) at level $\alpha = 0.95$. We then assign each automaton a conservative rank via Equation (11), which counts how many other automata have a lower confidence bound exceeding its upper confidence bound. This rule permits ties when confidence intervals overlap. Letting $Q = \{a \in \mathcal{A}_{g,m} : r(a) \leq k\}$, the procedure terminates if $|Q| = k$ (exactly k candidates are separated as top-k under the CI ranking).

CI-separated selection is not guaranteed to terminate: if multiple automata are (nearly) equal in strength, their confidence intervals may overlap even as more matches are observed. This behavior is especially common in low-complexity environments (e.g., tic-tac-toe). To ensure finite runtime, we impose a match cap of L games per pair. If the cap is reached before CI-separation yields exactly k qualifiers, we return the top-k automata according to the Elo point estimates using Equation (6).

Algorithm 1 Qualification via CI-Separated Elo Ranking (with match cap)

Require: Game g, model m, target k (top-k to select), confidence level $\alpha = 0.95$, max matches per pair L
Ensure: A set of qualifiers from $\mathcal{A}_{g,m}$ (exactly k)

Sample the candidate pool $\mathcal{A}_{g,m} = \{a_1, \ldots, a_n\}$ for game g from model m
Initialize match-history $\mathcal{H} \leftarrow \emptyset$
Initialize match counter $c \leftarrow 0$
while $c < L$ **do**
  for all unordered pairs (a_i, a_j) with $1 \leq i < j \leq n$ **do**
    Play one match of g with a_i first vs. a_j second; append outcome to $\mathcal{H}$
    Play one match of g with a_j first vs. a_i second; append outcome to $\mathcal{H}$
  end for
  $c \leftarrow c + 2$
  Compute Elo point estimates $\{\widehat{E}(a_i)\}_{i=1}^n$ from $\mathcal{H}$
  Compute confidence intervals $\{[L(a_i), U(a_i)]\}_{i=1}^n$ at level α
  for $i \leftarrow 1$ to n **do**
    $r(a_i) \leftarrow r(i)$ using Equation (11) ▷ Ties possible when intervals overlap
  end for
  $Q_{g,m} \leftarrow \{a_i \in \mathcal{A}_{g,m} : r(a_i) \leq k\}$
  if $|Q_{g,m}| = k$ **then**
    return $Q_{g,m}$
  end if ▷ Otherwise, CIs are too wide to uniquely identify top-k; continue
end while
(Termination by match cap)
Compute final Elo point estimates $\{\widehat{E}(a_i)\}_{i=1}^n$ from $\mathcal{H}$
Let $Q_{g,m} \leftarrow \mathrm{TopK}\big(\mathcal{A}_{g,m}, \widehat{E}, k\big)$ ▷ select k largest point estimates
return $Q_{g,m}$

Table 1: Benchmarks on the Pareto frontier of alignment vs cost in all four plots (Figure 9 and Figure 10). STRAUTOMATA is the only benchmark that is saturation resistant, needs no reprompting, and lies on all four frontiers.

ID	Benchmark	Saturation Resistant	No Reprompting	All-model frontier		Top-10 frontier	
				$/unit	Total $	$/unit	Total $
11	AA-Omniscience	✗	✓	✓	✗	✓	✗
12	SciCode	✗	✓	✓	✗	✗	✗
20	BugHunt	✗	✓	✗	✗	✗	✓
25	LLM Debate	✓	✗	✓	✗	✓	✗
32	SimpleBench	✗	✓	✗	✓	✗	✓
46	SRE-Bench	✗	✓	✗	✗	✓	✓
23	**Ours (Strautomata)**	✓	✓	✓	✓	✓	✓

C BENCHMARK COST-ALIGNMENT PARETO FRONTIER

A key aspect of STRAUTOMATA is its highly cost-efficient paradigm for evaluating vital LLM capabilities. To quantitatively measure this, we collected other benchmarks that publish an ordering of LLMs together with the evaluation cost of frontier models (or their input and output token counts).

Many come from the aggregators ARTIFICIAL ANALYSIS[2] and VALS AI[3] while the rest are individually published leaderboards. We collected every benchmark we could find that meets these criteria as of September 2026, dropping composite indices whose individual components are already in the pool (e.g. the Vals Index). Model names are canonicalised across sites.

Following the Benchmark Agreement Testing (BAT) protocol of Perlitz et al. (2024b), we convert each benchmark's scores to normalized ranks in $[0, 1]$ and, for each benchmark B, build a leave-one-out consensus as the mean normalized rank of every model over all benchmarks other than B. The alignment of B is Kendall's τ between its ranking and this consensus on the models they share. A model that appears on only one benchmark, which happens rarely, is dropped. We require at least five shared models. In total 125 benchmarks enter the consensus, of which 51 publish a usable cost for both GPT-6 Astra and Claude Fable 5.1 and appear in our calculations. Cost is the sum for these two models at a common effort setting, per evaluation unit and for one pass of the evaluated split, and for Artificial Analysis it is computed from published token counts.

We note that high alignment suggests that a new benchmark captures capabilities similar to those measured by other benchmarks in the pool (Perlitz et al., 2024b; Polo et al., 2024; Perlitz et al., 2024a), whereas low alignment may instead reflect measurement of a novel capability not captured by the existing benchmark pool (Perlitz et al., 2024b; Yuan et al., 2024).

Arena benchmarks are those where models compete directly against each other as opposed to most benchmarks where LLMs accomplish tasks individually (**Individual benchmarks**). Arena benchmarks have many desirable properties such as automatically evolving in difficulty as the LLMs become better at the task and being resistant to saturation. We call a benchmark *saturated* when its top five models all score above 90%. We emphasize that STRAUTOMATA remains on the cost–alignment Pareto frontier whether *saturated* benchmarks are included in the pareto frontier calculation or not. We decide to plot yet exclude them from the pareto calculation to highlight the fact that many traditional benchmarks are saturating and that dynamically evolving benchmarks (such as arenas) are becoming more important. Figure 2 showcases that STRAUTOMATA substantially advanced the benchmark cost–alignment Pareto frontier. All plotted benchmarks are listed in detail in Table 2.

We additionally plot the alignment vs cost per task in Figure 9. The cost per task plot is slightly less informative than the total cost as a benchmark could have a low cost per task yet have a high total cost and thus would require substantial costs to truly replicate the alignment which it achieves. We demonstrate that even in this plot, STRAUTOMATA remains on the cost–alignment Pareto frontier. Additionally, we repeat this while only considering the alignment of the top-10 LLMs in each

[2] https://artificialanalysis.ai
[3] https://www.vals.ai

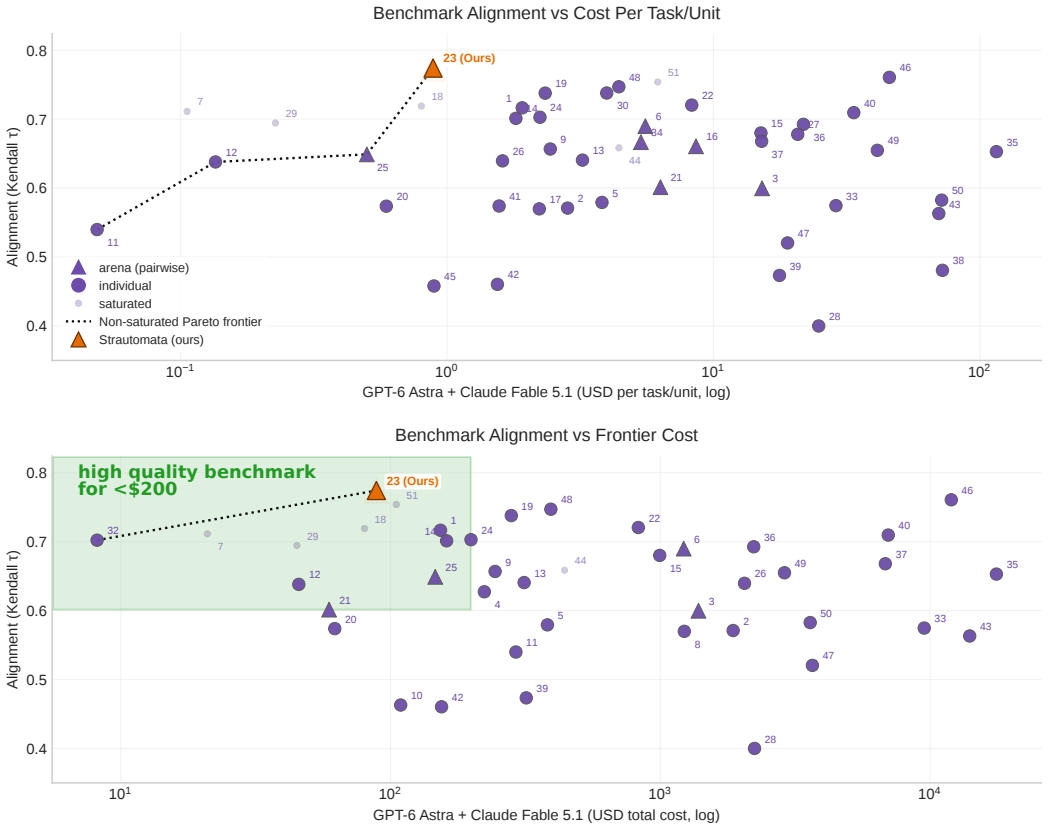


Figure 9: **Benchmark alignment vs. Frontier cost.** Alignment is measured using Kendall's τ with the leave-one-out benchmark consensus. Saturated benchmarks are shown but excluded from the Pareto frontier calculation. STRAUTOMATA remains on the cost-alignment Pareto frontier. Details in Section C.

leaderboard and report the two versions of the plot in Figure 10, STRAUTOMATA remains on the cost–alignment Pareto frontier for both. This illustrates that STRAUTOMATA is not only highly capable of ranking LLMs in general but also well suited to ranking frontier LLMs in a cost-efficient manner.

Finally, we list all benchmarks lying on any of the four Pareto Frontier calculations in Table 1. We define **Saturation Resistant** as benchmarks that dynamically become harder as LLMs get stronger (arena benchmarks satisfy this criterion). Another property is that the benchmarks require **no reprompting** of old LLMs when new LLMs join the benchmark. Non-arena benchmarks automatically satisfy this property as an LLM is only required once to achieve a grade and then never required again (unless the benchmark is changed). However, as previously mentioned, they suffer from saturation as newer models accomplish the majority of tasks successfully. STRAUTOMATA is the only benchmark in the Pareto Frontier which is simultaneously saturation resistant and requires no reprompting. Since evaluation is delegated to generated automata, previous LLMs don't need to be reprompted. Additionally, newer LLMs (regardless of ability) are still distinguishable given non-computationally solved games. This also means that scores of deprecated and discontinued closed-source LLMs continue to be graded and compared against all new LLMs in head-to-head battles. Finally, Table 1 showcase that STRAUTOMATA is the only benchmark simultaneously on all four Pareto Frontiers.

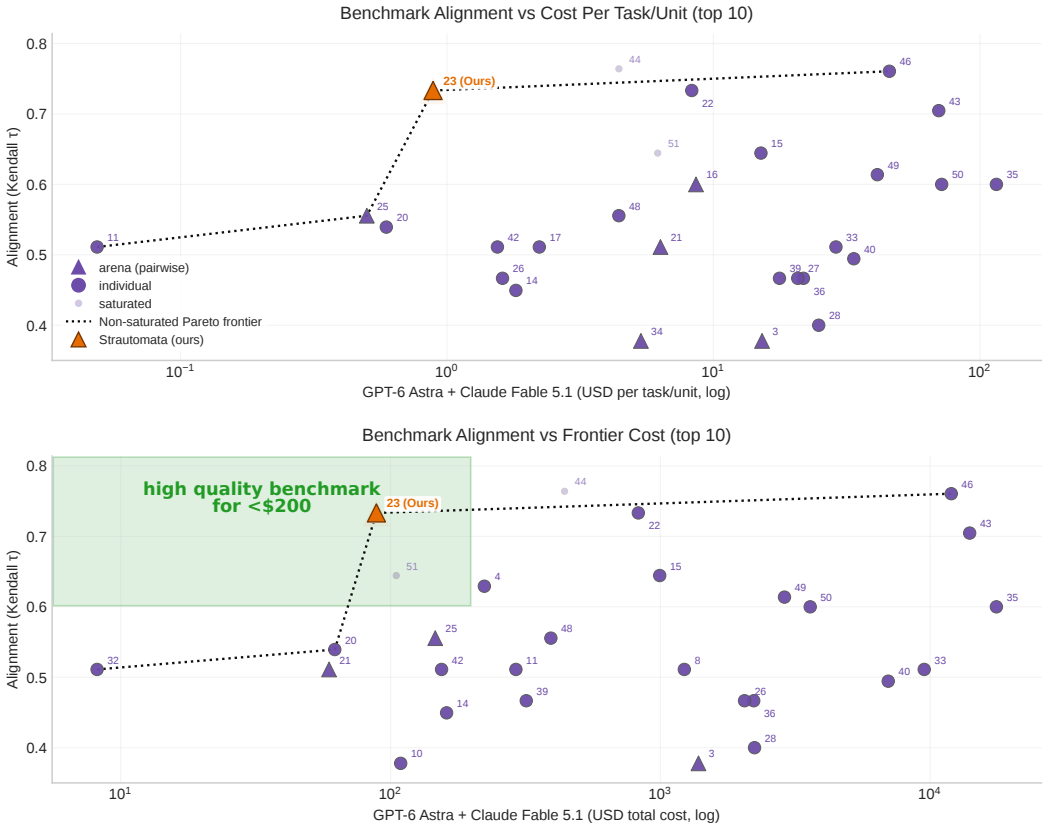


Figure 10: Similar to Figure 9 but only the top-10 models are considered. Details in Section C.

Table 2: Benchmarks in Figure 2 and Section C. *Models*: models considered in the benchmark. τ is Kendall's τ against the leave-one-out consensus of all other boards, on all shared models and on the top-10 models by consensus. Cost is GPT-6 Astra plus Claude Fable 5.1 in USD, per unit and for one pass of the evaluated split (n units). *Sat.*: Indicates a saturated benchmark where the top five models all score above 90%.

ID	Benchmark	Source	Type	Models	τ_{all}	τ_{10}	Cost/unit	Total	n	Sat.
1	AA Analyst Agent	Artif. Anal.	indiv.	31	0.72	0.29	$1.92 (task)	$153	80	
2	AutomationBench	Artif. Anal.	indiv.	46	0.57	-0.07	$2.83 (task)	$1,862	657	
3	AA Briefcase	Artif. Anal.	arena	47	0.60	0.38	$15.2 (task)	$1,385	91	
4	CritPt	Artif. Anal.	indiv.	338	0.63	0.63	– (run)	$223	–	
5	GDP-PDF	Artif. Anal.	indiv.	47	0.58	0.18	$3.82 (task)	$382	100	
6	GDPval-AA	Artif. Anal.	arena	173	0.69	0.29	$5.55 (task)	$1,221	220	
7	GPQA Diamond	Artif. Anal.	indiv.	409	0.71	-0.07	$0.11 (task)	$20.9	198	yes
8	HLE (AA)	Artif. Anal.	indiv.	403	0.57	0.51	– (run)	$1,227	–	
9	Long Context Reasoning	Artif. Anal.	indiv.	336	0.66	-0.02	$2.44 (task)	$244	100	
10	Multilingual LCR	Artif. Anal.	indiv.	37	0.46	0.38	– (run)	$109	–	
11	AA Omniscience	Artif. Anal.	indiv.	335	0.54	0.51	$0.05 (task)	$292	6000	
12	SciCode	Artif. Anal.	indiv.	122	0.64	0.04	$0.14 (task)	$45.7	338	
13	τ^3-Bench Banking	Artif. Anal.	indiv.	148	0.64	-0.51	$3.22 (task)	$313	97	
14	Terminal-Bench 2.1	Artif. Anal.	indiv.	174	0.70	0.45	$1.81 (task)	$161	89	
15	Terminal-Bench 4.0	Artif. Anal.	indiv.	109	0.68	0.64	$15.1 (task)	$995	66	
16	Agent Arena	other	arena	40	0.66	0.60	$8.60 (task, med.)	–	–	
17	AI Coding Daily	other	indiv.	24	0.57	0.51	$2.22 (prompt)	–	–	
18	ARC-AGI-1	ARC Prize	indiv.	85	0.72	0.34	$0.80 (task)	$80.1	100	yes
19	ARC-AGI-2	ARC Prize	indiv.	88	0.74	0.31	$2.34 (task)	$280	120	
20	Bug Hunt Bench	other	indiv.	28	0.57	0.54	$0.59 (run)	$62.1	105	
21	DuelLab GameBench 2	other	arena	57	0.60	0.51	$6.32 (game, wtd.)	$59.2	8/11	
22	FrontierCode	other	indiv.	35	0.72	0.73	$8.29 (rollout)	$829	100	
23	**Strautomata (ours)**	ours	arena	47	0.77	0.73	$0.89 (bot)	$88.6	100	
24	KiloBench	other	indiv.	38	0.70	0.18	$2.23 (run)	$199	89	
25	LLM Debate	other	arena	48	0.65	0.56	$0.50 (debate)	$146	278/302	

continued on next page

continued

ID	Benchmark	Source	Type	Models	τ_{all}	τ_{10}	Cost/unit	Total	n	Sat.
26	LiveBench	other	indiv.	55	0.64	0.47	$1.62 (task)	$2,052	1270	
27	MathArena	other	indiv.	86	0.68	0.47	$20.7 (problem)	–	–	
28	Normal CAD Arena	other	indiv.	6	0.40	0.40	$24.8 (task)	$2,235	90	
29	GPQA Diamond	OpenRouter	indiv.	122	0.69	-0.02	$0.23 (task)	$44.9	198	yes
30	PINNACLE	other	indiv.	32	0.74	0.11	$3.98 (correct)	–	–	
31	Real-SWE	other	indiv.	8	0.04	0.04	$11.6 (rollout)	$116	10	
32	SimpleBench	other	indiv.	92	0.70	0.51	– (run)	$8.17	–	
33	Terminal-Bench 4.0	tbench.ai	indiv.	14	0.57	0.51	$28.8 (run)	$9,511	330	
34	Three.js Eval	other	arena	16	0.67	0.38	$5.34 (prompt)	–	–	
35	Code Migration	Vals AI	indiv.	57	0.65	0.60	$115 (test, wtd.)	$17,610	130	
36	Excel Modeling	Vals AI	indiv.	54	0.69	0.47	$21.8 (test)	$2,219	102	
37	Finance Agent v2	Vals AI	indiv.	57	0.67	-0.16	$15.2 (test)	$6,824	450	
38	Harvey LAB	Vals AI	indiv.	58	0.48	-0.18	$72.4 (test)	–	–	
39	IOI	Vals AI	indiv.	25	0.47	0.47	$17.7 (test)	$318	18	
40	Legal Research	Vals AI	indiv.	57	0.71	0.49	$33.6 (test)	$6,996	208	
41	MedCode	Vals AI	indiv.	90	0.57	-0.07	$1.57 (test)	–	–	
42	MedScribe	Vals AI	indiv.	92	0.46	0.51	$1.55 (test)	$155	100	
43	ProgramBench	Vals AI	indiv.	43	0.56	0.70	$70.1 (test)	$14,018	200	
44	ProofBench	Vals AI	indiv.	31	0.66	0.76	$4.42 (test)	$442	100	yes
45	SAGE	Vals AI	indiv.	81	0.46	-0.16	$0.89 (test)	–	–	
46	SRE-Bench	Vals AI	indiv.	9	0.76	0.76	$45.7 (test)	$11,982	262	
47	Tax Agent Bench	Vals AI	indiv.	19	0.52	0.20	$19.0 (test)	$3,662	193	
48	Terminal-Bench 2.1	Vals AI	indiv.	59	0.75	0.56	$4.41 (test)	$393	89	
49	Terminal-Bench Science	Vals AI	indiv.	23	0.65	0.61	$41.2 (test)	$2,886	70	
50	Vibe Code	Vals AI	indiv.	86	0.58	0.60	$71.9 (test)	$3,594	50	
51	WeirdML	other	indiv.	121	0.75	0.64	$6.18 (run)	$105	17	yes

D LLM RATINGS AND RANKINGS PER GAME

Here we list the entire breakdown of each LLM's performance in terms of Elo Equation (6) and rankings Equation (11). Table 4 reports the Elo of each model in each game along with the median Elo in Figure 11. In Table 3 we report the rankings of each model in each game and visualize it (along with the mean) in Figure 13. Note that the Elo can be heavily skewed in certain games depending on the skill gap which is why we report the median Elo. Rankings, however, are not skewed even when the skill gap increases.

D.1 RANK OF *Random* POLICY

We note that the *Random* policy baseline is non-trivial to surpass because it is implemented to never make illegal moves and never causes timeouts. Consequently, an LLM-generated automaton must reliably infer and enforce game legality (effectively simulating the rules to enumerate valid actions) in order to outperform the *Random* policy. We report the rank of the *Random* policy in Figure 15 noting that in more complex games and games where a random legal move is sufficiently powerful, the *Random* policy tends to perform better.

D.2 CONFIDENCE INTERVAL

We explicitly restate that all ranks in Figures 13 and 14 and Table 3 were obtained using Equation (11) which calculates a bootstrapped confidence interval over the performance of the automata in the game and only decreases the rank once clear separability is achieved. This is why in simple games (e.g. Nim) almost half the models are tied at rank $\#1$ (breakdown in Table 3).

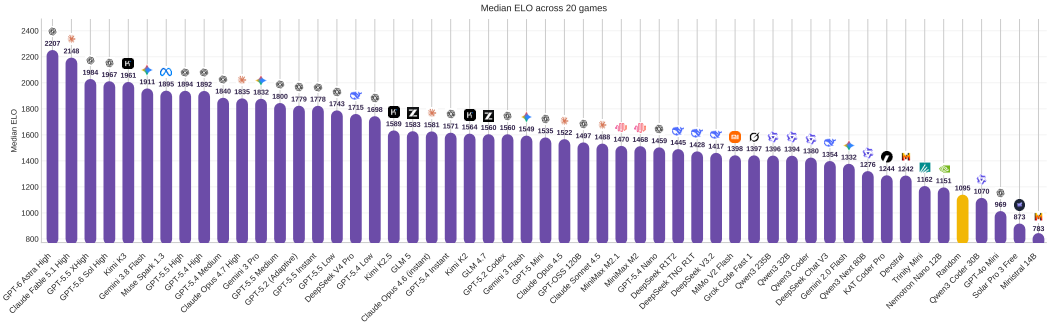


Figure 11: Similar to Figure 1 but all models are included.

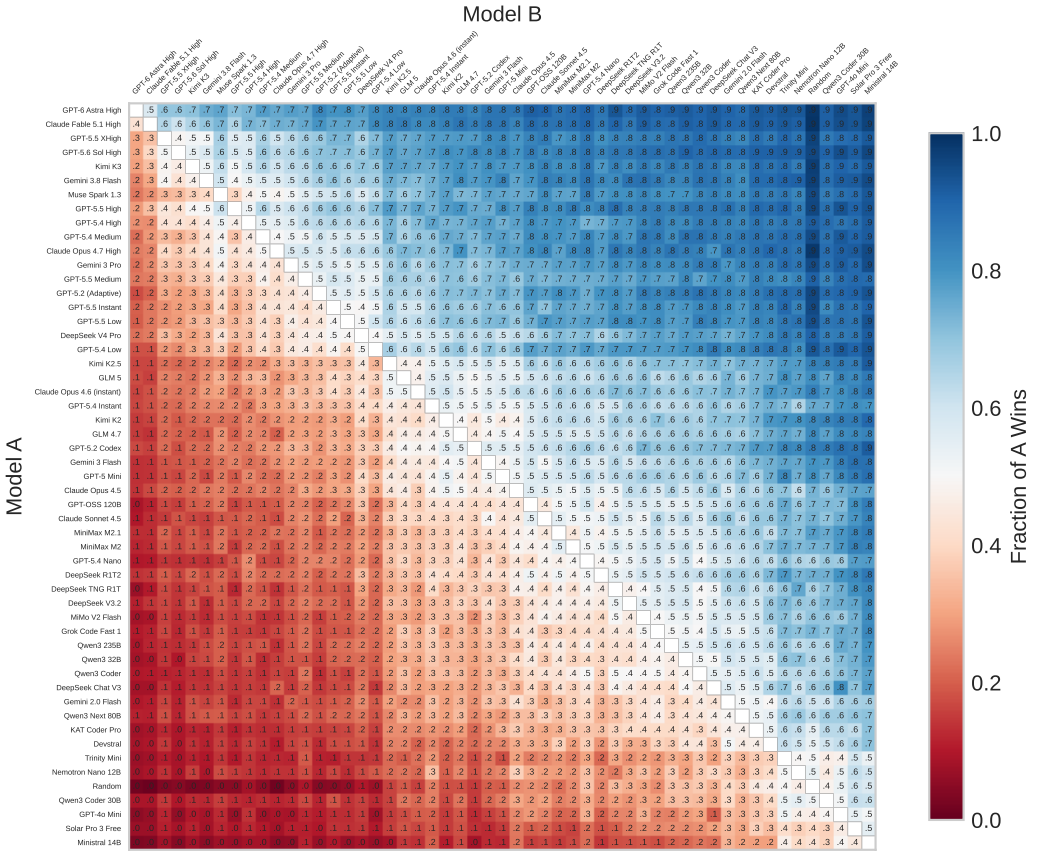


Figure 12: Similar to Figure 4a but all models are included.

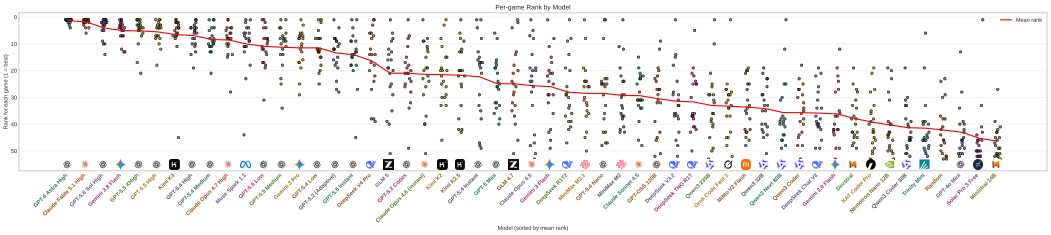


Figure 13: **Per-game rank by model across all games.** A visualization of Table 3. For each game $g \in \mathcal{G}$, we run the round-robin tournament and rank models *within that game* by their resulting Elo scores using Equation (11). Each dot corresponds to an $(m, g) \in \mathcal{M} \times \mathcal{G}$ pair and shows the rank (1 = best) of model m in game g *among all other models evaluated on the same game*, where the Elo for (m, g) is computed as described in Section 3. Models are ordered by their median per-game rank (red line).

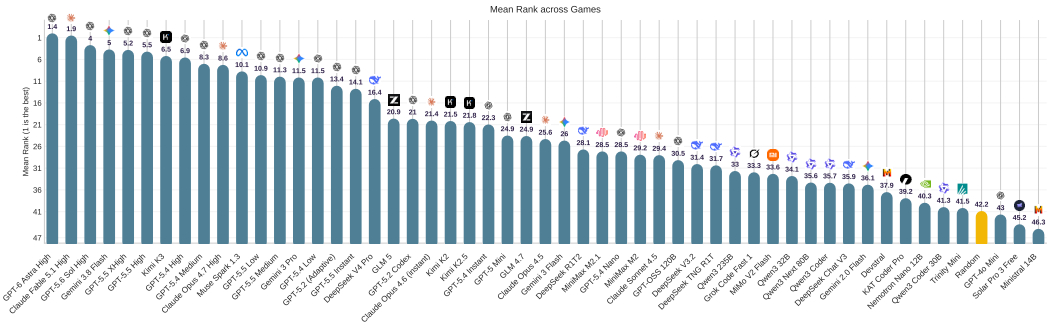


Figure 14: Similar to Figure 13 but only showing mean rank.

Table 3: The full breakdown of all games $\mathcal{G}$. Rankings for all $\mathcal{M}$ models across $\mathcal{G}$ games. For each $(m, g) \in \mathcal{M} \times \mathcal{G}$ pair the Elo is computed as described in Section 3. For each game g, Table 11 lists the number of matches played. Comparing a model's ranking to the ranking of the *Random* policy baseline is useful for indicating whether it was able to output an automaton capable of playing the game better than random chance.

	3D TTT	3x3 TTT	4x4 TTT	Battleships	Connect4	LoA	Mancala	Clobber	Pentago	Amazons	Backgammon	Nim	Othello	Dots&B	Breakthrough	Hex	Havannah	Go	Checkers	Chess UCI
Gpt 6 Astra High	2.0	**1.0**	**1.0**	**1.0**	2.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	3.0	4.0	**1.0**
Claude Fable 5.1 High	2.0	**1.0**	4.0	**1.0**	**1.0**	**1.0**	**1.0**	3.0	6.0	2.0	**1.0**	**1.0**	4.0	2.0	**1.0**	**1.0**	2.0	**1.0**	**1.0**	**1.0**
Gpt 5.5 Xhigh	2.0	**1.0**	4.0	6.0	4.0	3.0	**1.0**	17.0	**1.0**	4.0	**1.0**	**1.0**	3.0	7.0	3.0	5.0	21.0	4.0	10.0	5.0
Gpt 5.6 Sol High	2.0	**1.0**	**1.0**	3.0	5.0	9.0	**1.0**	3.0	**1.0**	2.0	5.0	**1.0**	2.0	4.0	10.0	2.0	3.0	2.0	14.0	10.0
Kimi K3	8.0	**1.0**	4.0	4.0	7.0	6.0	2.0	5.0	6.0	7.0	4.0	**1.0**	11.0	3.0	4.0	7.0	45.0	2.0	2.0	2.0
Gemini 3.8 Flash	7.0	**1.0**	2.0	7.0	4.0	3.0	4.0	4.0	**1.0**	7.0	4.0	**1.0**	5.0	3.0	5.0	3.0	13.0	12.0	2.0	13.0
Muse Spark 1.3	8.0	**1.0**	4.0	7.0	11.0	9.0	11.0	2.0	4.0	9.0	8.0	**1.0**	6.0	11.0	18.0	13.0	44.0	6.0	23.0	6.0
Gpt 5.5 High	7.0	**1.0**	4.0	4.0	4.0	4.0	4.0	18.0	**1.0**	2.0	3.0	**1.0**	3.0	7.0	8.0	8.0	4.0	7.0	10.0	12.0
Gpt 5.4 High	**1.0**	4.0	4.0	4.0	2.0	8.0	13.0	3.0	6.0	6.0	5.0	**1.0**	19.0	10.0	4.0	8.0	11.0	14.0	10.0	5.0
Gpt 5.4 Medium	**1.0**	**1.0**	4.0	7.0	21.0	9.0	11.0	13.0	8.0	11.0	8.0	**1.0**	10.0	13.0	9.0	10.0	5.0	11.0	10.0	4.0
Claude Opus 4.7 High	8.0	**1.0**	4.0	28.0	11.0	6.0	8.0	14.0	6.0	15.0	8.0	**1.0**	7.0	14.0	4.0	10.0	5.0	10.0	4.0	8.0
Gemini 3 Pro Preview	8.0	19.0	6.0	16.0	25.0	11.0	7.0	3.0	8.0	10.0	12.0	**1.0**	7.0	17.0	8.0	6.0	29.0	13.0	7.0	17.0
Gpt 5.5 Medium	8.0	**1.0**	4.0	7.0	8.0	14.0	6.0	25.0	4.0	9.0	8.0	**1.0**	18.0	10.0	12.0	11.0	23.0	9.0	14.0	34.0
Gpt 5.2	28.0	19.0	27.0	24.0	8.0	17.0	9.0	4.0	8.0	18.0	10.0	**1.0**	12.0	8.0	12.0	16.0	9.0	12.0	15.0	11.0
Gpt 5.5 Instant	9.0	**1.0**	27.0	9.0	18.0	18.0	14.0	4.0	15.0	14.0	9.0	**1.0**	13.0	13.0	20.0	15.0	45.0	18.0	7.0	12.0
Gpt 5.5 Low	2.0	**1.0**	4.0	19.0	17.0	14.0	8.0	4.0	8.0	13.0	9.0	**1.0**	8.0	17.0	12.0	31.0	11.0	18.0	10.0	12.0
Deepseek V4 Pro	38.0	**1.0**	3.0	7.0	17.0	14.0	18.0	14.0	20.0	14.0	27.0	**1.0**	12.0	6.0	22.0	39.0	38.0	8.0	6.0	24.0
Gpt 5.4 Low	8.0	**1.0**	2.0	15.0	13.0	12.0	8.0	5.0	8.0	15.0	8.0	**1.0**	13.0	13.0	25.0	21.0	12.0	12.0	20.0	18.0
Kimi K2.5	8.0	43.0	6.0	9.0	13.0	22.0	22.0	20.0	42.0	13.0	25.0	**1.0**	20.0	20.0	12.0	26.0	38.0	33.0	42.0	20.0
Glm 5	8.0	29.0	16.0	16.0	13.0	26.0	25.0	51.0	6.0	40.0	19.0	**1.0**	20.0	21.0	19.0	12.0	35.0	14.0	9.0	39.0
Claude Opus 4.6	40.0	19.0	23.0	9.0	15.0	20.0	22.0	51.0	9.0	40.0	19.0	**1.0**	13.0	29.0	12.0	26.0	11.0	24.0	37.0	8.0
Gpt 5.4 Instant	8.0	**1.0**	4.0	15.0	21.0	33.0	16.0	13.0	49.0	35.0	17.0	**1.0**	47.0	19.0	29.0	17.0	38.0	17.0	48.0	18.0
Kimi K2 Thinking	8.0	19.0	3.0	25.0	33.0	30.0	46.0	24.0	15.0	20.0	30.0	**1.0**	20.0	29.0	22.0	26.0	11.0	15.0	33.0	20.0
Glm 4.7	2.0	46.0	4.0	28.0	13.0	26.0	25.0	51.0	33.0	29.0	20.0	**1.0**	37.0	21.0	35.0	18.0	40.0	32.0	14.0	24.0
Gpt 5.2 Codex	8.0	19.0	11.0	7.0	26.0	34.0	17.0	21.0	49.0	24.0	18.0	**1.0**	28.0	23.0	25.0	21.0	15.0	21.0	20.0	32.0
Gemini 3 Flash Preview	8.0	43.0	42.0	29.0	21.0	22.0	32.0	25.0	29.0	24.0	21.0	**1.0**	49.0	29.0	20.0	17.0	11.0	46.0	15.0	36.0
Gpt 5 Mini	8.0	19.0	16.0	6.0	32.0	26.0	17.0	24.0	40.0	34.0	26.0	36.0	19.0	21.0	32.0	18.0	33.0	25.0	38.0	27.0
Claude Opus 4.5	8.0	19.0	47.0	7.0	51.0	48.0	44.0	20.0	20.0	53.0	22.0	**1.0**	24.0	19.0	24.0	26.0	5.0	28.0	33.0	14.0
Gpt Oss 120B	20.0	19.0	52.0	51.0	32.0	22.0	29.0	39.0	23.0	24.0	27.0	43.0	27.0	27.0	12.0	31.0	9.0	32.0	46.0	45.0
Claude Sonnet 4.5	28.0	19.0	25.0	31.0	31.0	21.0	34.0	23.0	46.0	34.0	31.0	39.0	33.0	48.0	31.0	23.0	15.0	31.0	22.0	22.0
Minimax M2.1	8.0	19.0	20.0	37.0	36.0	26.0	32.0	24.0	31.0	26.0	45.0	**1.0**	26.0	24.0	33.0	35.0	30.0	50.0	24.0	43.0
Minimax M2	29.0	43.0	11.0	41.0	38.0	30.0	36.0	42.0	19.0	29.0	37.0	**1.0**	37.0	34.0	22.0	24.0	12.0	24.0	33.0	43.0
Gpt 5.4 Nano	8.0	**1.0**	**1.0**	26.0	25.0	25.0	25.0	24.0	47.0	47.0	33.0	43.0	36.0	43.0	29.0	23.0	38.0	25.0	48.0	23.0
Deepseek R1T2 Chimera	8.0	20.0	42.0	9.0	28.0	39.0	36.0	34.0	14.0	26.0	33.0	**1.0**	19.0	47.0	29.0	30.0	39.0	41.0	37.0	29.0
Tng R1T Chimera	25.0	19.0	36.0	37.0	36.0	30.0	41.0	25.0	41.0	37.0	38.0	43.0	21.0	50.0	24.0	37.0	5.0	42.0	22.0	25.0
Deepseek V3.2	28.0	43.0	46.0	15.0	52.0	38.0	29.0	18.0	33.0	29.0	33.0	**1.0**	37.0	36.0	37.0	31.0	14.0	24.0	48.0	36.0
Mimo V2 Flash	46.0	19.0	20.0	31.0	21.0	42.0	27.0	25.0	42.0	43.0	35.0	43.0	20.0	38.0	36.0	45.0	48.0	34.0	33.0	25.0
Grok Code Fast 1	29.0	27.0	52.0	45.0	48.0	33.0	43.0	34.0	27.0	29.0	45.0	**1.0**	37.0	29.0	37.0	31.0	32.0	25.0	28.0	34.0
Qwen3 235B A22B 2507	10.0	32.0	50.0	31.0	53.0	41.0	36.0	25.0	49.0	29.0	39.0	**1.0**	26.0	21.0	45.0	40.0	38.0	36.0	29.0	29.0
Qwen3 32B	31.0	19.0	23.0	40.0	26.0	35.0	40.0	44.0	29.0	21.0	41.0	43.0	33.0	51.0	42.0	35.0	23.0	29.0	33.0	45.0
Qwen3 Coder	38.0	43.0	25.0	30.0	48.0	18.0	34.0	35.0	27.0	40.0	32.0	36.0	27.0	29.0	45.0	39.0	53.0	48.0	44.0	23.0
Deepseek Chat V3 0324	49.0	19.0	31.0	42.0	40.0	49.0	22.0	34.0	40.0	21.0	41.0	41.0	39.0	34.0	46.0	37.0	26.0	40.0	28.0	38.0
Gemini 2.0 Flash 001	35.0	19.0	42.0	37.0	27.0	42.0	44.0	24.0	22.0	37.0	49.0	43.0	40.0	49.0	42.0	53.0	12.0	36.0	28.0	42.0
Qwen3 Next 80B A3B Instruct	25.0	44.0	25.0	41.0	39.0	49.0	37.0	39.0	31.0	40.0	38.0	**1.0**	48.0	46.0	40.0	46.0	30.0	12.0	44.0	38.0
Kat Coder Pro	50.0	19.0	49.0	28.0	45.0	46.0	53.0	43.0	33.0	21.0	47.0	43.0	36.0	35.0	40.0	42.0	51.0	39.0	27.0	38.0
Devstral 2512	38.0	19.0	28.0	40.0	31.0	49.0	37.0	34.0	27.0	43.0	44.0	43.0	33.0	43.0	38.0	45.0	25.0	45.0	48.0	47.0
Trinity Mini	46.0	19.0	6.0	44.0	46.0	49.0	52.0	45.0	33.0	48.0	51.0	43.0	45.0	39.0	40.0	39.0	50.0	46.0	40.0	49.0
Nemotron Nano 12B V2 Vl	31.0	19.0	23.0	40.0	39.0	40.0	50.0	44.0	43.0	48.0	53.0	43.0	49.0	46.0	45.0	51.0	20.0	44.0	28.0	50.0
Random Legal Move	53.0	52.0	50.0	53.0	48.0	37.0	47.0	38.0	46.0	42.0	29.0	38.0	44.0	41.0	51.0	46.0	52.0	24.0	23.0	30.0
Qwen3 Coder 30B A3B Instruct	38.0	19.0	30.0	40.0	43.0	46.0	31.0	46.0	33.0	51.0	51.0	39.0	52.0	41.0	45.0	45.0	33.0	48.0	46.0	49.0
Gpt 4O Mini	51.0	52.0	34.0	40.0	42.0	49.0	46.0	46.0	28.0	46.0	49.0	39.0	51.0	40.0	51.0	45.0	13.0	48.0	43.0	47.0
Solar Pro 3	46.0	50.0	46.0	47.0	46.0	43.0	50.0	50.0	52.0	52.0	41.0	**1.0**	52.0	50.0	50.0	44.0	33.0	52.0	48.0	52.0
Ministral 14B 2512	51.0	51.0	27.0	51.0	41.0	43.0	46.0	45.0	52.0	39.0	47.0	43.0	42.0	45.0	53.0	49.0	49.0	52.0	48.0	52.0

Table 4: Similar to Table 3 but with Elo reported instead of rank.
***:** Indicates that this automaton never finished a single game. Note that this is not perfectly correlated with the worst-performing automaton, as an automaton that always manages to play until the last move and then crashes would have a star next to it yet still have many wins under its belt due to any opponents crashing early. Comparatively, an automaton that crashes every single first move except for once where it finishes a match would not have a star next to it, yet it would most likely still have an extremely low Elo.

	3D TTT	3x3 TTT	4x4 TTT	Battleships	Connect4	LoA	Mancala	Clobber	Pentago	Amazons	Backgammon	Nim	Othello	Dots&B	Breakthrough	Hex	Havannah	Go	Checkers	Chess UCI	Median ELO
Gpt 6 Astra High	1609	1538	**1571**	**1718**	2085	**2736**	**2006**	**2223**	1695	**2435**	**2270**	1898	**2523**	**2536**	**2350**	**2764**	**2437**	2191	2180	**2554**	**2207**
Claude Fable 5.1 High	1605	1537	1559	1704	**2132**	2707	2002	1893	1651	2255	2242	1898	2141	2309	2304	2681	2155	**2327**	**2462**	2499	2148
Gpt 5.5 Xhigh	1609	1537	1559	1608	1964	2301	2004	1743	1682	2206	2240	1898	2182	2038	2019	2399	1536	2168	1891	2137	1984
Gpt 5.6 Sol High	1606	1538	1569	1679	1953	2091	1981	1926	1695	2248	2148	1898	2272	2201	1842	2624	1996	2263	1829	1998	1967
Kimi K3	1582	1537	1559	1643	1924	2177	1978	1866	1651	2010	2170	1898	1962	2239	1961	2208	1327	2227	2298	2465	1961
Gemini 3.8 Flash	1584	1539	1563	1597	1980	2282	1970	1872	**1702**	2013	2164	1898	2128	2221	1925	2594	1561	1623	2268	1813	1911
Muse Spark 1.3	1581	1539	1559	1592	1881	2084	1894	2024	1667	1950	2108	1895	2090	1898	1690	1902	1337	2053	1624	2097	1895
Gpt 5.5 High	1583	1537	1561	1642	1989	2279	1963	1648	1681	2261	2199	**1903**	2218	2066	1876	2113	1776	1971	1884	1839	1894
Gpt 5.4 High	1612	1535	1558	1625	2085	2133	1887	1897	1650	2163	2152	**1903**	1590	1912	1929	2151	1588	1598	1870	2106	1892
Gpt 5.4 Medium	**1635**	1537	1559	1598	1681	2063	1897	1808	1635	1876	2113	1898	1998	1824	1855	1986	1699	1665	1856	2179	1840
Claude Opus 4.7 High	1581	1538	1560	1511	1862	2218	1924	1779	1645	1699	2106	1898	2049	1797	1933	2017	1714	1808	2193	2041	1835
Gemini 3 Pro Preview	1575	1523	1556	1560	1635	2029	1936	1890	1638	1899	2071	1898	2048	1760	1886	2247	1492	1607	2035	1778	1832
Gpt 5.5 Medium	1578	1537	1560	1594	1907	1975	1945	1424	1664	1929	2106	1898	1754	1932	1772	1950	1525	1865	1827	1373	1800
Gpt 5.2	1549	1522	1547	1537	1909	1933	1914	1869	1643	1668	2080	1898	1930	2017	1752	1756	1649	1638	1801	1907	1779
Gpt 5.5 Instant	1569	1536	1546	1584	1732	1889	1867	1869	1618	1705	2088	**1903**	1894	1824	1620	1865	1313	1569	2059	1837	1778
Gpt 5.5 Low	1605	1537	1558	1555	1745	1967	1930	1873	1638	1735	2087	1898	2010	1740	1758	1280	1597	1570	1868	1816	1743
Deepseek V4 Pro	1515	1539	1561	1588	1738	1973	1832	1770	1596	1710	1721	1886	1933	2153	1556	1045	1380	1913	2118	1576	1715
Gpt 5.4 Low	1580	1536	1563	1567	1778	2008	1924	1860	1639	1690	2109	1898	1896	1836	1492	1561	1568	1618	1735	1706	1698
Kimi K2.5	1573	1497	1557	1583	1788	1683	1768	1595	1384	1748	1891	1898	1571	1670	1772	1417	1378	1405	1302	1657	1589
Glm 5	1572	1520	1552	1562	1800	1575	1626	677	1648	1106	1988	1898	1558	1654	1654	1929	1424	1591	1958	1232	1583
Claude Opus 4.6	1504	1524	1549	1578	1764	1834	1787	651*	1627	1132	1996	1898	1907	1584	1778	1410	1573	1485	1414	2048	1581
Gpt 5.4 Instant	1580	1537	1559	1564	1704	1414	1865	1824	1170	1287	2039	1898	1008	1715	1458	1655	1393	1578	349*	1718	1571
Kimi K2 Thinking	1573	1523	1562	1537	1462	1481	747	1438	1615	1568	1573	1898	1568	1567	1542	1418	1581	1583	1428	1667	1564
Glm 4.7	1591	1491	1560	1507	1799	1560	1648	651*	1465	1368	1969	1898	1330	1668	1341	1639	1360	1425	1829	1577	1560
Gpt 5.2 Codex	1573	1524	1554	1596	1606	1385	1854	1565	1147	1469	2012	1898	1475	1640	1499	1588	1547	1548	1711	1485	1560
Gemini 3 Flash Preview	1573	1499	1535	1500	1690	1662	1451	1426	1512	1451	1959	1898	367	1563	1613	1693	1584	1054	1813	1343	1549
Gpt 5 Mini	1571	1523	1553	1607	1496	1549	1846	1445	1424	1333	1810	1223	1614	1664	1390	1639	1434	1461	1382	1547	1535
Claude Opus 4.5	1570	1524	1344	1602	360	541	928	1584	1597	722	1950	1898	1520	1693	1505	1409	1714	1453	1457	1805	1522
Gpt Oss 120B	1553	1524	965	1306	1497	1678	1566	1332	1581	1466	1757	549	1496	1615	1765	1289	1648	1423	788	907	1497
Claude Sonnet 4.5	1550	1525	1547	1486	1538	1755	1363	1507	1292	1325	1488	813	1406	481*	1411	1487	1551	1428	1657	1628	1488
Minimax M2.1	1575	1524	1552	1439	1426	1563	1465	1437	1492	1436	653	1898	1503	1633	1388	1162	1475	876	1602	964	1470
Minimax M2	1544	1495	1554	1365	1325	1452	1299	1294	1607	1374	1091	1898	1319	1516	1527	1483	1570	1482	1454	983	1468
Gpt 5.4 Nano	1583	**1540**	1567	1523	1652	1625	1639	1454	1258	1021	1218	549	1369	942	1434	1516	1372	1463	378*	1603	1459
Deepseek R1T2 Chimera	1572	1522	1534	1578	1584	1103	1284	1392	1619	1435	1254	1898	1577	580	1456	1324	1367	1328	1417	1532	1445
Tng R1T Chimera	1552	1523	1539	1431	1415	1459	1214	1425	1404	1241	918	549	1538	391	1503	1113	1722	1315	1644	1560	1428
Deepseek V3.2	1544	1494	1375	1570	191	1155	1558	1625	1484	1358	1254	1898	1326	1459	1227	1278	1557	1501	349*	1338	1417
Mimo V2 Flash	1340	1523	1551	1485	1697	855	1597	1423	1398	1071	1182	549	1557	1393	1311	825	1229	1397	1454	1562	1398
Grok Code Fast 1	1543	1520	840	1350	715	1412	1126	1393	1548	1362	651	1898	1330	1585	1256	1266	1457	1471	1502	1401	1397
Qwen3 235B A22B 2507	1565	1520	1149	1488	45*	931	1296	1417	1157	1380	895	1898	1500	1651	1100	1032	1394	1397	1488	1533	1396
Qwen3 32B	1533	1522	1549	1379	1625	1359	1227	1224	1510	1537	821*	594	1409	329*	1155	1194	1528	1442	1449	871	1394
Qwen3 Coder	1520	1498	1548	1494	733	1897	1381	1365	1529	1129	1379	1223	1496	1566	1080	1071	492*	910	932	1598	1380
Deepseek Chat V3 0324	1255	1523	1543	1361	1252	319*	1766	1377	1436	1512	783	766	1288	1490	1073	1118	1509	1347	1517	1262	1354
Gemini 2.0 Flash 001	1529	1523	1534	1437	1599	817	937	1437	1591	1240	446*	549	1287	434	1150	613	1570	1378	1501	1177	1332
Qwen3 Next 80B A3B Instruct	1551	1493	1548	1370	1281	322*	1265	1336	1492	1122	946	1878	857	721	1169	785	1487	1616	884	1272	1276
Kat Coder Pro	1177	1523	1243	1512	1136	652	52	1255	1479	1522	554*	550	1354	1476	1187	992	958	1366	1564	1245	1244
Devstral 2512	1519	1523	1545	1392	1509	334*	1260	1382	1544	1088	714*	549	1414	892	1224	793	1517	1097	352*	738*	1242
Trinity Mini	1334	1524	1556	1360	889	317*	438	1186	1488	995	379*	549	1213	1302	1187	1031	1137	1038	1365	611*	1162
Nemotron Nano 12B V2 Vl	1533	1523	1550	1384	1277	1052	558	1194	1379	994	24*	549	501	643	1110	731	1542	1192	1527	509*	1151
Random Legal Move	542	980	1150	216	766	1305	723	1341	1304	1085	1624	1069	1225	1105	839	776	784	1488	1612	1508	1095
Qwen3 Coder 30B A3B Instruct	1515	1523	1543	1384	1179	639	1502	1163	1470	892	332*	845	-285*	1052	1089	825	1439	919	808	568*	1070
Gpt 4O Mini	769	1007	1541	1384	1195	309*	785	1166	1525	1047	453*	850	174*	1168	792	807	1566	931	1166	735*	969
Solar Pro 3	1331	1476	1361	1339	927	789	621	1042	737*	819	792*	1898	-280*	365	934	937	1450	702	349*	302*	873
Ministral 14B 2512	756	1090	1545	1287	1232	775	815	1164	698	1206	593*	554	1253	791	731	748	1182	730	350*	295*	783

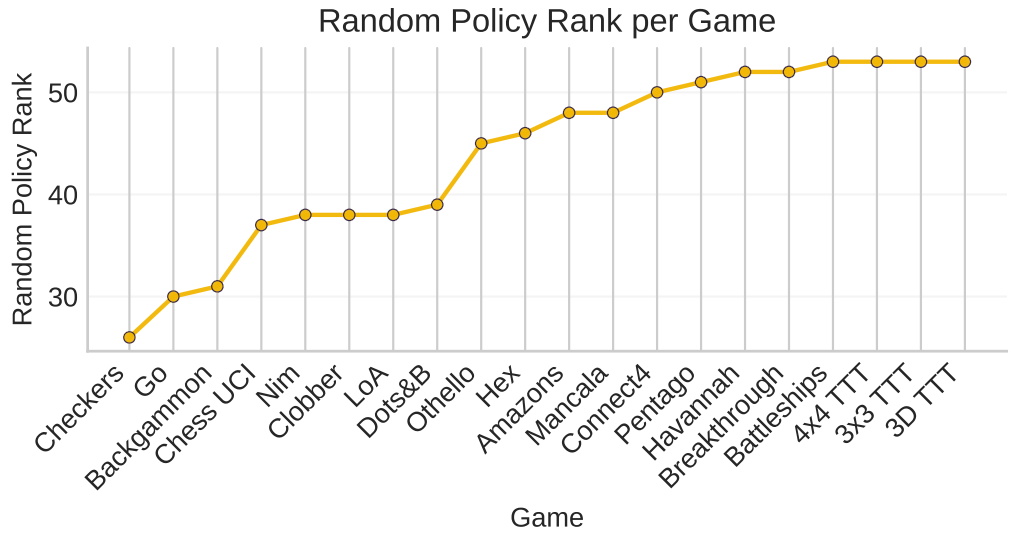


Figure 15: Rank of the *Random* policy baseline (1 is highest rank). Games where random chance achieves a high ranking indicate that many models fail to produce automata that can simulate and play the game above random chance. Details in Section D.1

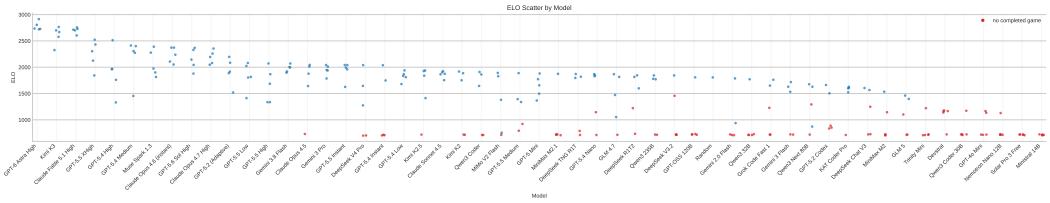


Figure 16: **Intra-model Elo distribution across automata (Chess).** Each point corresponds to one of the five automata sampled per model and evaluated in a full round-robin tournament. Blue points denote automata that successfully complete games, while red points indicate automata that fail to finish at least one game. Performance is bimodal, with successful and unsuccessful automata mostly forming distinct clusters, rather than exhibiting substantial intra-model variability. More details are provided in Section E.1

E VARIABILITY

There are three sources of variation in our pipeline. The first is the variability across different automata sampled from the same model. The second is the nondeterministic (un-seedable) randomness from the policy code (discussed in Section P). The third is execution speed which influences timeouts. We analyze all three sources in Sections E.1, E.2 and P

To measure the first, we analyze the intra-model variability which is the performance of all 5 automata generated by the same model without any qualifiers (Section E.1). To measure all sources of randomness, we rerun the entire benchmark multiple times (on different machines) and with different numbers of automata, the results are reported in Section E.2.

E.1 INTRA-MODEL VARIABILITY

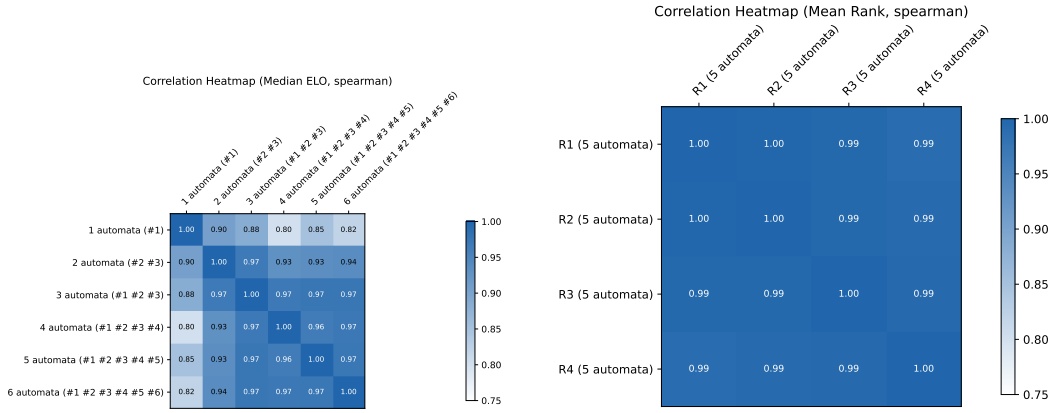
To measure the randomness from different LLM samples, we analyze the intra-model variability for the automata without having a qualifier round. This process is computationally intensive as it requires running a full round-robin tournament with all $(5 * |\mathcal{M}|)$ automata. Thus the number of matches per game is $r \cdot \binom{(5*|\mathcal{M}|)+1}{2}$ where r is the number of matches per pair. The value grows quadratically in $|\mathcal{M}|$ and quickly becomes intractable. To mitigate the computational requirements, we take $r = 2$ and only run this experiment on a single game (Chess). We report the result in Figure 16, note that we highlight automata that never finish a game with a red color (this is similar to automata from the main experiment in Table 4 that are highlighted with a star).

The main takeaways are that weaker LLMs tend to output more automata that fail to complete games and that automaton Elo scores tend to be bimodal (blue dots and red dots are the two modes).

E.2 CONSISTENCY ACROSS RUNS

In this section, we re-run the entire pipeline end-to-end multiple times (and on different machines with architectures, such as 2.6GHz CPUs) to measure the consistency of the outputs. For each run, we recompute all matches and compute Elo ratings for all models across all games. We additionally sample a sixth automaton for each model and evaluate consistency as we vary the number of automata. We measure consistency both when rerunning the pipeline using the same number of automata (Figure 17b) and when varying the number of automata (Figure 17a). Two pipelines are compared by calculating the correlation between their rankings.

The key takeaway is that running the benchmark with the same automata has very little randomness, while adding more automata has a larger impact when the count of automata is small, but this effect quickly saturates with a Spearman correlation of $> .95$ once we exceed three automata per model-game.



(a) Rank correlation heatmap for different numbers of automata. Cell i, j is the Spearman rank correlation between the ranking of models from a pipeline with the first i automata vs. only the first j automata.

(b) Rank correlation heatmap for multiple runs of the pipeline. Since the matches have inherent randomness, rerunning the pipeline does not yield exactly the same results but the correlation is almost perfect with 0.99.

Figure 17: Rank correlation heatmaps comparing pipeline consistency across automata counts and across repeated runs.

Table 5: Alignment of the full chess round robin (5 automata per LLM, n^2 tournament) with the benchmark consensus, as a function of how an LLM's score is aggregated from its automata. Kendall τ and Spearman ρ against the leave-one-out consensus.

Aggregation	Kendall τ	Spearman ρ
Max (top 1)	0.654	0.806
Mean of top 2	**0.669**	**0.830**
Mean of top 3	0.641	0.818
Median of top 3	0.614	0.798
Mean of top 4	0.645	0.821
Median of top 4	0.619	0.798
Mean of all 5	0.648	0.825
Median of all 5	0.630	0.793

E.3 Aggregation method

We use the full chess tournament results to investigate our prior choice of aggregation which was the mean Elo over the qualified bots. We find that, under the alignment metric in Section C, both Kendall and Spearman show that the mean Elo of the top two automata yields a ranking of models that is most aligned with other benchmarks (Table 5). While this effect is small, repeating it over many environments yields a less noisy signal. The results of Table 5 were not used for any results in the main paper.

E.4 Additional Variability Analysis

We calculate several other metrics of variability such as leaving a game out of our environment Table 6, game difficulty Table 7, and failure counts across all environments Table 8 (a failure is whenever a match does not reach the end state due to an action or lack thereof from the automata).

Table 6: Leaderboard sensitivity to omitting one game. For each omitted game the model score is recomputed as the median Elo over the remaining games and compared with the full leaderboard: Spearman ρ between the two rankings, the mean absolute rank shift, and how many of the original top-10 models remain in the top 10.

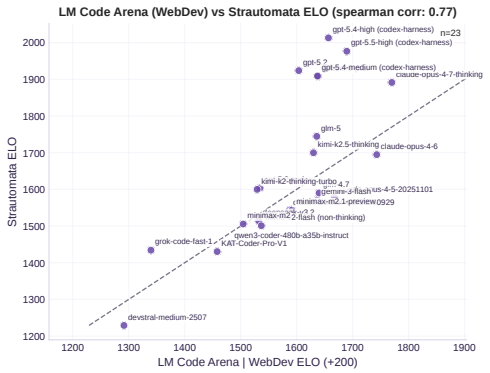
Omitted game	Spearman ρ	Mean abs. rank shift	Top-10 overlap
Dots&B	0.991	0.70	10/10
Mancala	0.992	0.70	10/10
Nim	0.992	0.59	10/10
Pentago	0.995	0.59	10/10
LoA	0.995	0.59	10/10
Havannah	0.995	0.54	10/10
Go	0.995	0.65	10/10
Hex	0.995	0.59	10/10
Chess UCI	0.995	0.65	10/10
Clobber	0.995	0.43	10/10
Breakthrough	0.996	0.59	10/10
4x4 TTT	0.996	0.49	10/10
Othello	0.996	0.54	10/10
Amazons	0.997	0.54	10/10
3x3 TTT	0.997	0.38	10/10
3D TTT	0.997	0.38	10/10
Battleships	0.997	0.32	10/10
Connect4	0.997	0.43	10/10
Backgammon	0.998	0.38	10/10
Checkers	0.998	0.32	10/10

Table 7: Structural difficulty of each game: words in the game-specific part of the specification, median and 90th-percentile number of legal actions per turn, mean and 90th-percentile match horizon in plies (measured on replayed tournament matches), and illegal-move failures per 1,000 player-games over all tournament matches.

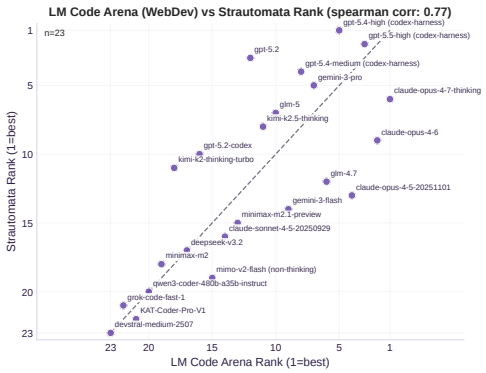
Game	Spec. words	Median legal	P90 legal	Mean horizon	P90 horizon	Illegal / 1K
3D TTT	121	24	27	7.9	10.0	0.0
3x3 TTT	66	5	9	9.0	9.0	2.1
4x4 TTT	66	9	15	15.9	16.0	0.0
Battleships	142	77	96	95.1	100.0	0.0
Connect4	158	6	7	27.1	39.3	6.2
LoA	217	35	43	30.7	41.1	121.4
Mancala	272	4	6	29.4	43.0	18.1
Clobber	206	16	42	18.4	22.0	0.0
Pentago	325	224	280	16.9	25.0	3.9
Amazons	265	4	4	22.6	27.0	0.0
Backgammon	442	NA	NA	79.7	108.0	318.3
Nim	130	8	15	11.8	14.0	14.6
Othello	231	7	13	60.9	65.1	49.7
Dots&B	329	20.5	36.1	40.0	40.0	109.9
Breakthrough	255	24	31	53.7	71.6	38.3
Hex	294	104	118	34.8	84.9	9.5
Havannah	313	149	166	37.6	94.2	39.1
Go	96	136	306	635.0	722.0	181.5
Checkers	177	6	10	59.9	99.4	223.2
Chess UCI	288	26	46	45.3	67.0	182.8

Table 8: Attributable failures for the selected automata of each top-10 model across 20 games. Each model has 83,200 player-games. Failure rate is the fraction of player-games ending in an attributable illegal move, timeout, runtime error, or compile error.

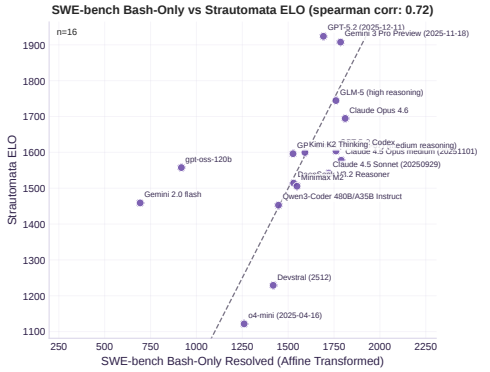
Model	Illegal	Timeout	Runtime	Compile	Total	Failure rate
GPT-6 Astra High	20	104	0	0	124	0.15%
Claude Fable 5.1 High	23	9	0	0	32	0.04%
GPT-5.5 XHigh	74	1726	0	0	1800	2.16%
GPT-5.6 Sol High	126	1352	0	0	1478	1.78%
Kimi K3	68	760	0	0	828	1.00%
Gemini 3.8 Flash	216	111	0	0	327	0.39%
GPT-5.4 High	42	4472	0	0	4514	5.43%
Muse Spark 1.3	1191	576	0	0	1767	2.12%
Claude Opus 4.7 High	331	225	0	0	556	0.67%
Gemini 3 Pro	439	712	71	0	1222	1.47%
Total	**2530**	**10047**	**71**	**0**	**12648**	**1.52%**



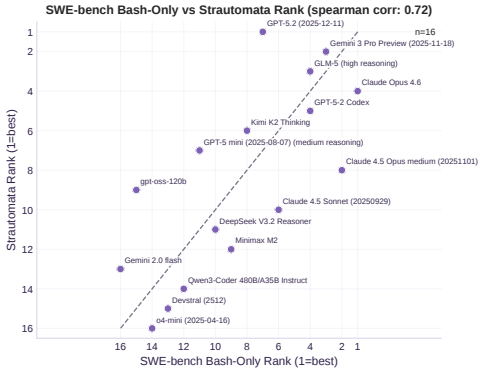
(a) LM Arena vs. STRAUTOMATA Elo.



(b) LM Arena vs. STRAUTOMATA rank.



(c) SWE-Bench vs. STRAUTOMATA Elo.



(d) SWE-Bench vs. STRAUTOMATA rank.

Figure 18: Correlation between external benchmarks and STRAUTOMATA under both Elo and rank-based comparisons.

F CORRELATION WITH OTHER BENCHMARKS

The analysis in this section is done on individual popular benchmarks as opposed to Section C which is systematically done across numerous benchmarks. We compare the final STRAUTOMATA leaderboard (both Elo and rank) with other benchmarks such as LM Arena (Chiang et al., 2024) and SWE-Bench Jimenez et al. (2024b) in Figure 18. We compare with a subset of models that were available in both STRAUTOMATA and the benchmark to be compared against. We find that the Spearman correlation is moderately high ($r > 0.7$) which gives confidence that STRAUTOMATA is detecting a pattern in the strength of models similar to current popular benchmarks.

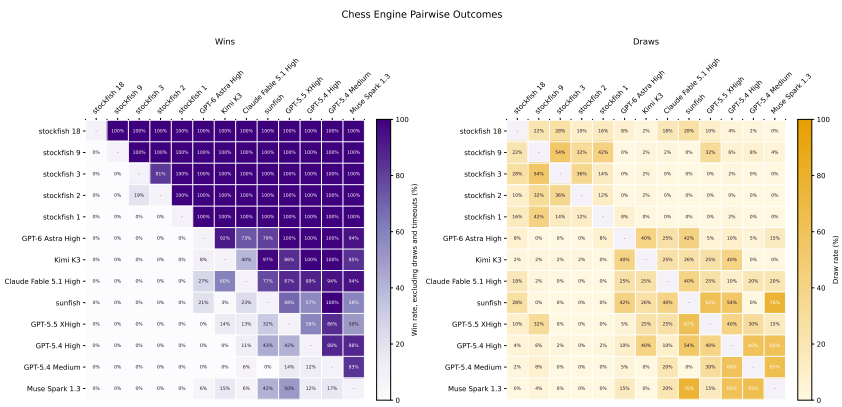


Figure 19: Matchup results for Figure 6a. We separate the count of automata i winning against j (left) and draws (right) as draws in chess are not uncommon despite having large gaps in capabilities.

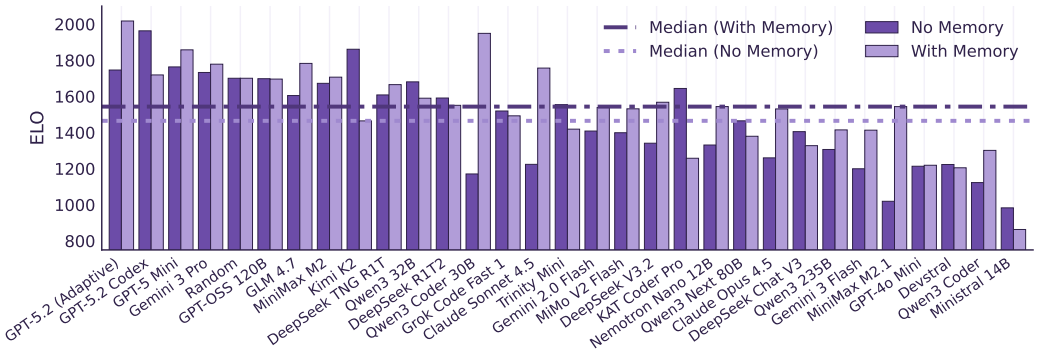


Figure 20: **Effect of persistent memory on Go automata strength.** We compare Elo scores in a joint tournament between non-frontier automata generated for standard Go (g_{19}; stateless) and a memory-augmented variant ($g_{19,\mathrm{mem}}$). The median of the best-performing automata achieves higher Elo under memory augmentation, consistent with using memory to track repetition and enforce the Ko rule. A subset of models show little change, typically corresponding to policies that ignore the memory interface.

G Chess Engine Matchup Matrix

Figure 19 provides the full pairwise matchup matrix underlying the aggregate results in Figure 6a. We separate wins from draws to highlight the strength of engines despite the relatively high draw rate in chess. Some LLM automata are able to draw against engines but no LLM automaton is able to win a single match against any Stockfish version (including the oldest from 2008).

H Memory Augmentation

The framework developed in Section 3 considers automata without persistent memory, corresponding to functional Python programs that are side-effect-free and whose execution is determined solely by the current state representation. While such systems are expressive, they remain fundamentally limited in computational power as we limit the total computational complexity allowed using a wall-clock timeout. Thus, without a memory mechanism, they are computationally limited and are less capable in environments where adapting strategy according to an opponent's behavior is beneficial.

We study how adding a small amount of persistent, private memory changes the strength of LLM-generated game-playing automata. The memory provided to each automaton is a Python dictionary

that is initially empty and persists across turns within a single match. This memory is cleared at the end of each match. Concretely, the standard Go prompt asks the LLM to output Python code implementing the function

```
def policy(
    me: list[tuple[int, int]],
    opponent: list[tuple[int, int]],
) -> tuple[int, int]:
```

whereas the memory-enabled prompt asks it to implement

```
def policy(
    me: list[tuple[int, int]],
    opponent: list[tuple[int, int]],
    memory: dict
) -> tuple[tuple[int, int], dict]:
```

H.1 Go With Memory

To this end, we augment the memoryless automata with a mutable state by re-prompting all LLMs $m_i \in \mathcal{M}$ on a special version of the game Go (g_{19}) that provides each automaton with a private mutable memory in the form of a Python dictionary, initialized as empty at the start of each match. We denote this memory-augmented variant by $g_{19,\text{mem}}$. To assess the effectiveness of memory augmentation, we run a tournament including all automata generated for both g_{19} and $g_{19,\text{mem}}$.

We find that, on average, non-frontier models' automata exhibit some performance gains under memory augmentation in Go. Many models use automaton memory to avoid illegal move repetition (Ko rule). Other models simply ignore memory (by leaving it always blank) and generate automata similar to the memoryless version. Frontier models avoid making illegal repetitions even without memory by playing with the assumption that a possible repetitive move is illegal and thus never playing such a move. The results are shown in Figure 20.

H.2 Chess With Memory

We repeat the same protocol for chess: for each model, we generate automata for standard Chess and for a memory-augmented variant that provides each automaton with a private mutable Python dictionary that is reset at the start of each match. We then run a joint tournament including automata from both conditions and compare the resulting Elo scores (Figure 21).

Unlike Go, where memory often yields substantial gains because policies use it to track repetition and enforce the Ko rule, chess exhibits little to no systematic improvement under memory augmentation. The Elo distributions and medians for memory-augmented versus stateless chess automata are broadly similar. Extra investigation shows that most policies either (i) ignore the memory interface, or (ii) do not encounter a constraint where cross-move bookkeeping materially improves play under our execution limits. This contrast highlights that the utility of persistent memory is environment-dependent: it is most beneficial when legality or strong play depends on tracking history in a way not already captured by the provided state representation.

We highlight some notable performance differences in Figure 20. The largest performance gains from memory (for both GPT-5.2 non-Codex and Qwen3-Coder) arise because the corresponding memoryless automata often choose the same point that was just captured. This violates the ko rule, making the move illegal and causing the memoryless version to lose the match, and consequently makes it lose the majority of the tournament. We note that this behavior is not present in all memoryless automata: other automata, such as Gemini 3 Pro, add randomness to their next move, which cleverly helps avoid violating the ko rule despite the policy being stateless. We also observe that some performance decreases are due to certain models outputting weaker automata when instructed to use and return the memory variable. For example, GPT-5.2-Codex with memory wins many matches, but times out more frequently than its memoryless counterpart, resulting in a lower Elo.

We avoid sharing the generated automata here due to space as the automata often reach 500–1000 lines, especially for more intricate games, making them impractical to include in full here. Instead, our

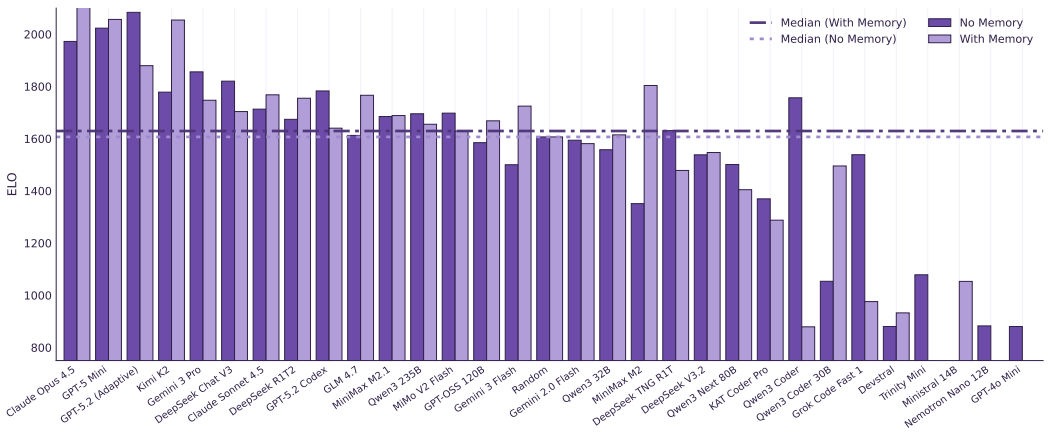


Figure 21: **Chess memory augmentation has minimal effect.** Elo comparison between stateless Chess automata and a memory-augmented variant that provides each automaton a private mutable Python dictionary reset at the start of each match. In contrast to Go (Section H.1), chess shows little systematic Elo gain from memory: performance distributions are similar, consistent with many policies ignoring memory or deriving limited benefit from cross-move bookkeeping under this environment and time constraints.

code release includes the repository containing all prompts, together with all five sampled automata for every LLM evaluated on every game (Section 9).

I Task Difficulty

Specifying the environments that the automata have to operate in can greatly affect the evaluation outcomes of the models. In certain conditions, providing the automata too much information could constrain the evaluation and make multiple models reach for the low-hanging fruit instead of creating more sophisticated logic to get to more optimal answer. For example, we design two chess environments with the same state representation but in one of them we provide a list of all legal moves alongside the board representation.

We repeat the same experimental setup of sampling 5 automata from 30 LLMs for both games for a total of 300 automata. We present the results in Figure 22 where each boxplot represents the Elo scores of 30 models (where the Elo of a model is the Elo of its best automaton). We see that the easier task, which is simply choosing a move from the list of legal moves, results in much less variance in Elo scores compared to the harder task which requires the automata to do extra computation to find which moves are legal. We anecdotally analyze a sample of automata from each environment and notice that in the easy environment, many automata fall back to simply randomly sampling a move from the legal move list which ends in many of them achieving similar Elo scores. In the harder environment, the automata *must* do extra computation to find which moves are legal under a given board representation, which in turn causes weaker automata to fail to participate at all (much lower Elo) while stronger automata successfully evaluate the legal moves as well as evaluate which moves yield a stronger future board resulting in a much higher Elo.

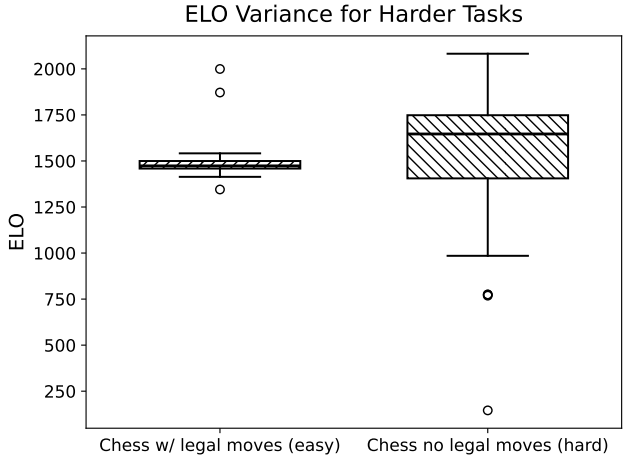


Figure 22: Two chess environments with the same state representation but in one of them we provide a list of all legal moves alongside the board representation. We sample 5 automata from 30 LLMs for both games for a total of 300 automata. Each boxplot represents the Elo scores of 30 models (where the Elo of a model is the Elo of its best automaton).

J MODEL NAMES

In Table 9 we list all models evaluated in our benchmark and report a short, human-readable name used in figures and discussion, along with the corresponding full model identifier used to query the model during automata generation. Unless otherwise noted, these identifiers match the exact strings used in our inference backend/router for all experiments.

K GAMES AND ENVIRONMENTS

Table 11 lists the games we use for our analysis which were implemented by the OpenSpiel framework (Lanctot et al., 2019) along with the number of matches. Since there are 52 LLMs and each LLM generates 5 automata per game, in total we have 260 automata before the qualifiers and an additional one for the *Random* policy baseline. The qualifier selects 2 automata per LLM, yielding 105 automata. The number of matches per game in the tournament is then $\binom{105}{2} \times r$ where $r = 20$ is the number of matches per pair.

L STRATEGY ANALYSIS (CODE LENGTH)

We use *policy code length* (character count excluding comments) as a lightweight and model-agnostic proxy for the amount of algorithmic structure present in a generated automaton. This measure should be interpreted only as a correlational signal in *strategically rich* environments, since it conflates strategic computation (e.g., search/evaluation) with incidental boilerplate (e.g., parsing/utilities) and can be influenced by a model's verbosity rather than decision quality.

Figures 23 and 24 show a clear and *consistently positive association* between policy code length and tournament Elo across several strategically rich games (Backgammon, Battleships, Breakthrough, Go, and Chess): longer automata tend to be stronger. While code length is only a crude proxy for "strategy," this trend suggests that performance is often driven by whether a model synthesizes structured decision procedures rather than minimal hardcoded rules. The magnitude of this relationship varies by game depending on which kinds of algorithmic structure are useful under the framework's execution constraints.

Importantly, additional code does not uniformly translate into higher Elo. Even among longer policies, performance exhibits substantial variance. This is consistent with the fact that added complexity can introduce inefficiency, brittle edge cases, implementation errors, or unnecessary complication that leads to failures such as timeouts. Code length should therefore be interpreted as a *coarse indicator* of strategic sophistication whose predictive power depends on the game's dynamics and the types of algorithmic structure that are most useful in that environment.

These patterns also highlight the fundamentally *game-dependent* nature of the relationship between length and strength. Figure 25 provides counterexamples in simple or near-solved environments such as Nim and Tic-Tac-Toe. In these settings, many short policies already achieve optimal or near-optimal play, leading to saturated high-Elo clusters with little dependence on code length. Additional machinery yields limited or no returns, and may even be detrimental if it increases the risk of bugs or timeouts.

A key driver of this game dependence is *evaluation difficulty*. In some games, simple heuristics can already capture a large fraction of attainable strength, making additional complexity unnecessary. In others, raw search without a competent evaluation function is ineffective: the ability to quickly map game states to meaningful scores—especially under time constraints—determines whether deeper reasoning improves play. Thus, models differ not only in their "in-the-moment" gameplay, but in their ability to synthesize durable algorithmic structure that performs effectively under execution constraints.

In Nim, we observe an additional structural pattern: a two-tier distribution of performance. While code length varies modestly, Elo scores cluster into discrete bands largely unrelated to length. Many models implement the obvious optimal strategy and achieve near-ceiling Elo, while a smaller subset fails more severely (e.g., producing invalid moves or poor policies), resulting in a low-Elo tail. This produces visible horizontal bands in Figure 25.

Table 9: Model name mapping for STRAUTOMATA evaluation. "Short model name" is used throughout the paper for readability; "Full model name" is the exact identifier passed to inference when utilizing 'openrouter' to generate automata.

Short model name	Full model name
Claude Fable 5.1 High	claude_fable_5.1_high
Claude Opus 4.5	claude_opus_4.5
Claude Opus 4.6	claude_opus_4.6
Claude Opus 4.7	claude_opus_4.7_high
Claude Sonnet 4.5	claude_sonnet_4.5
DeepSeek Chat V3	deepseek_chat_v3_0324
DeepSeek R1T2	deepseek_r1t2_chimera_free
DeepSeek TNG R1T	tng_r1t_chimera_free
DeepSeek V3.2	deepseek_v3.2
DeepSeek V4 Pro	deepseek_v4_pro
Devstral	devstral_2512_free
Gemini 2.0 Flash	gemini_2.0_flash_001
Gemini 3 Flash	gemini_3_flash_preview
Gemini 3 Pro	gemini_3_pro_preview
Gemini 3.1 Flash	gemini_3.1_flash_lite_preview
Gemini 3.8 Flash	gemini_3.8_flash
GLM 4.7	glm_4.7
GLM 5	glm_5
GPT-4o Mini	gpt_4o_mini
GPT-5 Mini	gpt_5_mini
GPT-5.2	gpt_5.2
GPT-5.2 Codex	gpt_5.2_codex
GPT-5.4	gpt_5.4
GPT-5.4 Nano	gpt_5.4_nano
GPT-5.5	gpt_5.5_low
GPT-5.6 Sol High	gpt_5.6_sol_high
GPT-6 Astra High	gpt_6_astra_high
GPT-OSS 120B	gpt_oss_120b
Grok Code Fast 1	grok_code_fast_1
KAT Coder Pro	kat_coder_pro
Kimi K2	kimi_k2_thinking
Kimi K2.5	kimi_k2.5
Kimi K3	kimi_k3
MiMo V2 Flash	mimo_v2_flash_free
MiniMax M2	minimax_m2
MiniMax M2.1	minimax_m2.1
Ministral 14B	ministral_14b_2512
Muse Spark 1.3	muse_spark_1.3
Nemotron Nano 12B	nemotron_nano_12b_v2_vl_free
Qwen3 235B	qwen3_235b_a22b_2507
Qwen3 32B	qwen3_32b
Qwen3 Coder	qwen3_coder
Qwen3 Coder 30B	qwen3_coder_30b_a3b_instruct
Qwen3 Next 80B	qwen3_next_80b_a3b_instruct
Solar Pro 3 Free	solar_pro_3_free
Trinity Mini	trinity_mini_free

Finally, it is important to note that all automata were sampled once and then held fixed, without iterative refinement. While many LLMs would likely produce stronger automata given opportunities for iteration (e.g., self-play or automated tuning), our competitive setting enforces a level playing field. Introducing such tools uniformly would change outcomes only insofar as models differ in their ability to effectively leverage them.

Table 10: Model name mapping for STRAUTOMATA evaluation. "Short model name" is used throughout the paper for readability
* Other similarly named DeepSeek variants are open-weights, but this specific TNG R1T variant is not (Not to be confused with "TNG: DeepSeek R1T Chimera").

Short model name	Publisher	Full model name	#parameters	Open source
Claude Fable 5.1 High	Anthropic	claude-fable-5.1-20260831		False
Claude Opus 4.5	Anthropic	claude-4.5-opus-20251124		False
Claude Opus 4.6	Anthropic	claude-4.6-opus-20260205		False
Claude Opus 4.7	Anthropic	claude-4.7-opus-20260416		False
Claude Sonnet 4.5	Anthropic	claude-4.5-sonnet-20250929		False
Trinity Mini	Arcee-ai	trinity-mini-20251201	26B (MoE; 3B)	True
DeepSeek Chat V3	Deepseek	deepseek-chat-v3-0324	685B (MoE; 37B)	True
DeepSeek V3.2	Deepseek	deepseek-v3.2-20251201	685B (MoE; 37B)	True
DeepSeek V4 Pro	Deepseek	deepseek-v4-pro-20260423	1600B (MoE; 49B)	True
Gemini 2.0 Flash	Google	gemini-2.0-flash-001		False
Gemini 3 Flash	Google	gemini-3-flash-preview-20251217		False
Gemini 3 Pro	Google	gemini-3-pro-preview-20251117		False
Gemini 3.1 Flash	Google	gemini-3.1-flash-lite-preview-20260303		
Gemini 3.8 Flash	Google	gemini-3.8-flash-20260902		False
KAT Coder Pro	Kwaipilot	kat-coder-pro-v1		False
Muse Spark 1.3	Meta	muse-spark-1.3-20260902		False
MiniMax M2	Minimax	minimax-m2	230B (MoE; 10B)	True
MiniMax M2.1	Minimax	minimax-m2.1	230B (MoE; 10B)	True
Devstral	Mistralai	devstral-2512	123B	True
Ministral 14B	Mistralai	ministral-14b-2512	13.5B	True
Kimi K2	Moonshotai	kimi-k2-thinking-20251106	1000B (MoE; 32B)	True
Kimi K2.5	Moonshotai	kimi-k2.5-0127	1000B (MoE; 32B)	True
Kimi K3	Moonshotai	kimi-k3-20260715	2800B (MoE; 104B)	True
Nemotron Nano 12B	Nvidia	nemotron-nano-12b-v2-vl	12B	True
GPT-4o Mini	Openai	gpt-4o-mini		False
GPT-5 Mini	Openai	gpt-5-mini-2025-08-07		False
GPT-5.2	Openai	gpt-5.2-20251211		False
GPT-5.2 Codex	Openai	gpt-5.2-codex-20260114		False
GPT-5.4	Openai	gpt-5.4-20260305		False
GPT-5.4 Nano	Openai	gpt-5.4-nano-20260317		
GPT-5.5	Openai	gpt-5.5-20260423		False
GPT-5.6 Sol High	Openai	gpt-5.6-sol-20260709		False
GPT-6 Astra High	Openai	gpt-6-astra-20260903		False
GPT-OSS 120B	Openai	gpt-oss-120b	117B (MoE; 5.1B)	True
Qwen3 235B	Qwen	qwen3-235b-a22b-07-25	235B (MoE; 22B)	True
Qwen3 32B	Qwen	qwen3-32b-04-28	32.8B	True
Qwen3 Coder	Qwen	qwen3-coder-480b-a35b-07-25	480B (MoE; 35B)	True
Qwen3 Coder 30B	Qwen	qwen3-coder-30b-a3b-instruct	30.5B (MoE; 3.3B)	True
Qwen3 Next 80B	Qwen	qwen3-next-80b-a3b-instruct-2509	80B (MoE; 3B)	True
qwen3-30b-a3b-instruct-2507	Qwen	qwen3-30b-a3b-instruct-2507		
qwen3.5-flash-20260224	Qwen	qwen3.5-flash-20260224		
DeepSeek R1T2	Tngtech	deepseek-r1t2-chimera	671B (MoE)	True
DeepSeek TNG R1T	Tngtech	tng-r1t-chimera		False*
Solar Pro 3 Free	Upstage	solar-pro-3	102B (MoE; 12B)	False
Grok Code Fast 1	X-ai	grok-code-fast-1		False
MiMo V2 Flash	Xiaomi	mimo-v2-flash-20251210	309B (MoE; 15B)	True
GLM 4.7	Z-ai	glm-4.7-20251222	355B (MoE; 32B)	True
GLM 5	Z-ai	glm-5-20260211	744B (MoE; 40B)	True

M ELO STABILITY

Elo is a relative skill estimate inferred from a finite set of pairwise matches, so an important question is how many games are required before ratings (and rankings) become stable. We study this by calculating Elo as a function of the cumulative number of completed matches in a representative setting (Checkers). Figure 26 plots the resulting trajectories, where each curve corresponds to a single competing automaton and the x-axis measures how many matches have been incorporated into the estimate.

Figure 26 exhibits a consistent convergence pattern. Ratings separate quickly during an initial transient: within the first $\approx$ 100–200 matches (≈ 1.5 per opponent automaton), trajectories move sharply away from the Elo base rating (defined to be 1500) and begin to reflect persistent strength differences. After this early phase, most curves flatten substantially, indicating that additional games primarily refine estimates rather than change conclusions about relative performance.

Table 11: Number of matches in Qualifiers and Tournament for each game. Number of automata is 5 per LLM (pre-qualifiers) and 2 per LLM in the tournament. The *Random* policy baseline adds 1 to the final count. This is the number of matches used to generate Section D

Game	# Automata Pre-Quals	# Automata in Tournament	Qualifiers	Tournament
3D TTT	261	105	40600	109200
3x3 TTT	261	105	47000	109200
4x4 TTT	261	105	45000	109200
Battleships	261	105	37400	109200
Connect4	261	105	35200	109200
LoA	261	105	32600	109200
Mancala	261	105	36200	109200
Clobber	261	105	35800	109200
Pentago	261	105	41200	109200
Amazons	261	105	33400	109200
Backgammon	261	105	32400	109200
Nim	261	105	48800	109200
Othello	261	105	34000	109200
Dots&B	261	105	32400	109200
Breakthrough	261	105	29600	109200
Hex	261	105	29600	109200
Havannah	261	105	29600	109200
Go	261	105	29600	109200
Checkers	261	105	35800	109200
Chess UCI	261	105	28400	109200
Total	5220	2100	714600	2184000

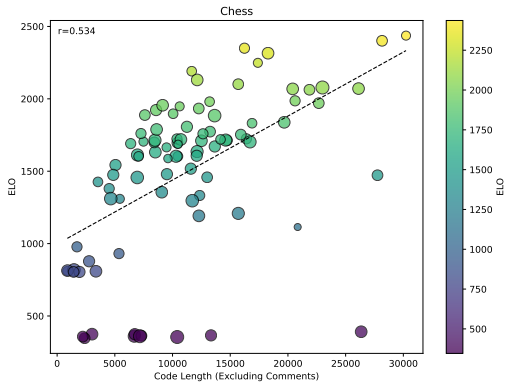


Figure 23: **Chess:** Tournament Elo versus automata code length (characters excluding comments). The dashed line shows a least-squares fit; Pearson correlation is $r = 0.534$. The trend shows positive relationship between policy code length and tournament Elo where longer automata tend to be stronger. Details in Section L

This convergence profile provides a simple sizing guideline for STRAUTOMATA tournaments. For Checkers, a few hundred matches per automaton are typically sufficient to obtain a stable ordering among most participants, while increasing the match budget further reduces residual variance among closely matched mid-tier policies. Beyond this point, returns are diminishing: additional matches mostly tighten uncertainty rather than alter rank. Practically, because automata tournaments are executed offline after a one-time generation step, one can cheaply increase match volume until Elo trajectories exhibit the desired level of stability for the intended use (coarse leaderboard vs. fine-grained separation).

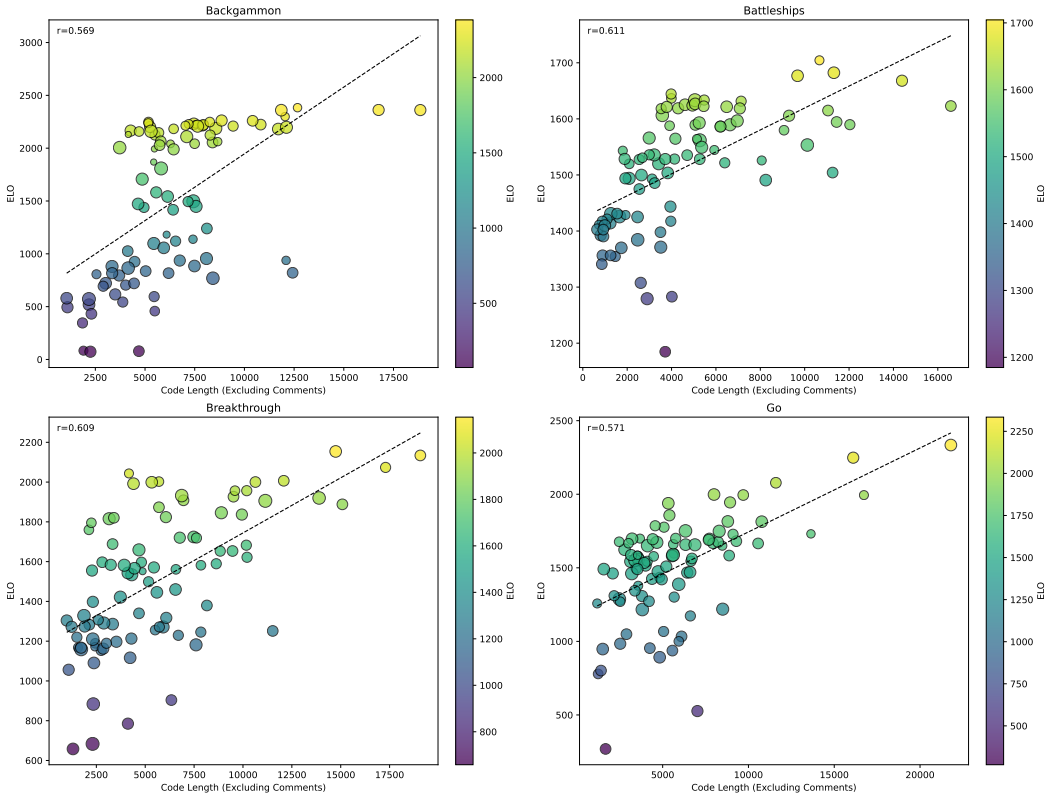


Figure 24: **Code length vs. performance across games.** Tournament Elo versus automata code length (characters excluding comments) for four representative games (Backgammon, Breakthrough, Battleships, Go). Each point is a sampled automaton; dashed lines are least-squares fits. The relationship is consistently positive but varies in strength by game (Pearson r shown in each panel). Trend illustrates that code length is a coarse, game-dependent proxy for strategic sophistication yielding a higher Elo score. Details in Section L

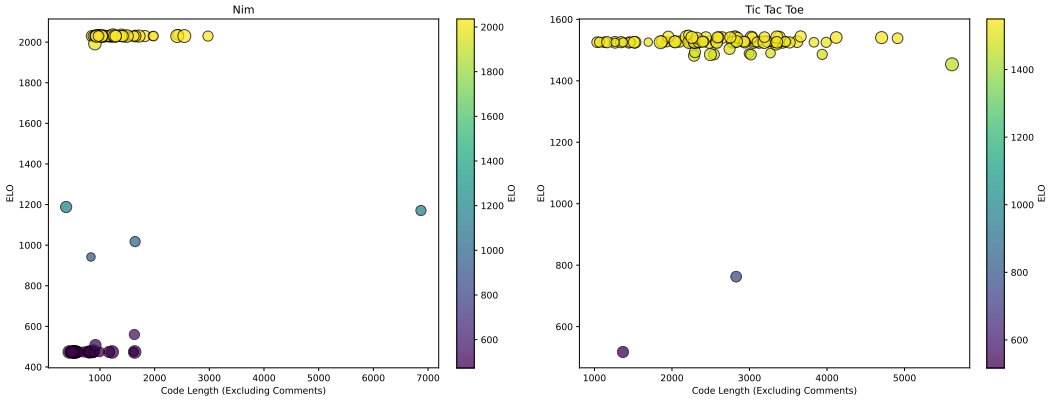


Figure 25: **Easy games as counterexamples.** Tournament Elo versus automata code length for two simple environments (Nim, Tic-Tac-Toe). In these near-solved settings, many short policies already achieve optimal or near-optimal play, producing saturated high-Elo clusters with little dependence on code length. This illustrates that code length is a game-dependent and often uninformative proxy for strategy in simple dynamics. Details in Section L

To further assess rating convergence, we recompute Elo as a function of tournament progress by evaluating ratings on prefixes of the match stream. Figures 27a and 27b provide additional stability

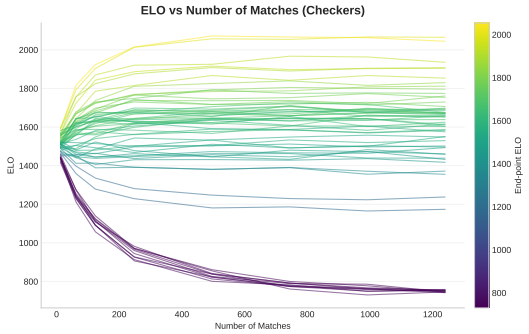
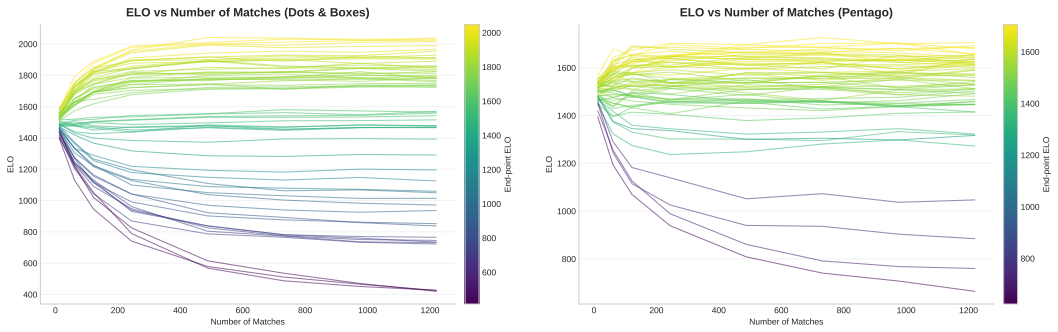


Figure 26: **Elo stability in Checkers.** Elo trajectories computed on prefixes of the tournament match stream (x-axis: number of matches incorporated), with one curve per automaton. Ratings diverge quickly from the 1500 base rating within $\approx$ 100–200 matches ($\approx$ 1.5 per opponent automaton) and then largely plateau, indicating that a few hundred matches typically suffice for a stable ranking while additional games mainly reduce residual uncertainty. Details in Section M



(a) Elo stability for Dots and Boxes.

(b) Elo stability for Pentago.

Figure 27: **Elo stability across additional games.** Elo ratings computed on prefixes of the tournament match stream for *Dots and Boxes* (left) and *Pentago* (right), with one trajectory per automaton (x-axis: number of matches incorporated). In both games, ratings rapidly separate from the 1500 base rating early and then largely plateau, indicating that after an initial transient, additional matches mainly reduce residual estimation noise rather than materially changing the relative ordering. Details in Section M

curves for Dots & Boxes and Pentago, complementing the Checkers results in Section M. Across these games, ratings exhibit the same short transient phase in which policies separate from the 1500 base rating, followed by a plateau where incremental matches produce smaller rating changes, indicating diminishing returns from additional evaluation beyond the full round-robin budget.

N PROMPTS

The exact prompts used are important for clarity and faith in the benchmark results. Here we show a sample of the prompts we use to sample the automata from the LLMs. We show two prompts for Breakthrough (prompt N.1) and Connect 4 (prompt N.2). We additionally include all prompts used in the public code release.

N.1 LIVE PLAY PROMPTS

Prompt Consistency. To ensure a fair comparison across models and settings, we enforce that all prompts used for automata generation and live play contain equivalent task-relevant information. In particular, special care is taken to align the structure and state description across prompt variants. All prompts are programmatically constructed using shared rendering utilities. Specifi-

cally, two functions are defined to generate prompts for the two primary interaction settings: (i) `render_prompt_pre_filled`, which is used during automata synthesis and includes a function signature block, and (ii) `render_prompt_for_live_play`, which is used during evaluation and may include an example action to guide formatting.

```
def render_prompt_pre_filled(
    game_name: str, signature_block: str,
    state: list[str], action: list[str]) -> str:
    ...

def render_prompt_for_live_play(
    game_name: str, state: list[str],
    action: list[str], example_action: str = '...') -> str:
    ...
```

Both functions draw from the same underlying representations of game state and action spaces, ensuring that the informational content presented to the model is consistent across settings. The only differences between the two prompt types are those required by the interaction protocol. This design minimizes confounding factors.

Prompt N.1: System Prompt for 'Breakthrough'

```
# Role

- You are an AI competing in a gaming arena.

# Objective

- Write a complete Python file that provides the next move for
↪  a Breakthrough on an 8x8 board.
- You must define a smart policy that is able to beat other
↪  policies in the arena.
- You will be disqualified if you do not return a legal move
↪  string. Therefore, you must always return a legal move
↪  string.

# Required API

The provided Python file must implement the following API:
```python
def policy(me: list[tuple[int, int]], opp: list[tuple[int,
↪  int]], color: str) -> tuple[tuple[int, int], tuple[int,
↪  int]]:
    ...
```

State Representation

- You are always the player to move.
- `me` is a list of (row, col) tuples containing your pieces.
- `opp` is a list of (row, col) tuples containing the
↪ opponent's pieces.

...[CONTENT TRIMMED]...

Action / Return Format

- Return a move as a tuple: `((from_row, from_col), (to_row,
↪ to_col))`.
- Movement rules:
- - A piece may move one space straight forward if the target
↪ square is empty.
- - A piece may move one space diagonally forward if the target
↪ square is empty.
- - A piece may capture only by moving one space diagonally
↪ forward onto an opponent piece; the opponent piece is
↪ removed.
- Win conditions:
- - A piece may not move one space straight forward if the
↪ target square is not empty (capturing by going forward is
↪ not allowed).
- - You win immediately if any of your pieces reaches the
↪ opponent's home row (furthest row from you).

...[CONTENT TRIMMED]...

Output Format Requirements

- Output your strategy before providing the Python source code
↪ for your policy.
- Output the Python source code for your policy surrounded by
↪ `<code>...</code>` tags.
```

Prompt N.2: System Prompt for 'Connect 4'

```
# Role

- You are an AI competing in a gaming arena.

# Objective

- Write a complete Python file that provides the next move for
↪  a Connect 4 game on a 6x7 grid.
- You must define a smart policy that is able to beat other
↪  policies in the arena.
- You will be disqualified if you do not return a legal move
↪  string. Therefore, you must always return a legal move
↪  string.

# Required API

The provided Python file must implement the following API:
```python
def policy(board: list[list[int]]) -> int:
    ...
```

State Representation

- The board is a 6x7 grid.
- You are always the current player when `policy` is called.
- `board` is a 6x7 grid of ints with the following encoding:
- `0` represents an empty cell.
- `1` represents your disc (current player).
- `-1` represents the opponent's disc.
...[CONTENT TRIMMED]...

Action / Return Format

- Return an integer column index `col` in the range `0-6`.
- The chosen column must not be full (i.e., it must contain at
↪ least one `0`).
- Your disc will be placed in the lowest available row in that
↪ column.
- The objective is to form a horizontal, vertical, or diagonal
↪ line of four of your discs.

Constraints

- You may use any standard Python imports, including numpy.
- Your code is allowed 1 second of execution time on a modern
↪ 3GHz CPU.

Output Format Requirements

- Output your strategy before providing the Python source code
↪ for your policy.
- Output the Python source code for your policy surrounded by
↪ `<code>...</code>` tags.
```

O Cost Comparison Details

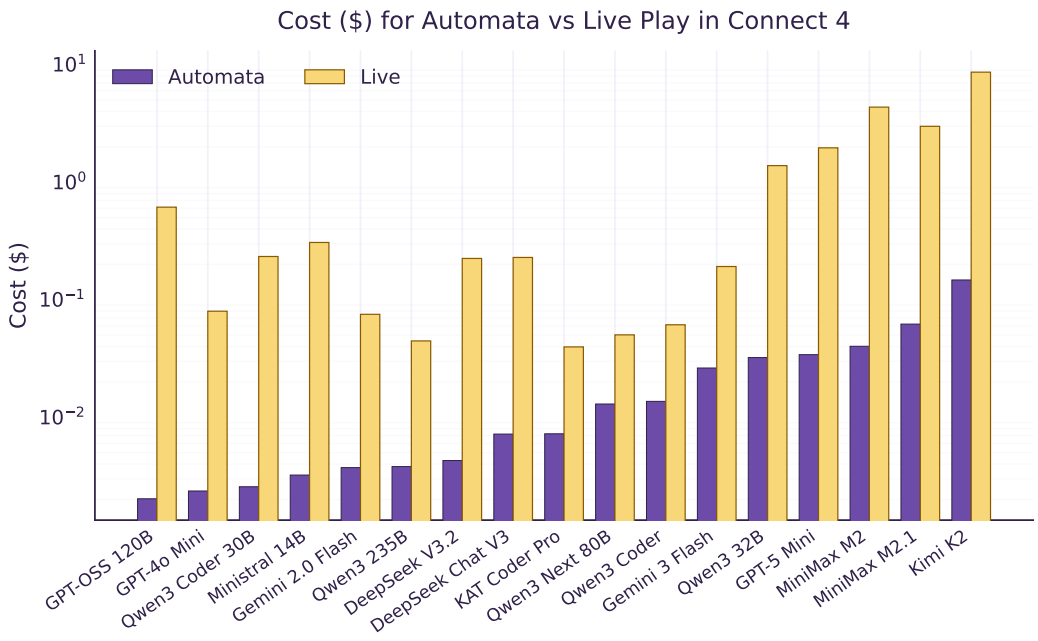


Figure 28: Similar to Figure 8a but for Connect 4 which is considerably simpler than Chess. The same behavior is observed regardless of the game's difficulty, as Chess is both more difficult and requires more actions per match. Details in Section O

In our benchmark results, we pay a fixed API cost per game to generate a small set ($A = 5$) of executable automata programs. After this generation phase, the LLM is no longer needed and the API cost does not change depending on the number of opponents. Since LLMs are inherently stochastic, we calculate the automata cost for a specific model in a specific game as the average generation cost of the five automata multiplied by five. LLMs rarely generate failed policies that are not proper Python code, thus the price of automata generation is not affected by failed generations.

Live play, by contrast, is inherently interaction-costly, as the model is queried at *every* decision point in *every* match. Thus, evaluation cost scales with both game length and the number of pairings. Since both the number of interactions (or moves) per match and the price per move are stochastic, we estimate cost using expected values. Specifically, we compute the expected cost of a single match as

$$C_{\text{match}} = \mathbb{E}[\text{cost per move}] \cdot \mathbb{E}[\text{moves per match}]. \tag{12}$$

We then assume a tournament with 30 other participants and 2 games per pair, yielding 60 matches per model, and estimate the total live-play cost as

$$C_{\text{total}} = 60 \cdot C_{\text{match}}. \tag{13}$$

This calculation is much lower than the true cost we would incur if we used all 105 participants (52 LLMs and 2 automata per LLM). However, even under this lenient estimate, the gap between live play and automata cost remains very large. In addition to the Chess comparison in Figure 8a, we also show results for Connect 4 in Figure 28. The same behavior is observed regardless of the game's difficulty, as chess is both more difficult and involves more actions per match.

P Automaton Randomness

Generated automata differ not only in strategic quality, but also in how deterministically they map game states to actions. In principle, an automaton can be fully deterministic, returning the same action whenever it is given the same state. In our observed set of LLM-generated policies, some included

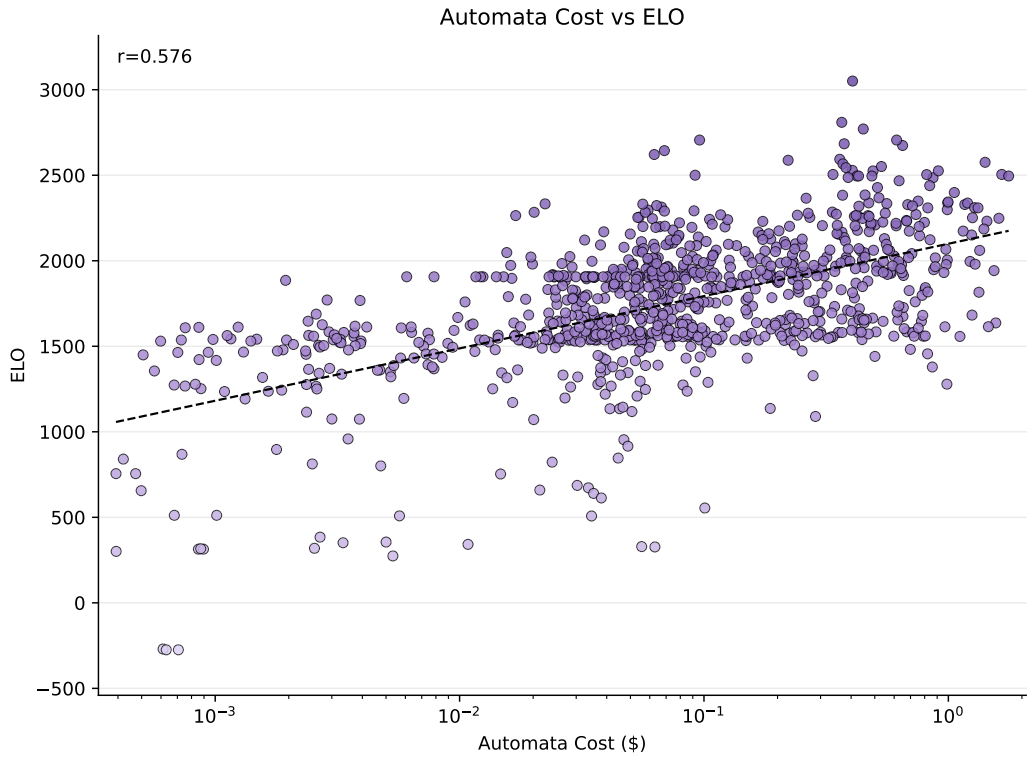


Figure 29: **Cost vs. Elo across all games.** Similar to Figure 8b but for all 20 games. The cost of a single automaton compared to its Elo in the tournament. As the Elo is the tournament Elo, only automata that passed the qualifiers are plotted. Details in Section O

stochastic components, such as random tie-breaks, time-based search cutoff to avoid timeouts, or purely random legal-action selection. This introduces an additional source of variation in match outcomes beyond the variation caused by sampling multiple automata from the same model (discussed in Section E).

We therefore classify generated automata according to their execution behavior. We make each automaton play against a seeded *Random* policy 100 times and record the actions taken (we apply 20 different seeds and repeat each seed 5 times). From the recorded actions we classify each automaton as being one of five categories. An "Error" category denotes that the automaton failed during execution the majority of the time. A "Timeout" category denotes that it exceeded the execution time limit the majority of the time (if the majority is a mix of "Error" and "Timeout" then it is classified as the more frequent of the two). Otherwise, the automaton succeeded in the majority of its games against the *Random* policy and is classified into one of three behavioral categories.

First, an automaton is classified as *Deterministic* if it produces the same action sequence across all repeated executions from the same initial condition. For these policies, repeated matches against the same opponent are expected to differ only when the opponent or game initialization differs.

Second, an automaton is classified as exhibiting *Intended Randomness* if its actions vary across different random seeds but are consistent across repeated executions with the same seed. This behavior is consistent with explicit use of seeded pseudorandomness (Python's `random` module or `numpy.random`). Such policies are stochastic, but their stochasticity is reproducible under a fixed seed. Third, an automaton is classified as exhibiting *Unintended Randomness* if its actions vary even when the same seed and same game trajectory are repeated. This indicates nondeterminism not controlled by the random seed. Common examples include time-dependent search cutoffs, where the chosen action depends on CPU scheduling or small variations in execution time, and iteration over unordered Python containers such as `set`, where element order may vary across processes or runs.

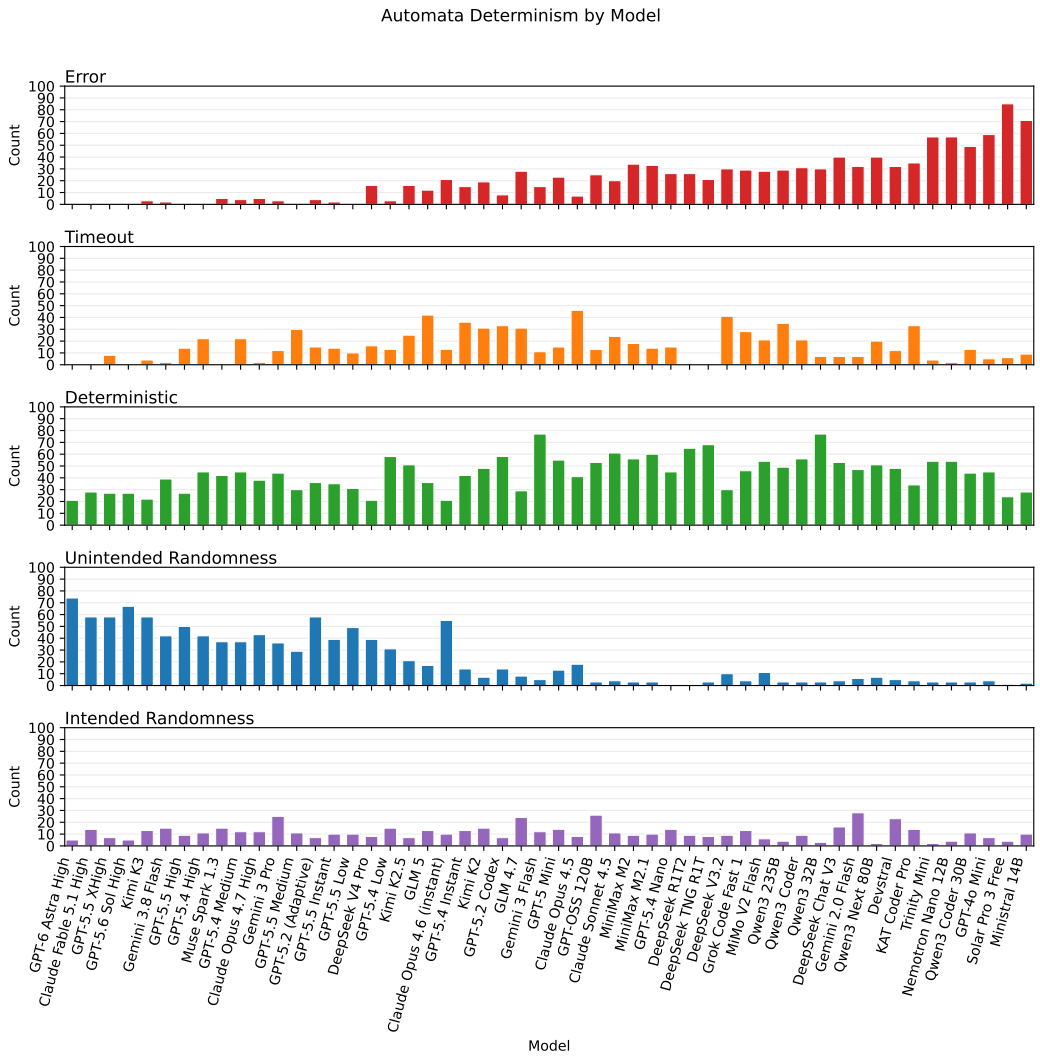
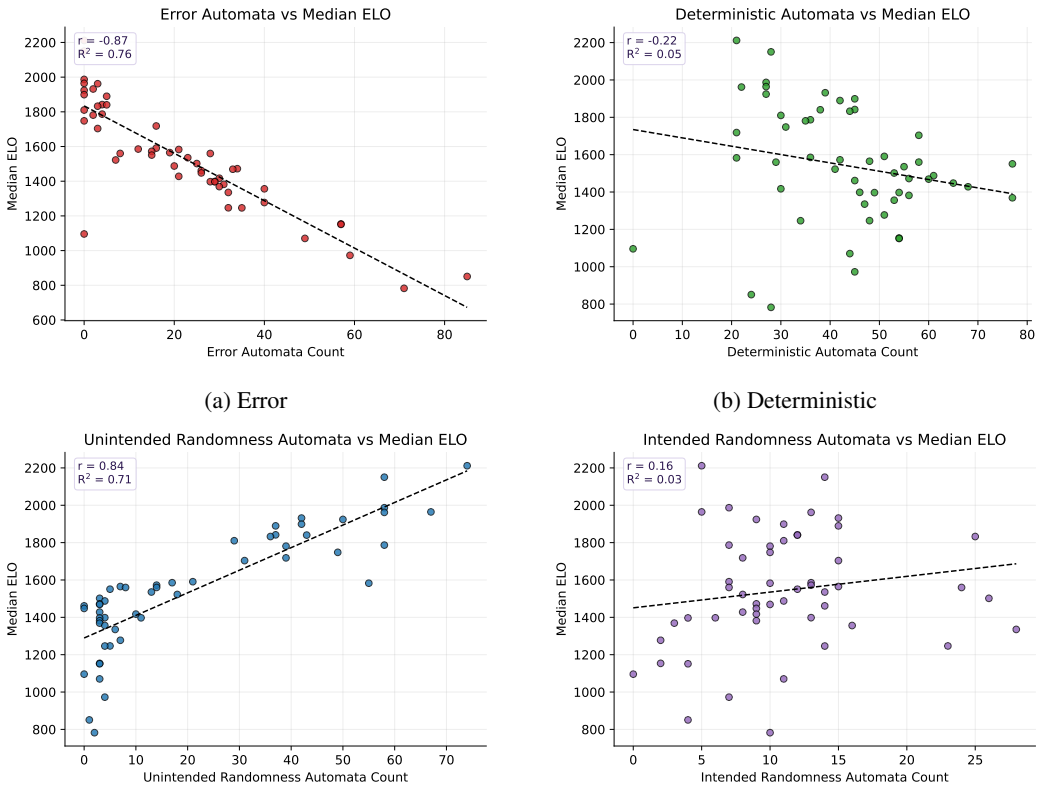


Figure 30: Randomness classification of all generated automata by model. Models on x-axis are sorted by median Elo. For each model, bars report the number of generated automata assigned to each of the five execution-behavior categories as explained in Section P.



(a) Error

(b) Deterministic

(c) Random

(d) Seedable randomness

Figure 31: Relationship between automata classification counts and median Elo across models. Each point is one model, with the x-axis showing the number of generated automata in the corresponding category and the y-axis showing the model's median Elo. Dashed lines show linear fits with Pearson correlation r and R^2 reported. More details are provided in Section P

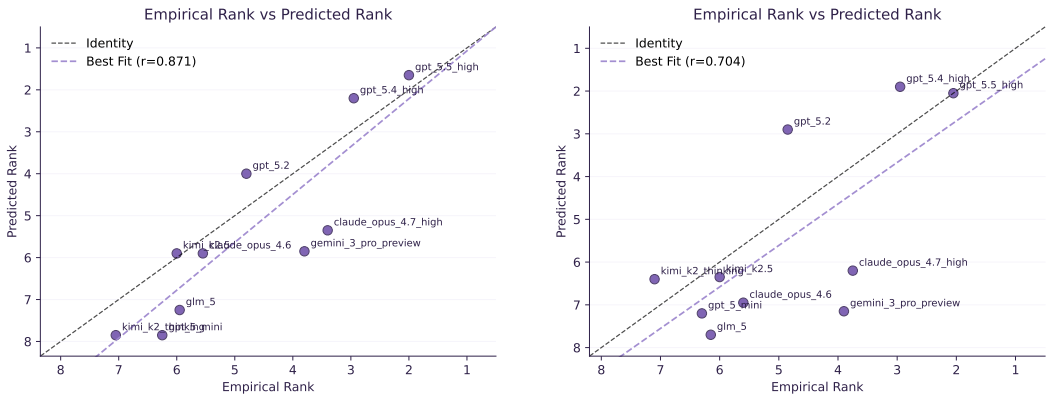
This form of randomness is less desirable for evaluation because it can introduce outcome variation that is not an explicit part of the generated strategy but is of course left intact as we never modify the automata generated by the LLM.

Figure 30 summarizes the frequency of each category across generated automata. Most successful automata are deterministic or exhibit unintended nondeterminism, while a smaller subset uses intended randomness. The error and timeout categories capture automata whose execution behavior is unreliable enough that their strategic quality cannot be meaningfully evaluated.

Figure 31 relates these categories to tournament performance. Automata that frequently error or timeout receive low Elo scores, as expected, since illegal actions and nontermination are treated as losses under the evaluation protocol. Among valid automata, deterministic and intended-randomness policies span a broad range of Elo values. In contrast, the count of automata exhibiting unintended randomness is strongly positively correlated with median Elo ($r = 0.84$). This suggests that some higher-performing models tend to use adaptive or execution-dependent policies, such as time-limited search, whose strength comes at the cost of nondeterminism not fully controlled by seeding.

Q LLM Simulator

An important question is whether LLMs can predict match outcomes directly from the generated automata. This is especially relevant in environments where exact simulation is expensive. If evaluating a policy pair requires substantially more computation than a single LLM query, then an



(a) Kimi K2.5 (b) GPT 5.4 Nano

Figure 32: LLM-predicted rankings versus empirical tournament rankings for the top 10 models. For each game, the oracle LLM is given the rules and source code for each pair of automata and predicts win/draw/loss probabilities. These predictions are aggregated across 900 automata pairs and converted into a predicted model ranking using Equation (11). Points closer to the diagonal indicate stronger agreement between source-code-based prediction and full simulation. More details are provided in Section Q

LLM-based predictor could serve as a cheap filtering mechanism, identifying promising automata for more rigorous downstream evaluation.

To test this, we select the top 10 LLMs from our main benchmark results (Section D). For each of the 20 games, we take the best-performing automaton generated by each of these models, yielding 10 automata per game. This gives 45 unordered automata pairs per game and 900 total pairs across all games. For each pair, we construct a prompt that includes the game rules (the same rules used to generate the automata) and provides the source code of both automata. We then ask an oracle LLM to predict the outcome distribution: the probability that each automaton wins and the probability of a draw.

We aggregate these predicted win/draw/loss probabilities across the 900 pairs and use Equation (11) to infer a predicted ranking of the 10 source models. We then compare this predicted ranking against the empirical ranking obtained from full tournament simulation in Section D. We run this experiment with two oracle models: GPT 5.4 Nano, representing a lightweight predictor, and Kimi K2.5, representing a stronger predictor. As shown in Figure 32, both models recover much of the empirical ordering from source code alone, with Kimi K2.5 achieving a correlation of $r = 0.87$ between predicted and empirical ranks.

More broadly, these results suggest that LLMs can serve not only as generators of executable policies, but also as approximate evaluators of such policies. In settings where exhaustive simulation is expensive or infeasible at full scale, an LLM-based simulator can provide a lightweight pre-evaluation layer prioritizing promising candidates for full empirical evaluation. This is especially useful when environments involve long horizons or costly rollouts. Because the evaluator reasons over the source code itself, this approach also makes the generated artifacts more inspectable as performance predictions can in principle be linked to identifiable strategic choices or flaws in the policy. Thus, LLM-based simulation should not be viewed as a replacement for execution-based ground truth, but as a scalable triage and analysis mechanism for making artifact evaluation more efficient and interpretable.

R Additional Details