HEURISTIC BENCH: EVALUATING LLM AGENTS BY AUTHORING EXECUTABLE POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents have shown strong capabilities in agentic problem solving. But assessing these capabilities across models in a human-interpretable and fine-grained way remains challenging. Inspired by a recent paradigm, *Heuristic Learning*, we introduce Heuristic Bench, a benchmark in which agents iteratively solve tasks by authoring executable code as policies that can be run and evaluated independently. As code, these policies are human-interpretable and inspectable, allowing readers to examine the strategies behind the scores and investigate potential shortcuts. Using this benchmark, we evaluate 47 models with varying coverage across 45 problem-solving tasks, with discrete and continuous action spaces in solo, competitive, and cooperative settings. Controlled execution, independent evaluation, and a wide range of metrics support reliable measurement of model performance. The benchmark's breadth allows us to rank models by their ability, and its analyses reveal fine-grained traits, such as performance-cost trade-offs across models and gains from additional attempts, uneven gains across model versions, imperfect yet still informative self-reported scores, and partial agreement in model rankings across tasks. Interestingly, policy inspection reveals that capable models can exploit unintended shortcuts, including copying opponent and task code to construct exact-simulation search policies. These findings highlight the importance of careful evaluation design to reveal model capabilities and inform how AI models are evaluated, used, and developed.

1 INTRODUCTION

Large language model (LLM) agents have shown strong problem-solving capabilities, including writing code, using tools, and iterative problem solving (Yao et al., 2023; Shinn et al., 2023; Wang et al., 2024). As these abilities develop, evaluating agents' capabilities across models and tasks through benchmarks (Liu et al., 2024b; Jimenez et al., 2024; Wang et al., 2026b; Sun et al., 2026) becomes essential. Measuring performance, however, is only part of the challenge. We need to understand the strategies behind their scores, and useful evaluations should therefore expose fine-grained traits in how agents solve problems in a concrete, human-interpretable way. A gap remains when capable models act as "one-off" solvers. Although powerful, they produce answers or actions for a particular task instance without leaving a reusable decision rule. Thus, their internal decision processes remain difficult to interpret, and success scores alone can conceal shortcuts or other sources of performance gains (Kapoor et al., 2025; 2026). Interaction traces offer evidence, but even explicit reasoning traces can hide an agent's intent once optimized for (Baker et al., 2025).

An alternative is for an agent to express its strategy as an executable code policy. Work on code-based policies shows how behavior can be represented in runnable programs (Liang et al., 2023), and program-search systems refine generated code using execution-based evaluation (Romera-Paredes et al., 2024; Liu et al., 2024a; Novikov et al., 2025). *Heuristic Learning* (Weng, 2026) presents a general paradigm in which a coding agent tests and revises code policies using execution feedback. In all these cases, the resulting policy code, with strategies ("heuristics") accumulated through refinement, can be run repeatedly and inspected independently after the agent finishes authoring it.

Following this paradigm, we introduce **Heuristic Bench** (Section 3), which uses agents' ability to author code policies as a basis for comparing models across a shared collection of tasks. Our benchmark spans 45 tasks with discrete and continuous action spaces in solo, competitive, and

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.

cooperative settings. Agents iteratively author policies under controlled execution conditions, and a separate process evaluates them using fixed protocols and shared measures. The benchmark thus supports both repeatable measurement of policy performance and analysis of how agents achieve it. Besides quantitative comparison, policies are human-interpretable and inspectable in their code form, so one can inspect the strategies behind scores and investigate shortcuts an agent may take.

We evaluate 47 models with varying task coverage (Section 4), ranking their ability to learn and examining the traits behind aggregate performance. The results reveal the distribution of performance and cost-related trade-offs among models and across the number of attempts needed for their best policies to emerge. Rollout galleries and code policies make strategies available for human inspection, which, in particular, reveals how highly capable models, when unshielded, can exploit access to existing solutions by copying an opponent's network weights or a trained controller. This highlights the necessity of careful system design to control the information agents can access. Further analyses show uneven gains across model versions, promising progress in routing-based models, a broad atlas of task difficulty in which even the best model faces challenges on hard tasks in continuous control, imperfect but informative self-reported scores, and partial agreement in model rankings across tasks.

Our contributions are a benchmark for iterative policy authoring with independent evaluation, a broad model comparison with analyses of model and task traits, and an artifact-based approach to investigating the strategies behind measured performance. Together, these support assessment of both how effectively agents solve problems and what their executable solutions reveal about their capabilities, informing further development of agentic learning.

2 Related Work

Agent benchmarks. Many agent benchmarks have recently been proposed. SWE-bench (Jimenez et al., 2024), Terminal-Bench (Merrill et al., 2026), MLE-bench (Chan et al., 2025), and Program-Bench (Yang et al., 2026) evaluate agents' ability to resolve GitHub issues, execute command-line workflows, perform machine-learning engineering, and rebuild programs from scratch, respectively. Agents' Last Exam (Sun et al., 2026) and ALE-Bench (Imajuku et al., 2025) evaluate long-horizon valuable workflows and algorithmic problem-solving capabilities. SEC-bench Pro (Lee et al., 2026), ExploitGym (Wang et al., 2026b), and CyberGym (WANG et al., 2026) evaluate agents on cybersecurity tasks. Meta discussions of agent evaluation examine what benchmarks reveal about agent capabilities and traits (Kapoor et al., 2025; 2026). Ours emphasizes agents' ability to develop executable policies while allowing analysis of both performance and the strategies or shortcuts used.

Evaluation validity and contamination. A score is reliable only when the evaluation is valid and contamination-proof. LLM evaluation faces contamination risks, as exposure to benchmark problems can inflate performance (Jain et al., 2025; White et al., 2025). Self-preference in LLM judges motivates independent scoring (Chen et al., 2025). Even with objective scoring, agents can take shortcuts, and instructions alone are insufficient to stop such cheating (Kouremetis et al., 2026). Careful design is needed, including audits of agent-produced artifacts for benchmark bugs (Zhu et al., 2025), search records for contamination (Wang et al., 2026a), and agents' thought traces for reward hacking (Baker et al., 2025). Meanwhile, infrastructure compromise motivates strong access controls (OpenAI, 2026c). We incorporate these lessons into our design. Reusable code policies as frozen artifacts reduces contamination risks, agents author policies under controlled and shielded conditions to avoid shortcuts, and an independent scoring process to avoid self-preference.

Programs as policies. Earlier work synthesizes programs from problem-solving traces (Langley & Choi, 2006) and input/output examples (Devlin et al., 2017). LEAPS (Trivedi et al., 2021) develops an approach connecting code search with interpretable policies. Code as Policies (Liang et al., 2023) and ProgPrompt (Singh et al., 2023) generate executable policies and plans for robotics tasks. FunSearch (Romera-Paredes et al., 2024), Evolution of Heuristics (Liu et al., 2024a) and AlphaEvolve (Novikov et al., 2025) propose searching executable code for heuristics and algorithms, and heuristic learning (Weng, 2026) emphasizes iterative revision by agents through feedback in code search. We follow this line of research and incorporate the idea into our benchmark design.

Self-improvement and accumulated knowledge. One line of agent self-improvement involves accumulating knowledge in a reusable form. DreamCoder (Ellis et al., 2021) accumulates abstractions from synthesized programs. For LLM agents, Voyager (Wang et al., 2024) accumulates reusable

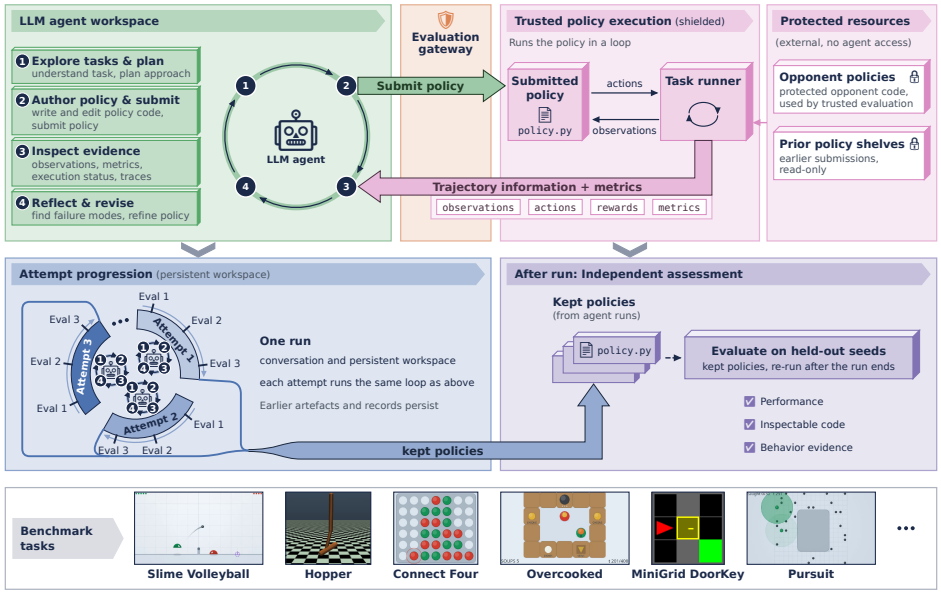


Figure 1: **Overview of our benchmark.** The agent is tasked with authoring a policy in code to solve a given task in an isolated environment. The submitted policies are handled by the execution platform, which runs each policy under shielding and returns permitted execution feedback. The agent also improves across multiple attempts. The benchmark supports a wide range of tasks.

Minecraft skills, while Reflexion (Shinn et al., 2023) and ExpeL (Zhao et al., 2024) keep agent experience in external memory. These LLM agents do not require weight updates. Another line is searching for executable code. ADAS (Hu et al., 2025) and the Darwin Gödel Machine (Zhang et al., 2026a) search over task-solving agents represented as code. Hyperagents (Zhang et al., 2026b) extend this search to a meta-agent that can modify itself. Combining these two directions, ALMA (Xiong et al., 2026) meta-learns memory mechanisms that accumulate lessons for agents. We follow this trend by incorporating a study of multiple attempts into our benchmark.

Games and multi-agent evaluation grounds. Game playing has long been used to benchmark AI agents (Genesereth et al., 2005; Perez-Liebana et al., 2019). Games have been used to evaluate single-agent and multi-agent reinforcement learning (Terry et al., 2021; Leibo et al., 2021; Towers et al., 2025), and more recently LLM agents in BALROG (Paglieri et al., 2025) and TextArena (Guertler et al., 2025). Games also motivate evaluating and ranking agents against opponents (Balduzzi et al., 2018; Omidshafiei et al., 2019). Inspired by this, we base our benchmark on a variety of game tasks and rank agents on a leaderboard. Ranking agents by head-to-head competition is an extension.

3 Method

3.1 Heuristic Bench Mechanism

Figure 1 provides an overview of our proposed benchmark. We draw inspiration from the feedback-driven approach of program-search systems (Romera-Paredes et al., 2024; Novikov et al., 2025) and Heuristic Learning (Weng, 2026). The agent, solving a task directly, authors a policy as code and iteratively revises it using feedback. Concretely, the agent submits policies to an execution platform that runs the task. At each step of an episode, the platform gives observations to the policy, receives its actions, and advances the task. The platform then assembles execution feedback, including trajectory information and metrics, and returns it to the LLM agent, which reflects on the feedback and uses it to improve later submissions. The agent is free to take several such steps.

The platform implements shielding in two parts. First, the LLM agent's workspace is isolated from the rest of the system, so it cannot freely probe the evaluation infrastructure or search the internet.

This matters because capable agents may find shortcuts easier than solving the intended problem. Second, the execution platform must verify submitted policy code and run it under controlled conditions. The platform keeps attempt artifacts after the run ends for quantitative assessment (Section 3.3) and for human inspection. Within these boundaries, the agent retains substantial freedom to solve the task. It can still choose what to submit, what lessons to draw from feedback, and how to use tools, including writing scripts for testing and analysis.

Furthermore, we use a multi-attempt mechanism to assess improvement through accumulated feedback, as explored in a line of work on self-improvement (Shinn et al., 2023; Hu et al., 2025; Zhang et al., 2026a). We call one complete interaction, which can include multiple policy submissions before submitting the final policy, an *attempt*. Each run is one continuous agent conversation. The agent starts the next attempt by creating a new directory, keeping the conversation (subject to harness context compaction) and the entire workspace, including earlier policies, trial records, analyses, and tools, without a platform reset. The agent must write an analysis of the previous attempt's failures before writing the new policy and a self-report after evaluation. Proposed lessons are harvested after the run and, following human review, seed future runs rather than the next attempt. This lets us observe whether agents find better policies over multiple attempts with accumulated feedback.

3.2 Benchmark Implementation

We describe the implementation details central to our benchmark. To drive the agent, the benchmark platform provides it with a brief specifying the full task contract. The brief gives general instructions on the goal and available tools, followed by task-specific specifications for observations, actions, rewards and scoring, player roles, the held-out evaluation target, and the evaluation call used during iteration through the same interface used for measurement. The brief provides no strategy, since telling the agent how to play would make the result depend on our hints. Appendix A.4 illustrates the complete instruction package.

We use Bubblewrap (Bubblewrap Contributors, n.d.) to provide the agent with an isolated workspace and tools. Under `unshielded` access, the agent can read task code and invoke the task runtime within its workspace. Under `shielded` access, protected implementations and prior run artifacts are withheld, and the agent must test its policies through the execution platform. Evaluation runs outside the sandbox and returns both trajectory information and metrics to the agent. Both access levels retain the same baseline protections, permitted tools, and policy submission and execution protocols. Submitted policies undergo import checks and run in a separate restricted sandbox. Appendix A.3 details the access rules, checks, and execution protections.

The intended evaluation target is a policy that acts using the task's permitted information within execution limits. One possible shortcut, which we observed in the controlled experiment in Section 4.3, is to copy both opponent and task implementation code to perform A^* search. Each search step considers candidate actions and evaluates their outcomes by exactly simulating the opponent and task dynamics. This demonstrates how powerful models exploit access to existing solutions when shielding is disabled. Such search can often lead to a very slow policy. Shielding the task and opponent code prevents this code-copying shortcut, while protected policy execution enforces time and memory limits to constrain computationally expensive search. These restrictions do not prohibit agents from modelling task dynamics using permitted observations and public task rules.

For artifact registration and evaluation, each submitted policy is assessed under a frozen protocol file. The protocol specifies the task, evaluated player role, score, episode count, seed block, opponent pool, ranking claims, and any ratified evaluation time limit. Seeds used during policy authoring by LLMs are separate from those used for independent measurement. For each batch, evaluation protocols, model-task assignments, and run specifications are fixed and tagged before launch.

3.3 Tasks, Metrics and Measures

Tasks We build our task collection primarily from established single-agent and multi-agent reinforcement learning (RL) tasks, including Atari console games and multi-agent evaluation tasks, alongside other ported games. We select established tasks with discrete and continuous action spaces in solo, competitive, and cooperative settings to reveal different model traits. Figure 1 shows examples of tasks and the full roster is in Appendix A.1. Three considerations guide our selection. (1)

We seek tasks that expose a broad range of performance differences across difficulty levels, while retaining room for improvement by powerful models (Section 4.1), (2) we select tasks that can be evaluated within reasonable time and cost budgets, given the large collection of tasks and models, and (3) our choice of established tasks allows calibration against non-LLM baselines (Section 4.4).

Performance metrics. We propose the following metrics to evaluate models across heterogeneous tasks. We independently score the attempts in each run and select its champion under the frozen protocol (Appendix A.2). As tasks have different score ceilings, we consider two ways to standardize measurements within each task for comparison across tasks. The z score (standardized score) emphasizes the magnitude of performance gaps. The percentile is on a 0–100 scale, emphasizing relative standing. We compute z scores over the same runs from the primary model roster pooled in the ranking at each task's reporting access level, using the sample standard deviation and scoring model-caused failures at the random-policy floor. Both measures use the selected champion's score, and higher values indicate better performance. Tasks with fewer than two valid runs or zero score variation contribute $z = 0$ and are flagged. Numerically tied scores receive the average of the ranks.

Aggregation and uncertainty. We aggregate each model's run-level z scores and percentiles separately using an empirical-Bayes one-way random-effects model, reporting posterior means and 95% posterior intervals. These estimates use fitted variance components, and the fit weights runs equally without adjusting for differences in task coverage or difficulty. Monte Carlo draws from the fitted posteriors give rank intervals. Appendix A.2 gives the estimator and its assumptions. Our emphasis on uncertainty in evaluation is motivated by Agarwal et al. (2021). Best denotes the highest measured single-run score, excludes imputed failure scores, and does not determine the ranking.

Attempt Rates and Resource Usage. To assess fine-grained differences in model capabilities, we report the *First Attempt* Ace Rate, the fraction of measured runs whose first attempt produces the selected champion. This indicates how often a model produces its champion on the first try, and a high value suggests the model "aced" the task. We also report *All Attempts*, the fraction of registered attempts yielding a measured score, which measures how successfully a model produces measurable policies across attempts. This rate helps distinguish authoring a valid policy from authoring a high-performing policy that solves the task, revealing finer-grained challenges for models. Cost and time are means per full authoring run, with costs taken from vendor reports or estimated from recorded token usage and registry rates. Pricing details are in Appendix B.1.

4 Experiments

4.1 Benchmarking Results

Overview. We ask what the benchmark reveals about differences in LLM capability for 47 models detailed in Appendix B.1. We compare models that author code using metrics defined in Section 3.3.

Results. Table 1 (and Appendix B.2) shows that the leading models combine strong aggregate performance with substantial differences in cost. However, peak single-run scores do not always translate into a high overall ranking. GPT-6 Astra has the highest posterior z and percentile means, although the leading models' posterior z intervals overlap. Competitive performance is available at substantially lower cost and shorter run times. GPT-5.6 Luna and DeepSeek V4.1 Flash cost less per run than other, more expensive leading models. Attempt success rates suggest that even for frontier models, the best policy often emerges only after further attempts, which affirms the self-improvement paradigm and motivates the analysis of iterative improvement in Section 4.2. Figure 2 shows the performance-cost trade-off and Pareto frontier, identifying models for which no cheaper alternative achieves a higher posterior z mean.

4.2 Iterative Improvement

Overview. We ask how the best policy found by an agent improves with additional attempts, in the spirit of self-improvement through reflection on feedback (Shinn et al., 2023). Figure 3 tracks each run's best measured score after successive attempts for selected tasks and models, showing improvements and stalls. Authoring agents receive evaluation scores, so they can identify and keep

Table 1: Key metrics for 10 highlighted models, including posterior z and percentile means, 95% empirical-Bayes posterior intervals ("95% I"), best scores, attempt rates, and mean cost and time (in min) per run. Model details in Appendix B.1. Full results for all models in Appendix B.2.

	Posterior z			Percentile			Attempt Rate		Per Full Run	
Model	**Mean**	**95% I**	**Best**	**Mean**	**95% I**	**Best**	**First**	**All**	**Cost**	**Time**
GPT-6 Astra	+0.846	[+0.66, +1.03]	+6.043	81	[76, 86]	100	2%	100%	$11.20	24.8
Claude Fable 5.1	+0.658	[+0.48, +0.84]	+3.787	70	[65, 75]	100	29%	100%	$5.48	16.5
GPT-5.6 Sol	+0.499	[+0.32, +0.68]	+2.188	71	[66, 76]	100	3%	100%	$6.34	24.8
Claude Fable 5	+0.489	[+0.31, +0.67]	+1.583	68	[63, 73]	98	19%	100%	$11.62	21.0
DeepSeek V4.1 Flash	+0.436	[+0.26, +0.62]	+3.851	62	[57, 67]	100	41%	96%	$0.12	14.5
Claude Opus 5	+0.413	[+0.23, +0.60]	+2.879	65	[60, 70]	100	12%	100%	$5.21	21.5
Muse Spark 1.2	+0.378	[−0.09, +0.85]	+1.093	48	[35, 62]	84	11%	100%	$1.08	10.5
GPT-5.6 Luna	+0.340	[+0.16, +0.52]	+2.016	58	[53, 63]	92	10%	99%	$0.09	18.0
Grok 4.6	+0.331	[+0.14, +0.52]	+1.377	63	[58, 68]	94	7%	99%	$5.51	22.3
Gemini 3.5 Flash-Lite	+0.265	[−0.21, +0.74]	+1.093	49	[36, 62]	84	22%	97%	$0.25	2.3

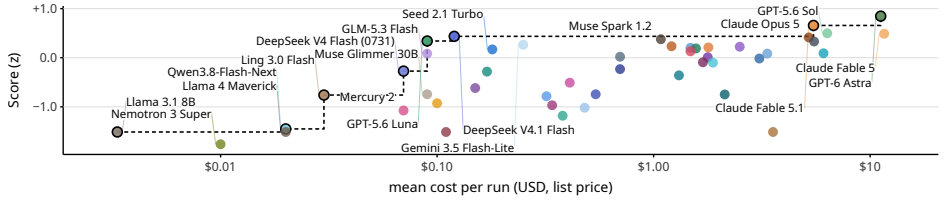


Figure 2: Mean cost per run versus posterior z, with a Pareto frontier based on point estimates. We also highlight models near the frontier since estimation uncertainty limits strict dominance claims.

their best policy so far. Table 2 summarizes improvement by model, using mean best-so-far within-task z scores across measured tasks after each attempt. We select one run per task, preferring a complete attempt record. Before the first scored attempt, the running best is the task's measured random-policy score. After the last attempt, its final value is carried forward.

Results. Individual trajectories show different paces of improvement across models (Figure 3), while best-so-far scores improve in aggregate with additional attempts (Table 2). Several attempts may pass before the next improvement. A new policy can perform worse and need not achieve a strictly better score. Still, best-so-far scores cannot decrease by construction because earlier policies remain available. We found that our empirical limit of 12 attempts provides a useful window for observing rapid gains and slower improvement. A formal analysis of convergence is left for future study. These observations support the usefulness of repeated attempts, without isolating the contribution of the lessons carried forward. Together with the scores and costs in Section 4.1, these trajectories suggest a route toward a performance-cost frontier through choices of model and attempt budget. Quantifying this joint frontier would require costs measured at each attempt budget.

4.3 Assessing Capable Models through Shielding

Overview. We examine whether strong policy scores reflect agents' ability to develop solutions or their access to existing ones. Relying on instructions alone is insufficient, as models still cheat on benchmark tasks even when explicitly instructed not to (Kouremetis et al., 2026). Thus, we compare runs where we deliberately create an *unshielded* condition, keeping the task, model, repeat, and budget unchanged, with runs in which opponent policies and prior champions are properly shielded (Section 3.2). These new runs are separate from the main benchmark ranking. We compare unshielded and shielded scores in three tasks' native units in Figure 4, with a closer view of paired Slime Volleyball scores in Figure 5.

Results. We observe large score drops under shielding in the challenging Slime Volleyball and Hopper tasks, but only small drops in the comparatively simple Connect Four task, where many agents nearly reach the ceiling even with shielding. These results suggest larger gains from access

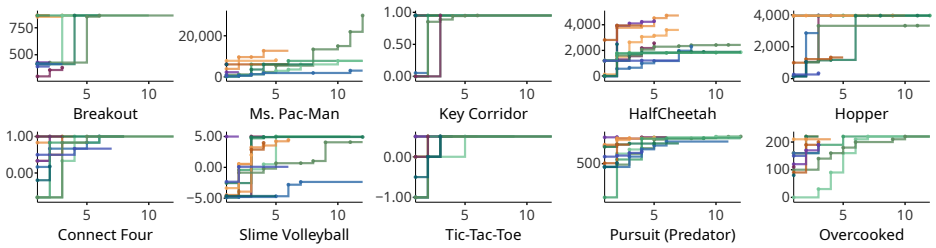


Figure 3: Running-best scores across policy-authoring attempts. Dots mark improvements.

Table 2: Best-so-far policy scores after each attempt for the highlighted models, reported as mean within-task z scores. These are simple means over one selected run per task, rather than posterior means over the fleet's runs as in Table 1. Note that z is standardized using final champions' scores, so models can receive large negative values in early attempts if their early policies are not good enough.

	Mean z of the best score so far, after attempt											
Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
GPT-6 Astra	−22.04	−0.77	−0.06	+0.22	+0.45	+0.53	+0.58	+0.66	+0.70	+0.77	+0.85	+0.91
Claude Fable 5.1	−0.21	+0.42	+0.51	+0.55	+0.59	+0.60	+0.61	+0.61	+0.61	+0.61	+0.61	+0.61
GPT-5.6 Sol	−26.79	−1.14	−0.14	+0.01	+0.19	+0.36	+0.41	+0.42	+0.45	+0.48	+0.49	+0.49
Claude Fable 5	−19.58	+0.06	+0.17	+0.28	+0.33	+0.36	+0.36	+0.36	+0.36	+0.36	+0.36	+0.36
DeepSeek V4.1 Flash	−0.09	+0.34	+0.45	+0.50	+0.50	+0.51	+0.51	+0.51	+0.51	+0.51	+0.51	+0.51
Claude Opus 5	−23.63	−0.12	+0.16	+0.23	+0.26	+0.32	+0.34	+0.34	+0.34	+0.34	+0.34	+0.34
Muse Spark 1.2	−1.13	−0.47	−0.32	−0.32	−0.01	−0.01	+0.06	+0.06	+0.06	+0.06	+0.06	+0.06
GPT-5.6 Luna	−21.08	−0.43	−0.09	+0.09	+0.18	+0.20	+0.23	+0.25	+0.29	+0.30	+0.31	+0.31
Grok 4.6	−20.13	−0.19	+0.05	+0.13	+0.24	+0.27	+0.28	+0.28	+0.29	+0.29	+0.29	+0.30
Gemini 3.5 Flash-Lite	−1.39	−0.50	−0.26	−0.26	−0.26	−0.26	+0.00	+0.00	+0.00	+0.00	+0.00	+0.00

to existing solutions on the more challenging tasks compared here. Strong unshielded Slime Volleyball scores can also fall sharply under shielding. For example, the highlighted models include Claude Fable 5.1 and GPT-5.6 Sol, which also rank highly in Table 1. Inspection shows that unrestricted access can yield high benchmark scores through solution reuse. As detailed in Appendix C, two unshielded Slime Volleyball policies reproduce the opponent's network weights, and an unshielded Hopper policy transcribes a trained controller. We argue that capable models can direct their problem-solving ability toward unintended shortcuts, which relates to risks of infrastructure compromise identified during searches for benchmark solutions (OpenAI, 2026c). Thus, evaluating increasingly capable agents requires carefully engineered access controls and inspection of how they achieve their scores.

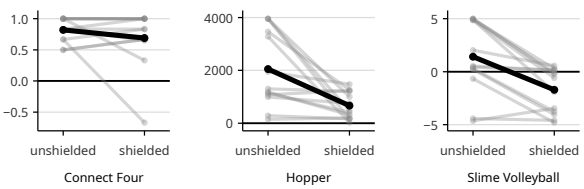


Figure 4: Unshielded-to-shielded scores. Grey lines show paired runs of a single model. Dark lines show means.

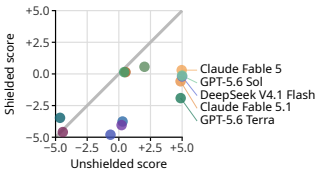


Figure 5: Slime Volleyball paired scores.

4.4 Analysis

Inspectable policy. Executable policies expose concrete strategies that can be examined alongside their scores. The best measured shielded Slime Volleyball champion was authored by GPT-6 Astra and scores +0.663 against the built-in recurrent-network baseline opponent over 1,000 held-out episodes. Figure 6 shows one rollout, and Code 1 shows the contact-scoring logic. The policy combines ball-flight prediction, searches over jump choices and horizontal movement targets, and

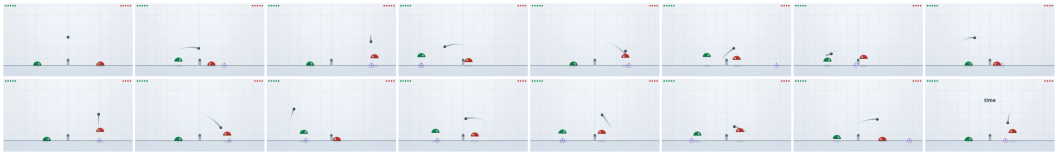


Figure 6: GPT-6 Astra's shielded Slime Volleyball policy. Frames are read left to right, then top to bottom. They show one game in which the agent-authored policy scores against the RNN baseline.

```python
    def act(obs):
        nonlocal armed
        x,y,vx,vy,bx,by,bvx,bvy,ox,oy,ovx,ovy = [float(v)*10 for v in obs]
        if bx<0 or (abs(bx)<0.001 and abs(by -12)<0.001):
            armed = False
        path = flight(bx,by,bvx,bvy)
        lx = next((p[0] for p in path if p [1]<3.2),path[-1][0])
        fallback = max(2,min(22.5,lx)) if lx > 0 else HOME
        best = -1e9
        best_set = False
        best_i = -1
        choice = (-1 if x>fallback+0.25 else 1 if x<fallback-0.25 else 0,0)
        if bx < -5 and bvx < 0:
            return [int(choice[0]<0),int(choice[0]>0),0]
        for jump in (0,1) if y < 1.51 else (0,):
            sy,svy = y,vy
            vertical = []
            targets = {round(fallback*12)/12}
            for i,(px,py,ux,uy) in enumerate(path[:65]):
                svy = 13.5*jump if sy <= 1.5034 else svy-0.98
                sy = max(1.5,sy+svy/30)
                if sy == 1.5:
                    svy = 0
                vertical.append((sy,svy))
                if px>0 and 0<py-sy<2.5 and abs(px-x)<17.5*(i+1)/30+2:
                    for off in (-0.9,0,0.9):
                        targets.add(round(max(2,min(22.5,px+off))*12)/12)
            # Close destinations duplicate the same actuator sequence.
            for target in sorted(targets):
                sx = x
                first = 0
                for i,((px,py,ux,uy),(sy,svy)) in enumerate(zip(path,vertical)):
                    a = -1 if sx>target+0.25 else 1 if sx<target-0.25 else 0
                    if i == 0:
                        first = a
                    oldx = sx
                    sx = max(2,min(22.5,sx+a*17.5/30))
                    pvx = (sx-oldx)*30
                    dx,dy = px-sx,py-sy
                    d2 = dx*dx+dy*dy
                    if 0.001<d2<4:
                        d = math.sqrt(d2)
                        nx,ny = dx/d,dy/d
                        un = (ux-pvx)*nx+(uy-svy)*ny
                        outx,outy = ux-2*un*nx,uy-2*un*ny
                        attack = shot(px,py,outx,outy,ox)
                        setting = setup_score(px,py,outx,outy,sx) if not armed else -100
                        score = max(attack,setting)-0.012*i
                        if score>best:
                            best,choice = score,(first,jump)
                            best_set,best_i = setting>attack,i
                        break
        if best_set and best_i == 0:
            armed = True
        return [int(choice[0]<0),int(choice[0]>0),choice[1]]
```

Code 1: Action selection in the showcased policy. Complete code in Code 2, Appendix D.1.

comparisons of attacking shots and setup contacts before returning an action. This means readers can examine both effective strategies and unintended shortcuts behind the scores (Section 4.3), which illustrates why human-interpretable, inspectable policy code matters. Appendix D.2 provides a gallery of policies from the best-performing models across tasks.

At the model level, strong models tend to produce longer policies with higher z scores (Pearson's $r = +0.54$). However, at the individual-policy level, source line count has a much weaker association with z scores (Pearson's $r = +0.14$). This suggests that longer policies are not necessarily stronger and that policy length is better attributed to a model's capacity and traits. Self-reported scores also often agree with independent measurements, with agreement in 74% of cases assuming a 1% tolerance. Prior work finds that LLM judges can favor their own outputs (Panickssery et al., 2024; Chen et al., 2025). Here, access to objectively scored task trajectories may help explain the agreement, since agents can report execution results rather than judge the quality of their own output. Nevertheless, noticeable errors remain and overestimates outnumber underestimates. These findings support inspecting policies and measuring them independently. Appendix D.3 provides details.

Model invocation traits. Given providers' rapid iteration, we examine model version and thinking tier as agent invocation choices. For the compared model pairs, we observe uneven improvements across matched tasks, with larger gains for DeepSeek on discrete puzzle games and for Claude and Gemini on continuous-control tasks and multiplayer games. We also observe regressions on some tasks. Details are in Appendix D.4. We conduct an initial exploration covering four models and three tasks, without observing a consistent benefit from increasing the thinking tier above high. This observation motivates a more systematic study of thinking tiers.

Routing-based models. A recent direction extends agentic improvement into model services by dynamically orchestrating calls to downstream LLM agents based on context. Recent examples include Fugu and Fugu Ultra (Sakana AI, 2026), whose small trainable parameter set could facilitate adaptation to new use cases. We investigate how these services perform on our benchmark. Fugu Ultra (orchestrating multiple agent workers) and Fugu (routing to a single agent worker) respectively achieve performance comparable to that of GPT-6 Astra and Claude Fable 5.1, the two highest-ranking models in the primary comparison, suggesting that routing is a promising approach to im-

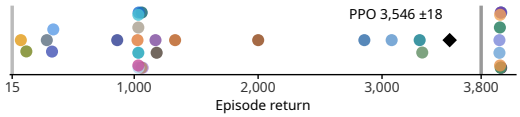


Figure 8: Hopper episode returns for code policies (dots) and trained PPO (diamond).

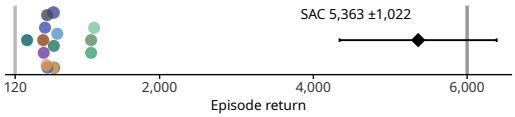


Figure 9: Humanoid episode returns for code policies (dots) and trained SAC (diamond).

proving model performance. Because routing-based models use a different mechanism from the other models, we exclude them from the primary comparison. Further details are in Appendix D.5.

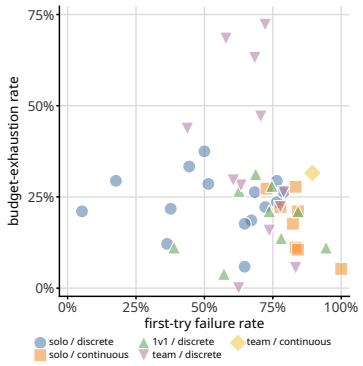


Figure 7: First-try failure versus budget exhaustion.

Atlas of task difficulty. When the best policy emerges and whether a run exhausts its budget capture different aspects of difficulty. Figure 7 compares first-try failure and budget-exhaustion rates by task categories. First-try failure means the champion is found after attempt 1. Budget exhaustion means a run stops at its token or wall-clock limit. These rates measure how often later attempts improve on the first and how often runs exhaust their budget. Notably, budget exhaustion can co-exist with finding the best policy early.

Continuous-control results are mixed. We trained Proximal Policy Optimization (PPO) (Schulman et al., 2017) and Soft Actor-Critic (SAC) (Haarnoja et al., 2018) policies on the simpler Hopper and much harder Humanoid tasks, respectively. These baselines use Stable-Baselines3 with fixed training budgets and are evaluated over 30 episodes with deterministic actions. Details are in Appendix D.6. Figures 8 and 9 show that our best Hopper code policy exceeds both the solved threshold and PPO's score, but a large performance gap remains on Humanoid. These comparisons use different training procedures and interaction budgets. The Humanoid gap leaves room for improvement even for frontier models. We hypothesize that additional guidance or substantially larger time and cost budgets could help.

Cross-task performance. We compare standardized model scores across task pairs to identify task characteristics within our suite. We find that scores are partially correlated across tasks. Appendix D.7 provides the agreement matrix and comparisons of selected task pairs. The results favour evaluation across diverse tasks over reliance on a single task or family.

Comparison with established benchmarks. We compare our percentile ranking with 15 public leaderboards using Kendall's τ_b when at least five models are shared. Agents' Last Exam (Sun et al., 2026) and ALE-Bench (Imajuku et al., 2025) show positive but imperfect agreement and each shares more than ten models with our ranking. Appendix D.8 gives the full table and a rank comparison with ALE-Bench, which shares the most models with our benchmark. Although limited model overlap and differing evaluation settings make firm conclusions difficult, the positive but imperfect agreement reveals similarities and differences in model ordering. This connects our benchmark to existing evaluations while offering distinct evidence and insights into agentic LLM capabilities.

5 Conclusion

We introduced Heuristic Bench to evaluate LLM agents through iterative policy authoring and independent execution. Our evaluation of 47 models across 45 tasks reveals improvements over successive attempts, performance-cost trade-offs, and uneven capabilities across tasks. Beyond aggregate scores, the resulting executable policies support human inspection of strategies. This inspection reveals a wide range of model traits. Together, these findings support using inspectable policies and controlled evaluation to understand both what agents can achieve and how they achieve it, providing unique insights into agentic LLM capabilities.

AI USE STATEMENT

In this work, (1) we used generative AI tools for tasks requiring disclosure, including designing or providing feedback on research methodology or experiments, implementing methods, and supporting qualitative and thematic data analysis. (2) We have not used generative AI tools to interpret results. (3) The remaining tasks requiring disclosure are not applicable to this work. These include generating synthetic datasets, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, assisting with translation, and cleaning and reformatting datasets. (4) Additionally, we used generative AI tools for tasks for which disclosure is recommended, including creating or modifying scientific figures or images, creating or editing software code, brainstorming, discovering research topics or identifying gaps, and identifying relevant literature.

We have reviewed all AI-assisted work. We checked LLM-generated figures/plots and code that was used for running tasks. Through a manual literature survey, we also checked all ideas and related work identified by LLMs during brainstorming. We used LLMs for grammar checking and language polishing in the writing. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our benchmark emphasizes the importance of proper shielding when evaluating agents to prevent unwanted behaviours. As our benchmark is built on publicly available tasks, we expect their availability to facilitate use of the benchmark. We identify no other ethical concerns arising from this work.

REPRODUCIBILITY STATEMENT

We provide benchmark details to support reproducibility. Besides the descriptions of our method (Section 3) and experiments (Section 4) in the main text, Appendix A.1 provides references and implementation details for the tasks we adopt and reuse in this benchmark. Appendices A.2 and A.3 provide further methodological details. Details of the models benchmarked are provided in Appendix B.1. We also provide downloadable source code as supplementary material.

REFERENCES

John P Agapiou, Alexander Sasha Vezhnevets, Edgar A Duéñez-Guzmán, Jayd Matyas, Yiran Mao, Peter Sunehag, Raphael Köster, Udari Madhushani, Kavya Kopparapu, Ramona Comanescu, DJ Strouse, Michael B Johanson, Sukhdeep Singh, Julia Haas, Igor Mordatch, Dean Mobbs, and Joel Z Leibo. Melting pot 2.0. *arXiv preprint arXiv:2211.13746*, 2022. URL https://arxiv.org/abs/2211.13746.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 29304–29320. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/f514cec81cb148559cf475e7426eed5e-Paper.pdf.

Amazon Artificial General Intelligence. Amazon nova premier: Technical report and model card. *Amazon Technical Reports*, 2025. URL https://www.amazon.science/publications/amazon-nova-premier-technical-report-and-model-card.

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised state representation learning in atari. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/6fb52e71b837628ac16539c1ff911667-Paper.pdf.

Anthropic. Claude Fable 5 and Claude Mythos 5 System Card. Technical report, 2026a. URL https://www.anthropic.com/claude-fable-5-mythos-5-system-card.

Anthropic. Claude Fable 5.1 and Claude Mythos 5.1 System Card. Technical report, 2026b. URL https://www.anthropic.com/claude-fable-5-1-mythos-5-1-system-card.

Anthropic. Claude Opus 4.8 System Card. Technical report, 2026c. URL https://www.anthropic.com/claude-opus-4-8-system-card.

Anthropic. Claude Opus 5 System Card. Technical report, 2026d. URL https://www.anthropic.com/claude-opus-5-system-card.

Anthropic. Claude Sonnet 5 System Card. Technical report, 2026e. URL https://www.anthropic.com/claude-sonnet-5-system-card.

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation, 2025. URL https://arxiv.org/abs/2503.11926.

David Balduzzi, Karl Tuyls, Julien Perolat, and Thore Graepel. Re-evaluating evaluation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/cdf1035c34ec380218a8cc9a43d438f9-Paper.pdf.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. doi: 10.1613/jair.3912.

Bubblewrap Contributors. Bubblewrap. GitHub repository, n.d. URL https://github.com/containers/bubblewrap. Accessed September 20, 2026.

ByteDance Seed. Seed2.0 Model Card: Towards Intelligence Frontier for Real-World Complexity. Technical report, 2026a. URL https://arxiv.org/abs/2607.00248.

ByteDance Seed. Seed2.1 Model Card: Agentic Intelligence for Productivity. Technical report, 2026b. URL https://lf3-static.bytednsdoc.com/obj/eden-cn/lapzild-tss/ljhwZthlaukjlkulzlp/seed2.1/Seed2_1_Model_Card.pdf.

Micah Carroll, Rohin Shah, Mark Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/f5b1b89d98b7286673128a5fb112cb9a-Paper.pdf.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. Mlebench: Evaluating machine learning agents on machine learning engineering. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 50466–50494, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/7e3767db483c942b883eb4f8cfb74e31-Paper-Conference.pdf.

Liang Chen, Weichu Xie, Yiyan Liang, Hongfeng He, Haozhe Zhao, Zhibo Yang, Zhiqi Huang, Haoning Wu, Haoyu Lu, Y. Charles, Yiping Bao, YuanTao Fan, Guopeng Li, Haiyang Shen, Xuanzhong Chen, Wendong XU, Shuzheng Si, Zefan Cai, Wenhao Chai, Ziqi Huang, Fangfu Liu, Tianyu Liu, Baobao Chang, Ming Wu, Xiaobo Hu, Kaiyuan Chen, Yixin Ren, Yang Liu, Yuan Gong, and Kuan Li. BabyVision: Visual reasoning beyond language. In *Forty-third International Conference on Machine Learning*, 2026.

Zhi-Yuan Chen, Hao Wang, Xinyu Zhang, Enrui Hu, and Yankai Lin. Beyond the surface: Measuring self-preference in LLM judgments. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1653–1672, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.86. URL https://aclanthology.org/2025.emnlp-main.86/.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and J Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 73383–73394. Curran Associates, Inc., 2023. doi: 10.52202/075280-3209. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/e8916198466e8ef218a2185a491b49fa-Paper-Datasets_and_Benchmarks.pdf.

DeepSeek-AI. DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence. Technical report, 2026a. URL https://arxiv.org/abs/2606.19348.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. Technical report, 2026b. URL https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash/blob/main/DeepSeek_V41_Tech_Report.pdf.

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel rahman Mohamed, and Pushmeet Kohli. RobustFill: Neural program learning under noisy I/O. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 990–998. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/devlin17a.html.

A. K. Dewdney. Computer recreations. *Scientific American*, 250(5):14–22, 1984. doi: 10.1038/scientificamerican0584-14.

Jingzhe Ding, Shengda Long, Changxin PU, Ge Zhang, zhou huan, Hongwan Gao, Xiang Gao, Chao He, Yue Hou, FEI HU, Zhaojian Li, Weiran Shi, Zaiyuan Wang, Daoguang Zan, Chenchen Zhang, Xiaoxu Zhang, Chen Qizhi, Xianfu Cheng, Bo Deng, Qingshui Gu, Kai Hua, Juntao Lin, Pai Liu, Mingchen Li, Minghao Li, Xuanguang Pan, Zifan Peng, Yujia Qin, Yong Shan, Zhewen Tan, Haoran Wang, Zihan Wang, Weihao Xie, Yishuo Yuan, Jiayu Zhang, Yunfei Zhao, He Zhu, LIYA ZHU, Chenyang Zou, Ming Ding, Jianpeng Jiao, Jiaheng Liu, Minghao Liu, Qian Liu, Chongyang Tao, Jian Yang, Tong Yang, Zhaoxiang Zhang, Xinjie Chen, and Wenhao Huang. NL2repo-bench: Towards long-horizon repository generation evaluation of coding agents. In *Forty-third International Conference on Machine Learning*, 2026.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI '21, pp. 835–850. ACM, June 2021. doi: 10.1145/3453483.3454080. URL http://dx.doi.org/10.1145/3453483.3454080.

Stefano Ermon. Introducing Mercury 2. Inception product announcement, 2026. URL https://www.inceptionlabs.ai/blog/introducing-mercury-2.

Florian Felten, Umut Ucak, Hicham Azmani, Gao Peng, Willem Röpke, Hendrik Baier, Patrick Mannion, Diederik M Roijers, J K Terry, El Ghazali Talbi, Grégoire Danoy, Ann Nowe, and Roxana Rădulescu. MOMAland: A set of benchmarks for multi-objective multi-agent reinforcement learning. *Journal of Data-centric Machine Learning Research*, 2026. URL https://openreview.net/forum?id=vzHLRK0sSp.

Suhaas Garre, Chris Mutty, Sushant Mehta, and Edwin Chen. Chartography: A benchmark for professional chart understanding, 2026. URL https://arxiv.org/abs/2608.10677.

Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aaai competition. *AI Magazine*, 26(2):62–72, 2005. doi: 10.1609/aimag.v26i2.1813.

GLM-5 Team. GLM-5: From Vibe Coding to Agentic Engineering. Technical report, 2026. URL https://arxiv.org/abs/2602.15763.

Google DeepMind. Gemini 3.1 Pro: Model Card. Model card, 2026a. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/.

Google DeepMind. Gemini 3.5 Flash: Model Card. Model card, 2026b. URL https://deepmind.google/models/model-cards/gemini-3-5-flash/.

Google DeepMind. Gemini 3.5 Flash-Lite: Model Card. Model card, 2026c. URL https://deepmind.google/models/model-cards/gemini-3-5-flash-lite/.

Google DeepMind. Gemini 3.6 Flash: Model Card. Model card, 2026d. URL https://deepmind.google/models/model-cards/gemini-3-6-flash/.

Google DeepMind. Gemini 3.7 Flash: Model Card. Model card, 2026e. URL https://deepmind.google/models/model-cards/gemini-3-7-flash/.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Leon Guertler, Bobby Cheng, Simon Yu, Bo Liu, Leshem Choshen, and Cheston Tan. Textarena, 2025. URL https://arxiv.org/abs/2504.11442.

Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In Gita Sukthankar and Juan A. Rodriguez-Aguilar (eds.), *Autonomous Agents and Multiagent Systems*, pp. 66–83, Cham, 2017. Springer International Publishing. ISBN 978-3-319-71682-4.

David Ha. Slime volleyball gym environment. https://github.com/hardmaru/slimevolleygym, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/36b7acf6f6010652b3f2a433774a66fe-Paper-Conference.pdf.

Wenqi Huang and Peter Jiang. Deepswe v1.1: a cleaner, more reproducible benchmark for frontier coding agents, 2026. URL https://github.com/datacurve-ai/deep-swe.

Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. Ale-bench: A benchmark for long-horizon objective-driven algorithm engineering. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-5226. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/e555eab2c68f886052f404397ea52001-Paper-Datasets_and_Benchmarks_Track.pdf.

inclusionAI. Ling-3.0-Flash. Model card, 2026. URL https://huggingface.co/inclusionAI/Ling-3.0-flash.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/94074dd5a072d28ff75a76dabed43767-Paper-Conference.pdf.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf.

Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=Zy4uFzMviZ.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, Franck Ndzomga, Dheeraj Oruganty, Sophie Luskin, Kangheng Liu, Botao Yu, Amit Arora, Dongyoon Hahm, Harsh Trivedi, Huan Sun, Juyong Lee, Tengjun Jin, Yifan Mai, Yifei Zhou, Yuxuan Zhu, Rishi Bommasani, Daniel Kang, Dawn Song, Peter Henderson, Yu Su, Percy Liang, and Arvind Narayanan. Holistic agent leaderboard: The missing infrastructure for ai agent evaluation. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 98778–98849, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/a0928f924a344aaebbb7f6cd8d56e34c-Paper-Conference.pdf.

Kimi Team. Kimi K2: Open Agentic Intelligence. Technical report, 2025. URL https://arxiv.org/abs/2507.20534.

Kimi Team. Kimi K3: Open frontier intelligence. Technical report, 2026. URL https://github.com/MoonshotAI/Kimi-K3/blob/main/k3_tech_report.pdf.

Michael Kouremetis, Ads Dawson, Raja Sekhar Rao Dheekonda, and Brian Greunke. Every model cheats: Prompt-level mitigation of cheating on offensive cyber tasks, 2026. URL https://arxiv.org/abs/2607.21763.

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, 2019. URL https://arxiv.org/abs/1908.09453.

Pat Langley and Dongkyu Choi. Learning recursive control programs from problem solving. *Journal of Machine Learning Research*, 7(17):493–518, 2006. URL http://jmlr.org/papers/v7/langley06a.html.

Hwiwon Lee, Jiawei Liu, Dongjun Kim, Wubing Xia, Ziqi Zhang, Chunqiu Steven Xia, and Lingming Zhang. Sec-bench pro: Can language models solve long-horizon software security tasks?, 2026. URL https://arxiv.org/abs/2605.26548.

Joel Z Leibo, Edgar A Dueñez-Guzman, Alexander Vezhnevets, John P Agapiou, Peter Sunehag, Raphael Koster, Jayd Matyas, Charlie Beattie, Igor Mordatch, and Thore Graepel. Scalable evaluation of multi-agent reinforcement learning with melting pot. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6187–6199. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/leibo21a.html.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500. IEEE, May 2023. doi: 10.1109/icra48891.2023.10160591. URL http://dx.doi.org/10.1109/ICRA48891.2023.10160591.

Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 21–27 Jul 2024a. URL https://proceedings.mlr.press/v235/liu24bs.html.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/e9df36b21ff4ee211a8b71ee8b7e9f57-Paper-Conference.pdf.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search of static linear policies is competitive for reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/7634ea65a4e6d9041cfd3f7de18e334a-Paper.pdf.

Ryan Marten. Terminal-Bench 4.0, 2026. URL https://www.tbench.ai/news/terminal-bench-4-0.

Ryan Marten, Alex Shaw, and Andy Konwinski. Terminal-Bench 3.0, 2026. URL https://www.tbench.ai/news/terminal-bench-3-0.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong (Ryan) Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/444a3737adaee10d86ad2ef5f74468e6-Paper-Conference.pdf.

Meta. Llama 4 Maverick Model Card. Model card, 2025. URL https://huggingface.co/meta-llama/Llama-4-Maverick-17B-128E-Instruct.

Meta Superintelligence Lab. Muse Glimmer Model Card. Model card, 2026. URL https://huggingface.co/meta-models/Muse-Glimmer-30B/blob/main/README.md.

Meta Superintelligence Labs. Introducing muse code and muse spark 1.2. Meta AI Research blog, August 2026. URL https://research.meta.ai/blog/introducing-muse-code-and-muse-spark-1-2.

MiniMax. MiniMax M3: Frontier coding, 1M context, native multimodality—all in one model. MiniMax Research blog, June 2026. URL https://www.minimax.io/blog/minimax-m3.

Mistral AI. Introducing Mistral 3. Mistral AI release announcement, December 2025. URL https://mistral.ai/news/mistral-3/.

Moonshot AI. Kimi K2.6. Model card, 2026. URL https://huggingface.co/moonshotai/Kimi-K2.6.

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

NVIDIA. Nemotron 3 Super: Open, Efficient Mixture-of-Experts Hybrid Mamba-Transformer Model for Agentic Reasoning. Technical report, 2026. URL https://arxiv.org/abs/2604.12374.

Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M. Czarnecki, Marc Lanctot, Julien Perolat, and Remi Munos. α-Rank: Multi-agent evaluation by evolution. *Scientific Reports*, 9(1):9937, 2019. doi: 10.1038/s41598-019-45619-9.

OpenAI. GPT-5.6 System Card. Technical report, 2026a. URL https://deploymentsafety.openai.com/gpt-5-6.

OpenAI. GPT-6 Astra System Card. Technical report, 2026b. URL https://deploymentsafety.openai.com/gpt-6-astra.

OpenAI. The Hugging Face incident and the road ahead. OpenAI blog, August 2026c. URL https://openai.com/index/hugging-face-incident-and-the-road-ahead/.

Davide Paglieri, Bartł omiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Woł czyk, Akbir Khan, Eduardo Pignatelli, Ł ukasz Kuciński, Lerrel Pinto, Rob Fergus, Jakob Foerster, Jack Parker-Holder, and Tim Rocktaeschel. Balrog: Benchmarking agentic llm and vlm reasoning on games. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 96666–96702, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/f0b1515be276f6ba82b4f2b25e50bef0-Paper-Conference.pdf.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. Llm evaluators recognize and favor their own generations. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 68772–68802. Curran Associates, Inc., 2024. doi: 10.52202/079017-2197. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/7f1f0218e45f5414c79c0679633e47bc-Paper-Conference.pdf.

Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius, and Simon M. Lucas. General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions on Games*, 11(3):195–214, 2019. doi: 10.1109/tg.2019.2901021.

Qwen Team. Qwen3-max: Just scale it, September 2025. URL https://qwen.ai/blog?id=241398b9cd6353de490b0f82806c7848c5d2777d.

Qwen Team. Qwen3.8-Flash-Next: A new architecture, towards ultimate cost-efficiency, August 2026a. URL https://qwen.ai/blog?id=qwen3.8-flash-next.

Qwen Team. Qwen3.7: The agent frontier, May 2026b. URL https://qwen.ai/blog?id=qwen3.7.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026c. URL https://qwen.ai/blog?id=qwen3.8.

Antonin Raffin. Rl baselines3 zoo. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Jonathan Roberts, Mohammad Reza Taesiri, Ansh Sharma, Akash Gupta, Samuel Roberts, Ioana Croitoru, Simion-Vlad Bogolin, Jialu Tang, Florian Langer, Vyas Raina, Vatsal Raina, Hanyi Xiong, Vishaal Udandarao, Jingyi Lu, Chen Shiyang, Sam Purkis, Tianshuo Yan, Wenye Lin, Gyungin Shin, Qiaochu Yang, Anh Totti Nguyen, David Atkinson, Alexandru Coca, Mikah Dang, Sebastian Dziadzio, Jakob D. Kunz, Kaiqu Liang, Alexander Lo, Brian Pulfer, Steven Walton, Charig Yang, Kai Han, and Samuel Albanie. Zerobench: An impossible visual benchmark for contemporary large multimodal models. In *Forty-third International Conference on Machine Learning*, 2026.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Sakana AI. Sakana Fugu technical report, June 2026. URL https://arxiv.org/abs/2606.21228.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Daniel Shepard and Robin Salimans. Automationbench, 2026. URL https://arxiv.org/abs/2604.18934.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530. IEEE, May 2023. doi: 10.1109/icra48891.2023.10161317. URL http://dx.doi.org/10.1109/ICRA48891.2023.10161317.

StepFun. Step 3.7 Flash. Model card, 2026. URL https://huggingface.co/stepfun-ai/Step-3.7-Flash.

Yiyou Sun, Xinyang Han, Weichen Zhang, Yuanbo Pang, Tianyu Wang, Yuhan Cao, Yixiao Huang, Chris Duroiu, Haoyun Zhang, Jeffrey Lin, et al. Agents' last exam, 2026. URL https://arxiv.org/abs/2606.05405.

Tencent Hy Team. Hy3. Model documentation, 2026a. URL https://github.com/Tencent-Hunyuan/Hy3.

Tencent Hy Team. Hy4 Preview. Model documentation, 2026b. URL https://github.com/Tencent-Hunyuan/Hy4-preview.

J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, Niall Williams, Yashas Lokesh, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 15032–15043. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/803f7c4c3ff61b71be53a0c803bfb57f-Paper.pdf.

Jordan K Terry, Benjamin Black, and Mario Jayakumar. MAgent. https://github.com/Farama-Foundation/MAgent2, 2020. GitHub repository.

The Terminal-Bench Team. Terminal-Bench 2.1, 2026. URL https://www.tbench.ai/news/terminal-bench-2-1.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, October 2012. doi: 10.1109/iros.2012.6386109. URL http://dx.doi.org/10.1109/IROS.2012.6386109.

Mark Towers, Ariel Kwiatkowski, John Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Kallinteris Andreas, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, J Terry, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-4916. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/d7ff1795e8527f6443371c3933bdb52b-Paper-Datasets_and_Benchmarks_Track.pdf.

Dweep Trivedi, Jesse Zhang, Shao-Hua Sun, and Joseph J Lim. Learning to synthesize programs as interpretable and generalizable policies. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 25146–25163. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/d37124c4c79f357cb02c655671a432fa-Paper.pdf.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Yongjie Wang, Xinyue Zhang, Kunhong Yao, Zhiwei Zeng, Kaisong Song, Jun Lin, and Zhiqi Shen. Search-time contamination in deep research agents: Measuring performance inflation in public benchmark evaluation, 2026a. URL https://arxiv.org/abs/2606.05241.

Zhun Wang, Nico Schiller, Hongwei Li, Srijiith Sesha Narayana, Milad Nasr, Nicholas Carlini, Xiangyu Qi, Eric Wallace, Elie Bursztein, Luca Invernizzi, Kurt Thomas, Yan Shoshitaishvili, Wenbo Guo, Jingxuan He, Thorsten Holz, and Dawn Song. Exploitgym: Can ai agents turn security vulnerabilities into real attacks?, 2026b. URL https://arxiv.org/abs/2605.11086.

ZHUN WANG, Tianneng Shi, Jingxuan He, Matthew Cai, Jialin Zhang, and Dawn Song. Cybergym: Evaluating ai agents'real-world cybersecurity capabilities at scale. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 123341–123386, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/c876a2935a90aff21874e14c07dc3e33-Paper-Conference.pdf.

Jiayi Weng. Learning beyond gradients. https://trinkle23897.github.io/learning-beyond-gradients/, May 2026. Blog post.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Sandha, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. Livebench: A challenging, contamination-limited llm benchmark. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 91595–91631, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/e4a46394ba5378b3f9a186a5b4c650d1-Paper-Conference.pdf.

xAI. Grok 4.20 System Card. Technical report, 2026a. URL https://data.x.ai/2026-04-07-grok-4-20-model-card.pdf.

xAI. Model Card: Grok 4.6. Technical report, 2026b. URL https://media.x.ai/v1/website/card-4p6-4cd2dc57.pdf.

Xiaomi MiMo Team. MiMo-V2.5. Model card, 2026. URL https://huggingface.co/XiaomiMiMo/MiMo-V2.5.

Yiming Xiong, Shengran Hu, and Jeff Clune. Learning to continually learn via meta-learning agentic memory designs, 2026. URL https://arxiv.org/abs/2602.07755.

John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. Programbench: Can language models rebuild programs from scratch?, 2026. URL https://arxiv.org/abs/2605.03546.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Z.ai. GLM-5.3, GLM-5.2, GLM-5.1, and GLM-5. Model documentation, 2026. URL https://github.com/zai-org/GLM-5.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/aa5f5e6eb6f613ec412f1d948dfa21a5-Paper-Conference.pdf.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents, 2026b. URL https://arxiv.org/abs/2603.19461.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642, March 2024. ISSN 2159-5399. doi: 10.1609/aaai.v38i17.29936. URL http://dx.doi.org/10.1609/aaai.v38i17.29936.

Lianmin Zheng, Jiacheng Yang, Han Cai, Ming Zhou, Weinan Zhang, Jun Wang, and Yong Yu. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018. ISSN 2159-5399. doi: 10.1609/aaai.v32i1.11371. URL http://dx.doi.org/10.1609/aaai.v32i1.11371.

Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, Fazl Barez, Rahul Gupta, Jwala Dhamala, Jacob Merizian, Mario Giulianelli, Harry Coppock, Cozmin Ududec, Antony Kellermann, Jasjeet Sekhon, Jacob Steinhardt, Sarah Schwettmann, Arvind Narayanan, Matei A Zaharia, Ion Stoica, Percy Liang, and Daniel Kang. Establishing best practices in building rigorous agentic benchmarks. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-5547. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/f316275b44ee2de533102913828a8107-Paper-Datasets_and_Benchmarks_Track.pdf.

A TASKS AND PLATFORM DETAILS

A.1 TASKS

Our benchmark integrates a wide range of 45 tasks spanning solo control, head-to-head play, and team interaction. We adapt and integrate their varied upstream implementations into a unified interface for policy authoring, execution, and evaluation. Table 3 presents a roster of tasks, grouped by interaction type and action space, with references and upstream implementations. Figure 10 shows a gallery of all tasks. We use this order of tasks throughout this paper.

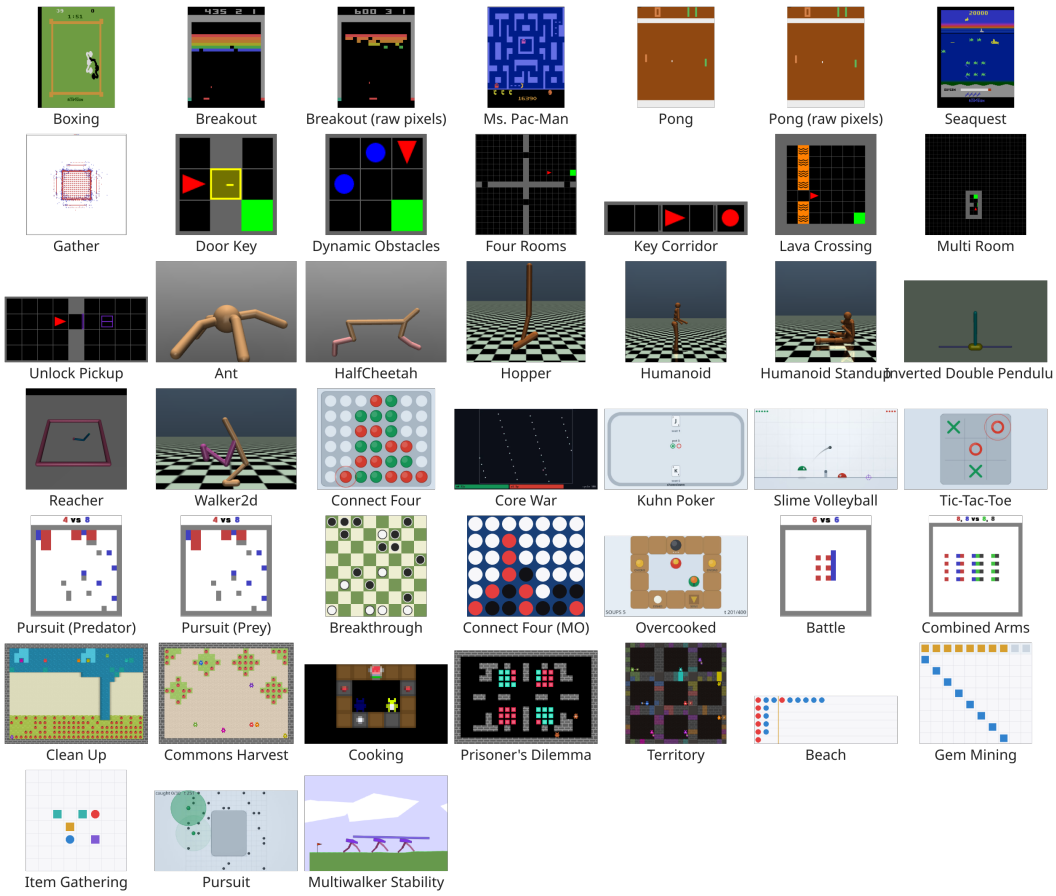


Figure 10: Task gallery with one random frame from a run of each task for illustration.

Table 3: Task roster.

Tasks	Reference	Upstream Implementation
Solo, discrete (15 tasks)		
Boxing / Breakout / Breakout (raw pixels) / Ms. Pac-Man / Pong / Pong (raw pixels) / Seaquest	Atari ALE (Bellemare et al., 2013), AtariARI (Anand et al., 2019)	`Farama-Foundation/Arcade-Learning-Environment`, `mila-iqia/atari-representation-learning`
Gather	MAgent / MAgent2 (Zheng et al., 2018; Terry et al., 2020)	`Farama-Foundation/MAgent2`
Door Key / Dynamic Obstacles / Four Rooms / Key Corridor / Lava Crossing / Multi Room / Unlock Pickup	MiniGrid (Chevalier-Boisvert et al., 2023)	`Farama-Foundation/Minigrid`

Continued on the next page

Table 3: Task roster (continued).

Tasks	Reference	Upstream Implementation
Solo, continuous (8 tasks)		
Ant / HalfCheetah / Hopper / Humanoid / Humanoid Standup / Inverted Double Pendulum / Reacher / Walker2d	MuJoCo (Todorov et al., 2012) through Gymnasium (Towers et al., 2025)	`google-deepmind/mujoco`, `Farama-Foundation/Gymnasium`
1v1, discrete (9 tasks)		
Connect Four	PettingZoo (Terry et al., 2021)	Adapted kernel with seat-relative observations. `Farama-Foundation/PettingZoo`
Core War	Core War (Dewdney, 1984)	MARS from `rodrigosetti/corewar`
Kuhn Poker	OpenSpiel (Lanctot et al., 2019)	Clean-room reimplementation checked against `google-deepmind/open_spiel`
Slime Volleyball	(Ha, 2020)	Ported physics with a net-collision fix. `hardmaru/slimevolleygym`
Tic-Tac-Toe	OpenSpiel (Lanctot et al., 2019)	Clean-room reimplementation checked against `google-deepmind/open_spiel`
Pursuit (Predator) / Pursuit (Prey)	MAgent / MAgent2 (Zheng et al., 2018; Terry et al., 2020)	`Farama-Foundation/MAgent2`
Breakthrough / Connect Four (MO)	MOMAland (Felten et al., 2026)	`Farama-Foundation/momaland`
Team, discrete (12 tasks)		
Overcooked	(Carroll et al., 2019)	Adapted from `HumanCompatibleAI/overcooked_ai`
Battle / Combined Arms	MAgent / MAgent2 (Zheng et al., 2018; Terry et al., 2020)	`Farama-Foundation/MAgent2`
Clean Up / Commons Harvest / Cooking / Prisoner's Dilemma / Territory	Melting Pot (Leibo et al., 2021; Agapiou et al., 2022)	`google-deepmind/meltingpot`
Beach / Gem Mining / Item Gathering	MOMAland (Felten et al., 2026)	`Farama-Foundation/momaland`
Pursuit	PettingZoo SISL (Gupta et al., 2017; Terry et al., 2021)	`Farama-Foundation/PettingZoo`
Team, continuous (1 task)		
Multiwalker Stability	MOMAland (Felten et al., 2026)	`Farama-Foundation/momaland`

A.2 Metrics and Measurement

Scoring For each task, scores follow the source implementations. After each run, we independently score its attempts using seeds separate from those used during LLM authoring. Stochastic tasks use average scores over multiple rollouts. These measurements select the champion (the attempt with the best mean score, choosing the earliest attempt in case of a tie) and provide its reported score. LLMs are asked to report scores for reference, but these scores do not determine selection.

Statistical model. We summarize each model's observed performance by pooling its run-level z scores and percentiles separately, using an empirical-Bayes one-way random-effects model. Each score is represented as an overall mean, a model-specific effect, and a residual. We assume normally distributed model effects and independent normal residuals with a common variance. Task-specific performance differences enter the residual variation, since the model has no separate task effects. Each run receives equal weight, including repeats of the same configuration, so tasks with more repeats contribute more. The estimates therefore summarize the observed mix of runs, without further adjustment for task coverage or difficulty.

Estimation and uncertainty. We estimate within-model and between-model variances using method-of-moments analysis of variance, which allows unequal numbers of runs per model. Negative estimates of between-model variance are set to zero. The resulting posterior mean is a weighted average of the model's own mean and the overall mean. Models with fewer runs are pulled more strongly toward the overall mean, a process called shrinkage. If every model has only one run, the two variance components cannot be separated. In that case, the implementation falls back to raw means and the observed spread, without shrinkage.

The 95% posterior interval is the posterior mean plus or minus 1.96 posterior standard deviations. It treats the estimated overall mean and variances as fixed and does not include uncertainty in those estimates. It is therefore a conditional empirical-Bayes interval, not a frequentist confidence interval. For rank uncertainty, we draw 4,000 samples independently from each model's fitted posterior and

rank the models in each draw. The central 95% of these ranks forms the rank interval. These intervals describe ranking uncertainty rather than provide pairwise significance tests.

Measuring with Failure Handling We discard and rerun runs affected by infrastructure failures, such as failed API calls or platform crashes. Model-caused failures, including budget exhaustion, stalls, or no scorable policy, are kept without reruns. If a run produces a measurable champion, we keep it even if the run later crashes or exhausts its budget. Otherwise, a run kept for ranking receives the task's random-policy score, estimated with five policy seeds, before standardization and pooling. This fallback generally ranks near the bottom, though a valid policy can occasionally perform worse than random.

Evaluation time limits. For 32 of the 45 tasks, evaluation time limits are set per simulator family using the 90th percentile of run times observed in initial exploratory runs. We use this percentile as an empirical cutoff. The remaining 13 tasks (Melting Pot, MOMAland, SISL Pursuit, and Core War tasks) have no evaluation time cap because their families lack usable timing data.

A.3 Isolation and Validation

Agent workspace and access levels. The agent's Bubblewrap sandbox exposes selected files, API credentials, and harness toolchains. A network proxy filters access, primarily permitting proxied LLM calls. We prefer Bubblewrap over Docker for its simplicity, but would consider Docker for future deployment at scale. Unshielded access is used when no opponent implementation needs protection and full access to the task is intended, as in single-player Atari tasks where game RAM is deliberately exposed. Shielded access withholds the opponent pool, prior policy collections, repository history, and run records, and, for Atari, the ROM and emulator. The shielded level adds the sandbox, the scrubbed environment and the proxy. Both levels use the same tools and the same policy submission and scoring protocol. Both levels retain the same baseline protections for network access, host filesystem access, and process isolation, with the same permitted tools and policy submission and execution protocols.

Static import checks and limits. Before executing policy code, the platform parses its source into an abstract syntax tree and checks every `import` and `from ... import ...` statement. Only imports with full dotted paths to allowed modules (NumPy, the standard library identified by `sys.stdlib_module_names`, and `core.rng`) pass the check. The scan checks import statements, not every route to code execution. `__import__`, `importlib`, generated `exec` code, and subprocess access through allowed standard-library modules can pass it. This is not a complete security boundary, but it serves the useful purpose of forcing shortcut behaviours into non-standard code that can be identified during inspection.

Execution protections. A separate Bubblewrap sandbox unshares all namespaces, disables networking, clears the environment, and permits only allowlisted read-only mounts and a size-capped temporary filesystem. Process, address-space, and execution-time limits constrain resource use. These runtime restrictions limit what dynamically loaded code can access.

Inspection findings. The champion listings examined in the ablation exhibits and policy spotlights use ordinary NumPy and standard-library code. Observed shortcuts involved unshielded agents copying accessible opponent weights or a trained controller (Appendix C). These were uses of deliberately exposed resources, not observed bypasses of the shielded sandbox. These shortcuts were observed only in runs with shielding explicitly disabled for the ablation.

A.4 Agent Instruction

A run receives a single information package containing lessons (reusable guidance from outside the current run), a task brief (the task interface and evaluation calls), and generic session instructions (requirements for attempt artifacts, tool use, and stopping conditions), followed by a kickoff line. The purpose is to provide the agent with enough context to solve the problem. We illustrate these components below using the Slime Volleyball task, with run and repository paths replaced by placeholders. This package exposes the target seed block and specifies the evaluation procedure for the

agent to use, which is needed to complete the task. However, reported scores follow the independent evaluation protocol and champion-selection rules already described in Appendix A.2.

```
# Session instructions (the thin-tick contract)

You are a headless agentic session in a LEARNING RUN. Your
  working directory is
the run directory

    <run directory>

That directory is your workshop and the only place you write.
   It is NOT inside
the repository: when a tool wants an absolute path, take it
  from the line above,
never build one under the repository root. Everything above
  this section is your
context: the LESSONS (accumulated, validated knowledge; use
  it) and the
ENVIRONMENT BRIEF (given facts about the environment,
  including how to evaluate
a policy and read the results).

Your goal: write and improve a code-as-policy heuristic for
  the environment
`slimevolley`, as strong as you can make it within budget.
  Strive for better
performance ruthlessly: the brief states the environment's
  known ceiling and
reference bars where they exist, and the ceiling, not your
  first improvement
over a baseline, is the aim. Where the brief states no
  ceiling, do NOT assess
or derive one yourself; the aim is then always better than
  your current best.

## The contract (mechanical; the harness relies on it)

- ATTEMPTS: an attempt is one HYPOTHESIS FAMILY, not one
  configuration: a
  mechanism change plus the parameter sweep that dials it in.
   Attempt
  directories are `attempt_01/`, `attempt_02/`, ... (zero-
  padded, in order),
  always directly under the run directory. A policy not at
  `<run directory>/attempt_NN/policy.py` is invisible to
  measurement until close-time
  adoption; do not rely on it.
  Parameterize your policy and evaluate as many configs
  inside an attempt as
  the idea needs; keep a trial ledger in the attempt dir (`
  trials.jsonl`: one
  line per config evaluated, config + score). The ceremony is
   per FAMILY:
  create the directory first (`mkdir attempt_NN` is your
  atomic claim), then
  write into it:
  - `analysis.md`: written BEFORE you write the policy. Your
  failure analysis of
    the previous attempt (what actually went wrong, with
  evidence from real
    play), and what this attempt changes and why.
  - `policy.py`: the family's best config, exposing `
  make_policy(seat)` which
    returns `act(obs) -> action`.
  - `self_report.json`: written after you evaluate. The shape
  :
    `{"kept": bool, "rationale": str, "built_on": "attempt_NN
  " | null,
    "self_score": number | null}`. State your keep/revert
  decision and its
    rationale honestly. Self-reports are provenance only: we
  re-measure every
    attempt ourselves after the run; nothing you report
  enters measurement.
- CANDIDATES: when you learn something durable, propose it as
   a lesson: one
  markdown file in `candidates/`. A candidate MUST cite the
  attempt diff and the
  measured delta it rests on, and must read as a testable
  prediction, never a
  narrative. Zero candidates is a valid, honest outcome.
- TOOLS: you are invited to build any tools you want inside
  the run directory
  (trajectory analyzers, search harnesses, population
  machinery, whatever raises
  your ceiling) and use them within budget.
- FILE WRITES: write EVERY file (contract artifacts like `
  analysis.md`,
  `self_report.json`, `trials.jsonl`, `candidates/*`, AND
  policies, probe
  scripts, scratch) with the file-writing tool, never shell
  heredocs: the shell
  guard blocks command substitution, a blocked command can
  return SUCCESS with
  a "Blocked: ..." message as its output, and a heredoc write
   can HANG your
  shell for many minutes, which reads as silence and gets the
   session killed.
  If any write is refused, retry with the file-writing tool;
  never leave a
  contract artifact unwritten.
- The repository at `<repository>` is READ-ONLY reference (
  environment code, eval
  seams). Never modify it; never write outside `<run
  directory>`. The run directory
  lives outside the checkout, never under it: a path
  beginning `<repository>` is not
  yours, however plausible it looks. If a shell command must
  run from the
  repository (a project-scoped tool), pass that as the
  command's own working
  directory and keep writing files by their `<run directory>`
   path.
- NO ENV SELF-IMPORT: `slimevolley` is not on the env-as-
  world-model allowlist. Your policy must never import the
  environment module (or its underlying engine/emulator) to
  use as its own simulator or world model; read the code
  freely, but the policy acts from `obs` alone.
- Web access is disabled. Work from your context and the
  repository.

## Budget and persistence

About 12 attempts, 100000000 tokens, ~30 minutes of
wall clock. The harness enforces all three as CEILINGS; a
  session that goes
silent (no output, no file writes) is killed within minutes.

Budget is fuel, not a deadline: whatever you leave unspent
  buys nothing, and a
good score is a reason to push harder, never a reason to stop
  . Do not conclude
while budget remains unless you can write down one of:
- PLATEAU: no improvement over your last ~5 attempts AND you
  cannot name an
  untested hypothesis worth an attempt. Changing direction is
   what remaining
  budget is FOR: exhaust direction changes before claiming a
  plateau.
- RAIL: a budget rail (attempts, tokens, wall) is genuinely
  near.
A ceiling, reached or unknown, is NEVER a stopping reason: at
   the brief's
stated ceiling, verified attainment and robustness work (
  internal-seed sweeps,
simplification that holds the score) replace score chasing;
  where no ceiling is
stated, do not derive one, and keep aiming above your current
   best. A new best
is never a finish line: it OBLIGATES further attempts (what
  does the champion
still fail at? where do the remaining points live?). Treat
  stopping before half
of 12 attempts as an early stop that needs a strong written
case.

Track your spend: state attempts-used/target and tokens-used/
  budget in each
analysis.md. Your FINAL message must restate those numbers
  and argue the
stopping case above against your own measurements; "the
  result looks good" is
not a stopping reason. Before you finish, also make sure
  every attempt
directory is complete and your candidates are written;
  completeness is
necessary to stop, never sufficient.
```

Session instructions for the Slime Volleyball run. Run-specific paths are placeholders.

```
# Environment brief: slimevolley

GENERATED from repo state by `learn.brief` (do not hand-edit;
   regenerate). These are GIVEN FACTS about the environment,
  not strategy. What to DO is in the lessons above; this
```

```
  section states only what the environment IS and how to
  measure a policy on it.

- env id: `slimevolley`
- sides: [1, 1], turn order: simultaneous, reward: per_seat

## Observation

A 12-vector (`f32`) per acting seat. mirrored per seat: [me x
  ,y,vx,vy, ball x,y,vx,vy, opp x,y,vx,vy] / 10.0
Because it is mirrored to each seat's own frame, one policy
  plays either side identically.

## Actions

`multi_binary`, 3 bits: `['forward', 'backward', 'jump']` (no
  -op `[0, 0, 0]`). `act(obs)` returns a length-3 0/1 vector;
   a bit is on when its named input is pressed.

## Reward and scoring

Reward is `per_seat` and zero-sum: one seat's gain is the
  other's loss. A rally won scores +1 to the winner and -1 to
   the loser.
A match ends when either slime loses all 5 lives, or after
  3000 simultaneous steps (truncation), whichever comes first
  .
An episode's SCORE is the per-episode net reward of the
  protagonist seat (sum of its step rewards): positive means
  your seat won more rallies than it lost.
KNOWN CEILING: 5 per episode (theoretical: win all 5 of the
  opponent's lives and lose none. Separately, as a measured
  headroom fact: a prior heuristic policy reached +0.26 mean
  net reward vs this same rnn on the held-out block (zero-
  trust, 1000 episodes), so any claim that a reactive single-
  frame heuristic cannot exceed a small margin over the rnn
  is refuted by that existence proof). It is a reference for
  how much headroom remains, never a stopping rule.

## The target (what WE measure, zero-trust)

The seed block and opponent pool below are the TARGET: they
  are visible, and beating the pool is the point. You iterate
   against them freely, but the real, uniform measurement is
  OURS and runs once after the run; nothing you self-report
  enters it.

- opponent pool (1), you play seat 0:
  - `slimevolley__reference__rnn` (reference): slimevolley
  baseline (tier 3, authored): the built-in 120-param RNN
  opponent.
- target seed block: base 99991, stride 101 (disjoint from
  the training block base 4242, stride 13)
- episodes: 100 paired episodes per opponent; read MEAN plus
  a confidence interval, per opponent and pooled. A one-
  episode swing is noise.
- the claims this run is judged on (resolution = the smallest
   margin that counts):
  - beats the rnn: measured against `
  slimevolley__reference__rnn`, resolution 0.1

## Fast eval

Score a policy through the SHIPPED eval seam (the same one
  that measures you) with the generic, decision-free helper.
  From your run dir:

```
uv run python -m runs.eval --env slimevolley --policy
  attempt_03/policy.py \
    --opponents slimevolley__reference__rnn \
    --episodes 30 --seed-base 4242 --stride 13
```

It prints mean + CI per opponent and pooled. Iterate on the
  training seeds (`--seed-base 4242 --stride 13`, above); to
  preview the target, pass `--seed-base 99991 --stride 101`.
  Add `--record-tapes DIR` to keep the per-episode JSONL and
  replay a single episode.
```

Slime Volleyball task brief, including its interface, target, and evaluation command.

```
---
scope: global
provenance: authored-seed
---
# Read real play and write a failure analysis before you edit

- Before changing the policy, watch or replay ACTUAL episodes
   and write down why it loses: the
  concrete failure mode, not a guess from the code.
- Every edit is a hypothesis aimed at ONE named failure.
  State the prediction before you run it
  ("fixing X should raise the score against opponent Y").
- After measuring, compare the outcome to the prediction. A
  surprise is information; record it and
  let it steer the next edit.
- Do not stack blind edits. A change you cannot attribute to
  a failure you understood is not
  progress, even if the number went up.
```

Example global lesson requiring evidence from actual play before policy revision.

```
---
scope: env-family:competitive
provenance:
  corpus: v1
  sources:
    - "multi_agent/instructions/running_with_scissors.md (
  scout the opponent's apparent type, then counter it)"
---
# Identify the opponent's type before committing to a counter

Where the winning move depends on what the opponent is doing,
   spend early actions to SCOUT the
opponent's apparent type or intent, then commit the counter.
  Committing to a counter before
identifying the target loses to whichever type you guessed
  wrong about.

Test: a policy that scouts then counters beats one that
  commits a fixed pick, because the fixed pick
is exploited by the type it does not beat, while the scout
  adapts.
```

Example lesson scoped to competitive tasks.

B More Details on Benchmark Results

B.1 Model Roster

Model roster, harnesses, and budget. Table 4 lists the 47 models in the primary comparison, their developers, and source references. All harnesses run headlessly and use subscriptions where available. Claude Code[1] serves five Anthropic models at maximum reasoning. Codex[2] serves four OpenAI models at high reasoning. OpenCode[3] serves the remaining 38 through OpenRouter[4] at high reasoning with pinned endpoints.

Table 4: Model roster of 47 models.

Developer	Model	Reference
Alibaba	Qwen3-Max	(Qwen Team, 2025)
Alibaba	Qwen3.7-Max	(Qwen Team, 2026b)
Alibaba	Qwen3.8-2.4T-A95B	(Qwen Team, 2026c), model card
Alibaba	Qwen3.8-Flash-Next[a]	(Qwen Team, 2026a)
Alibaba	Qwen3.8-Max	(Qwen Team, 2026c)
Amazon	Nova Premier	(Amazon Artificial General Intelligence, 2025)
Anthropic	Claude Fable 5	(Anthropic, 2026a)
Anthropic	Claude Fable 5.1	(Anthropic, 2026b)
Anthropic	Claude Opus 4.8	(Anthropic, 2026c)
Anthropic	Claude Opus 5	(Anthropic, 2026d)
Anthropic	Claude Sonnet 5	(Anthropic, 2026e)
ByteDance	Seed 2.0 Code[b]	(ByteDance Seed, 2026a)
ByteDance	Seed 2.1 Turbo	(ByteDance Seed, 2026b)
DeepSeek	DeepSeek V4 Flash (0731)	(DeepSeek-AI, 2026a), release notes
DeepSeek	DeepSeek V4 Pro	(DeepSeek-AI, 2026a)
DeepSeek	DeepSeek V4 Pro (0813)	(DeepSeek-AI, 2026a), release notes
DeepSeek	DeepSeek V4.1 Flash	(DeepSeek-AI, 2026b)
Google	Gemini 3.1 Pro	(Google DeepMind, 2026a)
Google	Gemini 3.5 Flash	(Google DeepMind, 2026b)
Google	Gemini 3.5 Flash-Lite	(Google DeepMind, 2026c)
Google	Gemini 3.6 Flash	(Google DeepMind, 2026d)
Google	Gemini 3.7 Flash	(Google DeepMind, 2026e)
Inception	Mercury 2	(Ermon, 2026)
inclusionAI	Ling 3.0 Flash	(inclusionAI, 2026)
Meta	Llama 3.1 8B	(Grattafiori et al., 2024)
Meta	Llama 4 Maverick	(Meta, 2025)
Meta	Muse Glimmer 30B	(Meta Superintelligence Lab, 2026)
Meta	Muse Spark 1.2	(Meta Superintelligence Labs, 2026)
MiniMax	MiniMax M3	(MiniMax, 2026)
Mistral AI	Mistral Large 3	(Mistral AI, 2025)
Moonshot AI	Kimi K2	(Kimi Team, 2025)
Moonshot AI	Kimi K2.6	(Moonshot AI, 2026)
Moonshot AI	Kimi K3	(Kimi Team, 2026)
NVIDIA	Nemotron 3 Super	(NVIDIA, 2026)
OpenAI	GPT-5.6 Luna	(OpenAI, 2026a)
OpenAI	GPT-5.6 Sol	(OpenAI, 2026a)
OpenAI	GPT-5.6 Terra	(OpenAI, 2026a)
OpenAI	GPT-6 Astra	(OpenAI, 2026b)
StepFun	Step 3.7 Flash	(StepFun, 2026)
Tencent	Hy3	(Tencent Hy Team, 2026a)
Tencent	Hy4 Preview	(Tencent Hy Team, 2026b)
xAI	Grok 4.6	(xAI, 2026b)
xAI	Grok 4.20	(xAI, 2026a)
Xiaomi	MiMo V2.5	(Xiaomi MiMo Team, 2026)
Z.ai	GLM-5.2	(GLM-5 Team, 2026; Z.ai, 2026)
Z.ai	GLM-5.3	(GLM-5 Team, 2026; Z.ai, 2026)
Z.ai	GLM-5.3 Flash	(GLM-5 Team, 2026; Z.ai, 2026)

[a]We evaluate the hosted Qwen3.8-Flash endpoint on OpenRouter, based on Qwen3.8-Flash-Next.
[b]We use the OpenRouter Seed-2.0-Code variant.

We pair models with harnesses that we found workable for the tasks to keep the integration effort manageable, since each harness needs to be integrated individually. We note that the rapidly growing

[1]https://claude.com/product/claude-code
[2]https://openai.com/codex/
[3]https://opencode.ai/
[4]https://openrouter.ai/

harness ecosystem also includes Gemini CLI[5], Qwen Code[6], DeepSeek Harness[7], and Cursor CLI[8], as well as more universal harnesses such as Pi[9], Cline CLI[10], and Goose[11]. We leave comprehensive integration and comparison of harnesses to future work.

Each run has a 30-minute budget and a 12-attempt target, a 15-minute liveness watchdog, a memory cap, and a token cap set high enough that wall-clock time binds first.

Pricing. Costs use recorded token counts and officially listed rates for Claude and GPT models. Other recorded costs are vendor-reported through OpenRouter, except for older GPT-5.6 Sol and Claude Fable 5 runs whose pricing basis was not recorded. Unpriced runs are excluded from mean cost, and models with no priced runs are absent from Figure 2.

Coverage and core selection. We selected the 19 core models shown in Figure 11 based on their expected ability to produce reasonable policies. We scheduled each core model for three runs per task on 13 representative tasks spanning the task categories, and one run per task on the remaining 32 tasks. Other models were scheduled for one run per task on all tasks. These decisions were made to prioritize important models and tasks while limiting the total budget.

B.2 Comparisons and Findings

Comparisons. Table 5 uses the same metrics to extend the highlight results in Table 1 in Section 4.1 to all 47 models. Figure 11 compares the core models on shared tasks, exposing task-specific strengths and weaknesses, and Figure 12 compares native scores across tasks.

Findings. Table 5 shows that the benchmark separates stronger and weaker models with a wide spread of posterior z scores and percentiles. Beyond the results for strong models presented in the main text (Table 1), the benchmark also reveals interesting traits of weaker models. (1) Weaker models have higher "First Attempt" rates (meaning the first attempt produces the best policy among all attempts), suggesting that further improvements are less likely even when additional attempts are provided. (2) Some weak models even struggle to produce a measurable policy. We observe a high first-try failure rate with low budget exhaustion in raw-pixel Pong and the reverse in Territory.

Figures 11 and 12 show that task difficulty distinguishes models while leaving headroom for improvement. For example, Pong and Door Key cluster near their ceilings, whereas Ant and Hopper separate models more widely. Humanoid remains far below its strong-agent reference (Section 4.4). This mixture avoids uniformly easy or hard tasks that would offer little discrimination.

Table 5: All 47 models, ordered by posterior z mean, using the columns of Table 1. Intervals are 95% empirical-Bayes posterior intervals. A dash (-) denotes no measured champion for Best, an undefined attempt rate, or unavailable cost. A value of 0% is a defined rate with no qualifying outcomes.

	Posterior z			Percentile			Attempt Rate		Per Full Run	
Model	Mean	95% I	Best	Mean	95% I	Best	First	All	Cost	Time
GPT-6 Astra	+0.846	[+0.66, +1.03]	+6.043	81	[76, 86]	100	2%	100%	$11.20	24.8
Claude Fable 5.1	+0.658	[+0.48, +0.84]	+3.787	70	[65, 75]	100	29%	100%	$5.48	16.5
GPT-5.6 Sol	+0.499	[+0.32, +0.68]	+2.188	71	[66, 76]	100	3%	100%	$6.34	24.8
Claude Fable 5	+0.489	[+0.31, +0.67]	+1.583	68	[63, 73]	98	19%	100%	$11.62	21.0
DeepSeek V4.1 Flash	+0.436	[+0.26, +0.62]	+3.851	62	[57, 67]	100	41%	96%	$0.12	14.5
Claude Opus 5	+0.413	[+0.23, +0.60]	+2.879	65	[60, 70]	100	12%	100%	$5.21	21.5
Muse Spark 1.2	+0.378	[−0.09, +0.85]	+1.093	48	[35, 62]	84	11%	100%	$1.08	10.5
GPT-5.6 Luna	+0.340	[+0.16, +0.52]	+2.016	58	[53, 63]	92	10%	99%	$0.09	18.0

Continued on next page

[5] https://github.com/google-gemini/gemini-cli
[6] https://github.com/QwenLM/qwen-code
[7] https://github.com/deepseek-ai/deepseek-harness
[8] https://cursor.com/cli
[9] https://github.com/earendil-works/pi
[10] https://cline.bot/cli
[11] https://goose-docs.ai/

Table 5 (continued)

Model	Posterior z			Percentile			Attempt Rate		Per Full Run	
	Mean	95% I	Best	Mean	95% I	Best	First	All	Cost	Time
Grok 4.6	+0.331	[+0.14, +0.52]	+1.377	63	[58, 68]	94	7%	99%	$5.51	22.3
Gemini 3.5 Flash-Lite	+0.265	[−0.21, +0.74]	+1.093	49	[36, 62]	84	22%	97%	$0.25	2.3
Claude Opus 4.8	+0.235	[+0.05, +0.42]	+1.877	55	[50, 60]	94	38%	99%	$1.21	28.8
GLM-5.2	+0.223	[−0.25, +0.69]	+0.608	52	[39, 65]	67	44%	100%	$2.50	24.7
Claude Sonnet 5	+0.209	[−0.26, +0.68]	+0.608	51	[38, 64]	70	56%	100%	$1.79	22.5
Gemini 3.6 Flash	+0.201	[+0.01, +0.39]	+1.887	55	[50, 60]	100	31%	99%	$1.48	16.2
GPT-5.6 Terra	+0.190	[+0.01, +0.37]	+1.344	56	[51, 61]	97	9%	99%	$1.57	13.6
Seed 2.1 Turbo	+0.169	[−0.30, +0.64]	+0.608	45	[32, 58]	67	33%	100%	$0.18	23.9
MiniMax M3	+0.131	[−0.34, +0.60]	+1.119	54	[40, 67]	87	33%	98%	$1.48	28.4
Qwen3-Max	+0.090	[−0.38, +0.56]	+0.599	38	[25, 51]	70	33%	100%	$6.09	23.1
GLM-5.3 Flash	+0.085	[−0.39, +0.56]	+0.608	50	[36, 63]	71	62%	100%	$0.09	29.2
Gemini 3.5 Flash	+0.085	[−0.10, +0.27]	+3.013	55	[50, 60]	100	44%	99%	$3.35	12.6
Grok 4.20	+0.020	[−0.45, +0.49]	+0.608	40	[27, 53]	67	44%	100%	$0.70	3.8
GLM-5.3	+0.010	[−0.17, +0.19]	+2.368	50	[44, 54]	100	51%	97%	$1.78	23.1
Gemini 3.1 Pro	−0.015	[−0.20, +0.17]	+1.074	45	[40, 50]	90	25%	98%	$3.08	18.3
Kimi K3	−0.093	[−0.27, +0.09]	+1.247	45	[40, 50]	90	46%	99%	$1.69	25.0
Qwen3.8-2.4T-A95B	−0.100	[−0.57, +0.37]	+0.599	37	[24, 50]	66	71%	100%	$1.88	30.0
DeepSeek V4 Pro (0813)	−0.231	[−0.70, +0.24]	+0.608	44	[31, 57]	67	43%	100%	$0.70	23.4
DeepSeek V4 Flash (0731)	−0.272	[−0.46, −0.09]	+1.093	41	[36, 46]	99	63%	93%	$0.07	21.9
Hy3	−0.282	[−0.78, +0.21]	+0.479	35	[21, 48]	62	100%	100%	$0.17	16.3
Qwen3.7-Max	−0.358	[−0.54, −0.17]	+1.417	37	[32, 42]	100	59%	97%	$1.31	21.8
Kimi K2	−0.507	[−0.98, −0.04]	+0.479	37	[24, 50]	67	38%	100%	$0.41	9.7
Step 3.7 Flash	−0.617	[−1.09, −0.15]	+0.479	29	[16, 42]	62	100%	100%	$0.15	9.8
Muse Glimmer 30B	−0.743	[−1.21, −0.27]	+0.479	25	[12, 38]	62	60%	100%	$0.09	4.4
DeepSeek V4 Pro	−0.744	[−0.93, −0.56]	+1.344	26	[21, 31]	75	62%	96%	$0.54	16.9
Qwen3.8-Max	−0.748	[−0.93, −0.57]	+0.833	24	[19, 28]	83	67%	98%	$2.13	27.0
Ling 3.0 Flash	−0.760	[−1.23, −0.29]	+0.428	23	[10, 36]	27	57%	88%	$0.03	9.1
Seed 2.0 Code	−0.785	[−1.26, −0.31]	+0.608	28	[15, 41]	67	67%	100%	$0.32	13.5
MiMo V2.5	−0.928	[−1.40, −0.46]	+0.564	27	[14, 40]	70	33%	100%	$0.10	11.3
Kimi K2.6	−0.967	[−1.44, −0.50]	+0.234	22	[9, 35]	40	80%	100%	$0.34	17.0
Gemini 3.7 Flash	−1.018	[−1.49, −0.55]	+0.608	28	[15, 41]	69	33%	100%	$0.48	4.9
Mercury 2	−1.075	[−1.55, −0.60]	+0.438	22	[9, 35]	33	86%	94%	$0.07	4.1
Hy4 Preview	−1.181	[−1.65, −0.71]	−0.296	19	[6, 32]	43	100%	100%	$0.38	16.3
Qwen3.8-Flash-Next	−1.450	[−1.92, −0.98]	−0.304	16	[3, 30]	42	0%	100%	$0.02	17.1
Llama 3.1 8B	−1.511	[−1.98, −1.04]	-	13	[0, 26]	-	-	-	$0.00	0.1
Llama 4 Maverick	−1.511	[−1.98, −1.04]	-	13	[0, 26]	-	-	-	$0.02	1.0
Mistral Large 3	−1.511	[−1.98, −1.04]	-	13	[0, 26]	-	-	-	$0.11	3.7
Nova Premier	−1.511	[−1.98, −1.04]	-	13	[0, 26]	-	-	0%	$3.56	9.5
Nemotron 3 Super	−1.761	[−2.23, −1.29]	−1.695	16	[3, 29]	17	100%	80%	$0.01	19.2

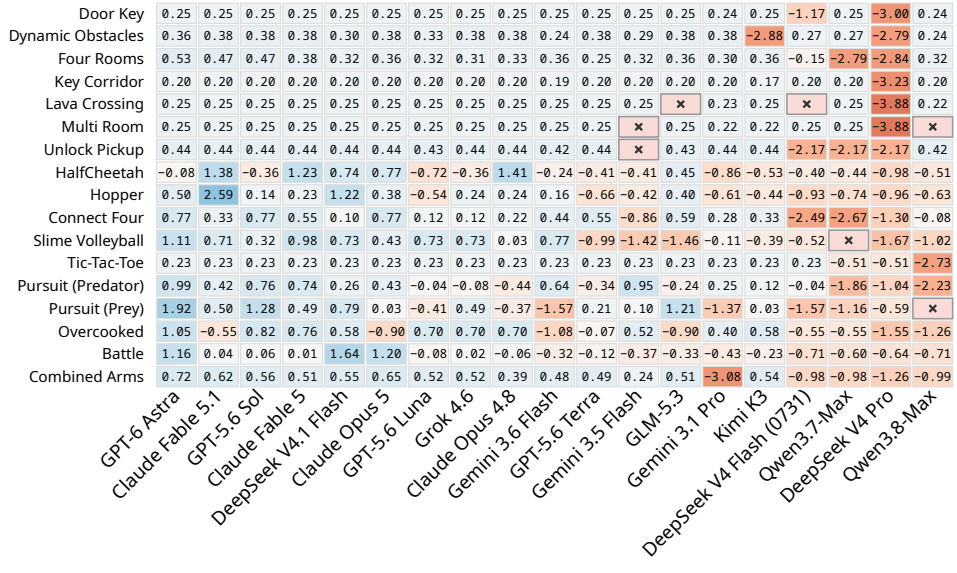


Figure 11: The 19 core models on core tasks, restandardized within this subset. We use blue and red to indicate above-average and below-average scores. Crosses mark missing measurements.

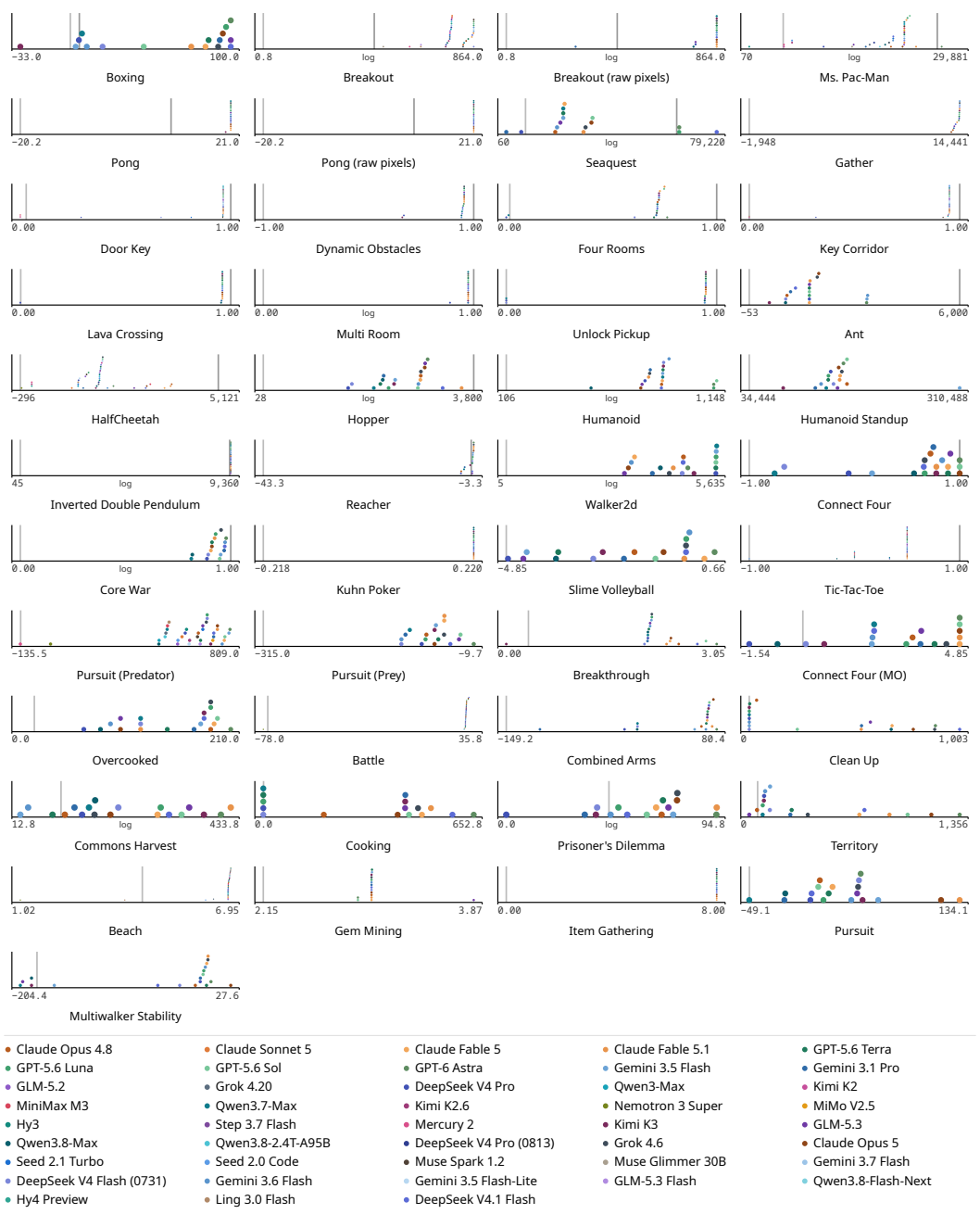


Figure 12: Scores across tasks. Coloured dots show model means. Reference lines mark random floors and solved thresholds where available.

C Details of Shielding Experiments

We provide a closer look at three cases of solution reuse from the deliberate shielding ablation in Section 4.3. All runs examined here are included in the paired comparison in Figure 4, but these runs explicitly remove shielding and thus do not count towards the benchmark ranking.

C.1 Slime Volleyball: reusing the opponent's policy with Qwen3.8-Max.

Unshielded. The agent's second file read accesses the baseline RNN opponent's source code, including its network weights. The same weight values appear in the scored champion policy, confirming the copying. The agent's reflection summary notes this reuse and describes simulating the physics and the mirrored opponent together, then searching for an action at each contact.

Second file read

```
[read] platform/refs/slimevolley/slimevolley__reference__rnn.py
15: # 7x15 weights (row-major) + 7 biases, verbatim from slimevolleygym's BaselinePolicy
16: # (David Ha's slime volleyball, otoro.net/slimevolley: the origin of these weights).
17: _W = [
18:     7.5719, 4.4285, 2.2716, -0.3598, -7.8189, -2.5422, -3.2034, 0.3935, 1.2202, -0.49, ...
```

Weights appearing in the champion policy

```
267: # ---------------- RNN opponent mirror (verbatim from policies/slimevolley__baseline__rnn.py)
----------------
268: RNN_W = [
269:     7.5719, 4.4285, 2.2716, -0.3598, -7.8189, -2.5422, -3.2034, 0.3935, 1.2202, -0.49, ...
```

Agent's reflection summary

```
## What this attempt changes and why
The shelf carries a measured ceiling-achiever for exactly this env/opponent:
...
I adopt it (not blind copy: I read its full logic) and RE-VERIFY zero-trust on this box.

THE MECHANISM (why it wins): the rnn opponent is deterministic and its obs is an exact mirror of
mine, so I can track its 7-unit hidden state step-for-step. The physics is deterministic with
public constants, re-derived as plain arithmetic (no env import).
...
contact scripts (stand-x offset x jump lead x air direction x contact time) through exact rollouts
against the tracked rnn, committing by a value ladder: confirmed KILL > safe line > emergency save
> reactive defense.
```

Shielded. The agent's search for the opponent policy returns no results. It instead produces a policy that predicts the ball's landing point without modelling the opponent.

Shielded champion policy

```
"""attempt_01: ballistic landing-point interceptor (seed baseline).

Mirror-space control (obs*10): net at x=0, my court x>0, wall x=24, forward = -x.
Predict the ball's first ground contact ballistically (gravity, wall/ceiling reflection,
crude solid-net reflection); stand OFFSET wall-side of it so the contact normal on my
net-side flank deflects the ball net-ward; jump for balls above ground reach.
"""
```

C.2 Slime Volleyball: reimplementing the opponent with Claude Fable 5.1.

Unshielded. The run records three reads of a prior champion and six reads of a reference policy. The unshielded champion policy is, as noted in its docstring, a bit-faithful reimplementation of the task physics with copied opponent weights, tracking the opponent's recurrent state to search for scoring contacts.

Unshielded champion policy

```
"""slimevolley: exact-model planner (attempt_02: finer contact grid).

Acts from obs alone. Carries a bit-faithful re-implementation of the kernel physics (read from
```

```
the env source) plus a copy of the opponent RNN's public weights, tracks the true world state and
the opponent's recurrent state from the obs stream, and searches contact scripts by rollout,
committing to the one that grounds the ball on the opponent's side soonest.
"""
...
# --- opponent RNN (public weights of slimevolley__reference__rnn) ---
```

Shielded. The shielded champion policy plans contacts using the ball's dynamics without simulating the opponent's policy.

Shielded champion policy

```
"""slimevolley heuristic: exact discrete-contact planner over my standing position.

Frame: obs is mirrored so I face right: my x>0, net at x=0, forward = -x. All obs fields are /10.
Model (kernel rules reimplemented; acts from obs alone): ball falls under g=-29.4, speed cap 22.5, reflects
...
```

C.3 HOPPER: COPYING A TRAINED CONTROLLER WITH KIMI-K3 AND ANOTHER MODEL'S POLICY WITH GLM-5.3.

Unshielded. This is an interesting finding. The unshielded Kimi-K3 run produces a champion by copying an available MLP controller trained with Augmented Random Search (ARS) (Mania et al., 2018). This shelf champion policy is then copied by GLM-5.3 in its first attempt, which is clearly noted in the policy's docstring. Scores in later attempts match the source policy's at the reported precision, and the model identifies the copied Kimi-K3 policy as an off-the-shelf implementation. Note that this ARS controller is a simple controller distinct from the PPO and SAC baselines we train for the comparison in Section 4.4.

Agent transcription statement

```
# attempt_01: transcribe + re-derive the shelf champion (kimi-k3 a04, ARS mlp16)
...
I transcribe the kimi champion byte-identical (excluding its provenance header) into
attempt_01 and re-measure it through the shipped eval seam ...
```

Copied controller in champion policy

```
"""mujoco_hopper trained champion: attempt 4, the run's selected champion ...
- run: 20260717-064450-103232e6
- pooled score: +3947.5585 (95% CI +3947.0503 .. +3948.0668), n=1000
...
Byte-identical to the assessed attempt policy below.
"""

"""ARS-trained mlp16 policy for mujoco_hopper (checkpoint val_mean=3946.9..., obs-norm baked in)."""
```

Shielded. Shielding hides prior trained policies. The available hopping controller is the platform's video-recording driver, documented as untuned and hand-written. It is explicitly described as an untrained energy-shaping controller.

Controller

```
A hand-written energy-shaping controller, not a trained policy. Its constants are
hand-picked and UNTUNED: the sinusoidal leg/foot pump guarantees a visibly deliberate
hop cycle regardless of tuning ...
```

D More Details of Analysis

D.1 Slime Volleyball Policy Source

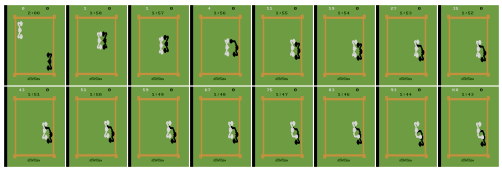
Code 2 gives the complete implementation of the shielded champion shown in Figure 6, including all helper functions and omitting only the exported provenance header. The policy forecasts the ball's motion and evaluates candidate contacts as either attacking shots or setups for a later contact.

```python
import math

HOME = 8.0
CLEARANCE = 5.0

def flight(x, y, vx, vy):
    path = []
    for i in range(100):
        vy -= 0.98
        speed = math.sqrt(vx*vx+vy*vy)
        if speed > 22.5:
            vx *= 22.5/speed
            vy *= 22.5/speed
        x += vx/30
        y += vy/30
        if abs(x) > 23.5:
            x = max(-23.4966667, min(23.4966667, x))
            vx = -vx
        path.append((x,y,vx,vy))
        if y < 2:
            break
    return path

def shot(bx, by, vx, vy, ox):
    speed = math.sqrt(vx*vx+vy*vy)
    if speed > 22.5:
        vx *= 22.5/speed
        vy *= 22.5/speed
    if vx >= -0.1:
        return -25 + min(10, vy)*0.2 - vx*0.1
    t = -bx/vx
    ny = by+vy*t-14.7*t*t
    if ny < CLEARANCE:
        return -20 + ny
    land_t = (vy+math.sqrt(max(0,vy*vy+58.8*(by-2))))/29.4
    land = bx+vx*land_t
    return 10 + min(10,abs(land+ox))*0.4 - max(0,ny-9)*0.1

SET_VALUE = 14

def setup_score(bx,by,vx,vy,sx):
    if bx<5 or vy<3:
        return -100
    for i,(px,py,ux,uy) in enumerate(flight(bx,by,vx,vy)):
        if px<1.5:
            return -100
        if py<3.2 and uy<0:
            if i>12 and 2.5<px<min(10,bx-2) and abs(px-sx)<17.5*(i+1)/30:
                return SET_VALUE-0.3*px-0.015*i
            return -100
    return -100

def make_policy(seat):
    armed = False
    def act(obs):
        nonlocal armed
        x,y,vx,vy,bx,by,bvx,bvy,ox,oy,ovx,ovy = [float(v)*10 for v in obs]
        if bx<0 or (abs(bx)<0.001 and abs(by-12)<0.001):
            armed = False
        path = flight(bx,by,bvx,bvy)
        lx = next((p[0] for p in path if p[1]<3.2),path[-1][0])
        fallback = max(2,min(22.5,lx)) if lx > 0 else HOME
        best = -1e9
        best_set = False
        best_i = -1
        choice = (-1 if x>fallback+0.25 else 1 if x<fallback-0.25 else 0,0)
        if bx < -5 and bvx < 0:
            return [int(choice[0]<0),int(choice[0]>0),0]
        for jump in (0,1) if y < 1.51 else (0,):
            sy,svy = y,vy
            vertical = []
            targets = {round(fallback*12)/12}
            for i,(px,py,ux,uy) in enumerate(path[:65]):
                svy = 13.5*jump if sy <= 1.5034 else svy-0.98
                sy = max(1.5,sy+svy/30)
                if sy == 1.5:
                    svy = 0
                vertical.append((sy,svy))
                if px>0 and 0<py-sy<2.5 and abs(px-x)<17.5*(i+1)/30+2:
                    for off in (-0.9,0,0.9):
                        targets.add(round(max(2,min(22.5,px+off))*12)/12)
            # Close destinations duplicate the same actuator sequence.
            for target in sorted(targets):
                sx = x
                first = 0
                for i,((px,py,ux,uy),(sy,svy)) in enumerate(zip(path,vertical)):
                    a = -1 if sx>target+0.25 else 1 if sx<target-0.25 else 0
                    if i == 0:
                        first = a
                    oldx = sx
                    sx = max(2,min(22.5,sx+a*17.5/30))
                    pvx = (sx-oldx)*30
                    dx,dy = px-sx,py-sy
                    d2 = dx*dx+dy*dy
                    if 0.001<d2<4:
                        d = math.sqrt(d2)
                        nx,ny = dx/d,dy/d
                        un = (ux-pvx)*nx+(uy-svy)*ny
                        outx,outy = ux-2*un*nx,uy-2*un*ny
                        attack = shot(px,py,outx,outy,ox)
                        setting = setup_score(px,py,outx,outy,sx) if not armed else -100
                        score = max(attack,setting)-0.012*i
                        if score>best:
                            best,choice = score,(first,jump)
                            best_set,best_i = setting>attack,i
                        break
            if best_set and best_i == 0:
                armed = True
            return [int(choice[0]<0),int(choice[0]>0),choice[1]]
    return act
```

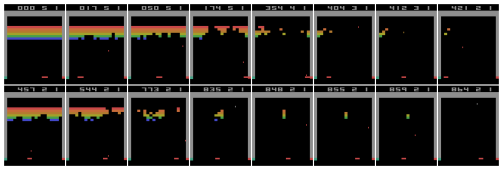
Code 2: Complete implementation of GPT-6 Astra's shielded Slime Volleyball champion.

D.2 Best Policy Galleries

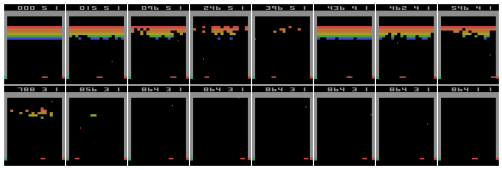
For each selected task, the gallery shows an episode from one of the highest-scoring policies reported for that task. We also note the authoring model, give the policy's held-out score, and mark shielded cases.



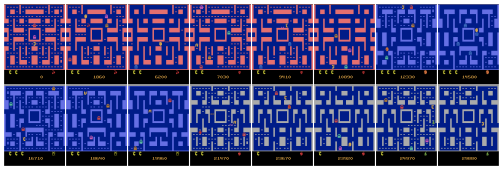
Boxing, DeepSeek V4.1 Flash, 100.



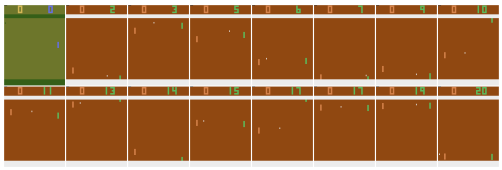
Breakout, Claude Fable 5, 864.



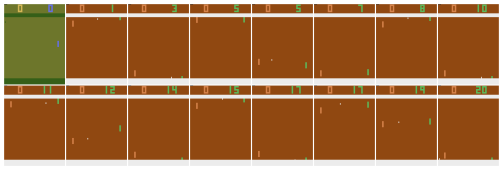
Breakout (raw pixels), Claude Opus 4.8, 864.



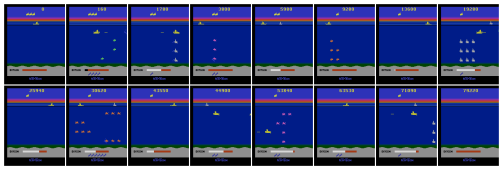
Ms. Pac-Man, GPT-6 Astra, 29,881.



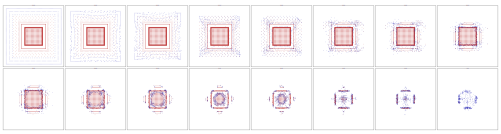
Pong, Claude Fable 5.1, 21.



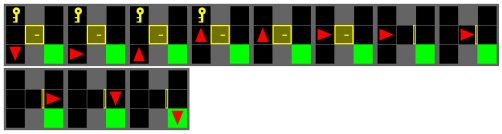
Pong (raw pixels), Claude Fable 5, 21.



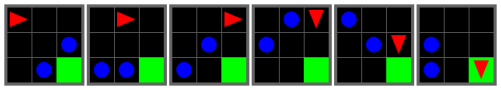
Seaquest, DeepSeek V4.1 Flash, 79,220.



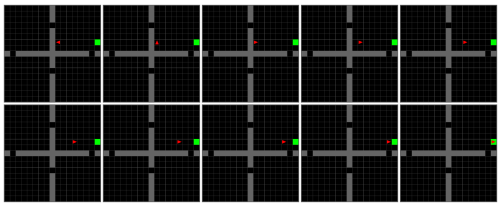
Gather, Claude Fable 5.1, 14,431.4.



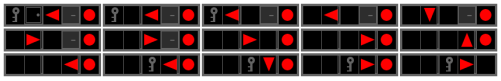
Door Key, Claude Opus 4.8, 0.964.



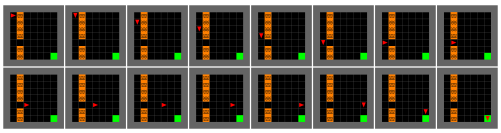
Dynamic Obstacles, Claude Fable 5, 0.909.



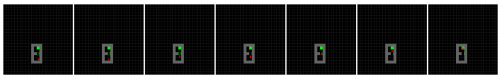
Four Rooms, GPT-6 Astra, 0.765.



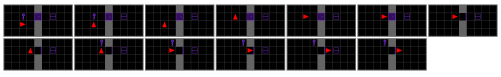
Key Corridor, Claude Fable 5, 0.951.



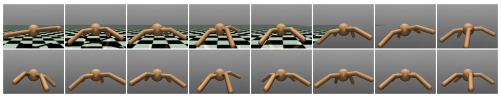
Lava Crossing, Claude Fable 5, 0.96.



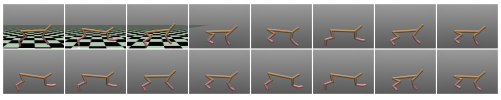
Multi Room, Claude Fable 5, 0.842.



Unlock Pickup, Claude Fable 5, 0.948.

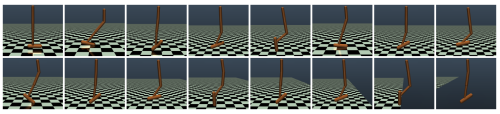


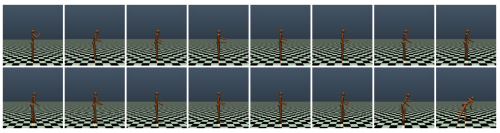
Ant, Gemini 3.6 Flash, 3,352.02.

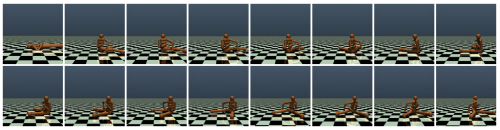


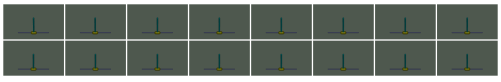
HalfCheetah, Claude Opus 5, 5,121.466.

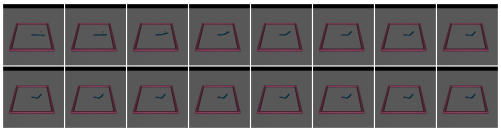
Figure 13: Policy rollouts across tasks (part 1 of 2).

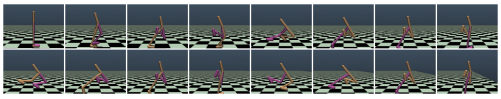

Hopper (shielded), Claude Fable 5.1, 3,732.019.

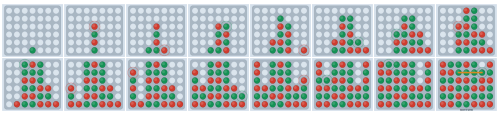

Humanoid, GPT-5.6 Sol, 1,147.702.


Humanoid Standup, Claude Opus 4.8, 162,659.248.


Inverted Double Pendulum, GPT-5.6 Terra, 9,359.889.

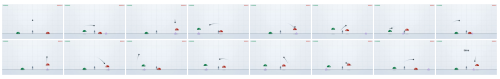

Reacher, GPT-6 Astra, -3.271.


Walker2d, Qwen3.7-Max, 5,635.257.

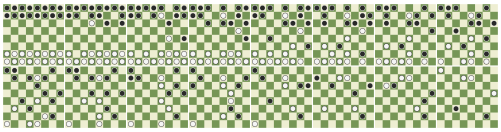

Connect Four (shielded), Claude Fable 5, 1.

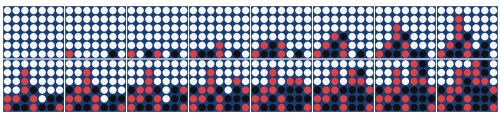

Core War, GPT-6 Astra, 0.843.

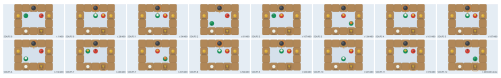

Kuhn Poker, Claude Opus 4.8, 0.22.

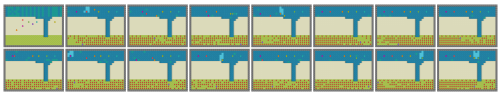

Slime Volleyball (shielded), GPT-6 Astra, 0.663.

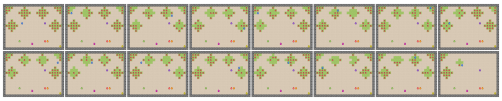

Tic-Tac-Toe, Claude Fable 5, 0.5.


Breakthrough, GPT-6 Astra, 3.051.

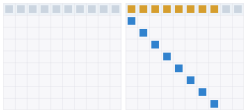

Connect Four (MO), Claude Fable 5, 4.845.


Overcooked (shielded), GPT-6 Astra, 210.

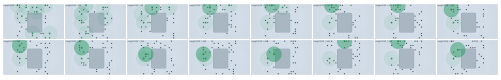

Clean Up, DeepSeek V4.1 Flash, 1,002.94.


Commons Harvest, Claude Fable 5.1, 433.76.


Beach, Claude Fable 5.1, 6.948.


Gem Mining, GLM-5.3, 3.869.


Item Gathering, Claude Fable 5, 8.


Pursuit, Claude Fable 5.1, 134.102.

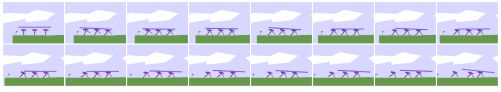

Multiwalker Stability, Claude Opus 5, 27.608.

Figure 14: Policy rollouts across tasks (part 2 of 2).

D.3 Code Length and Calibration

Comparison. Figure 15 compares median policy length (in lines of code, excluding comments, docstrings, and blank lines) with posterior z score across 42 models. The dashed line is a least-squares fit of posterior z to policy length on a linear scale. Figure 16 compares each champion's self-reported and independently measured scores, both standardized within its task. The agreement rate is computed using all paired reports and measurements.

Findings. Figure 15 shows a positive association between median policy length and posterior z at the model level (Pearson's $r = +0.54$, slope $b = 1.11$ z per 100 lines). However, across all individual champion policies, the association between line count and within-task z is much weaker (Pearson's $r = +0.14$, slope $b = 0.10$ z per 100 lines). This suggests that longer policies are not consistently stronger at the individual-policy level and that the model-level association is better attributed to model traits such as capability or tendencies.

Figure 16 shows that self-reports usually agree closely with independent measurements, but errors lean toward overestimation. Among all paired self-reported and independently measured scores, the two scores agree in 74% of cases, assuming a relative tolerance of 1% or an absolute tolerance of 10^{-6}. Disagreements include more overestimates than underestimates.

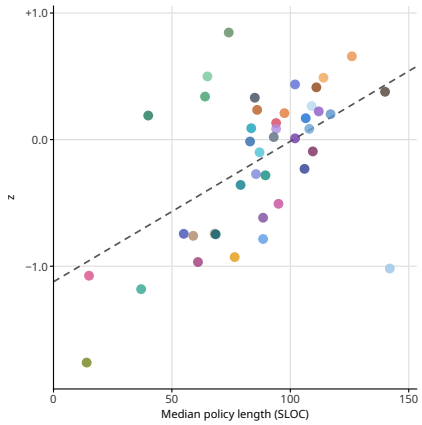


Figure 15: Median policy length versus posterior z score across 42 models. The dashed line is a least-squares fit on the linear scale.

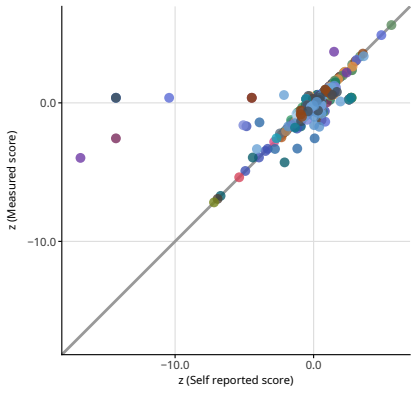


Figure 16: Self-reported versus independently measured scores, standardized within each task. The dashed line indicates agreement.

D.4 Model Version Comparisons

Figure 17 compares model versions of the same family. Newer versions improve unevenly across tasks, and every pair includes regressions. Many ties occur at scoring ceilings.

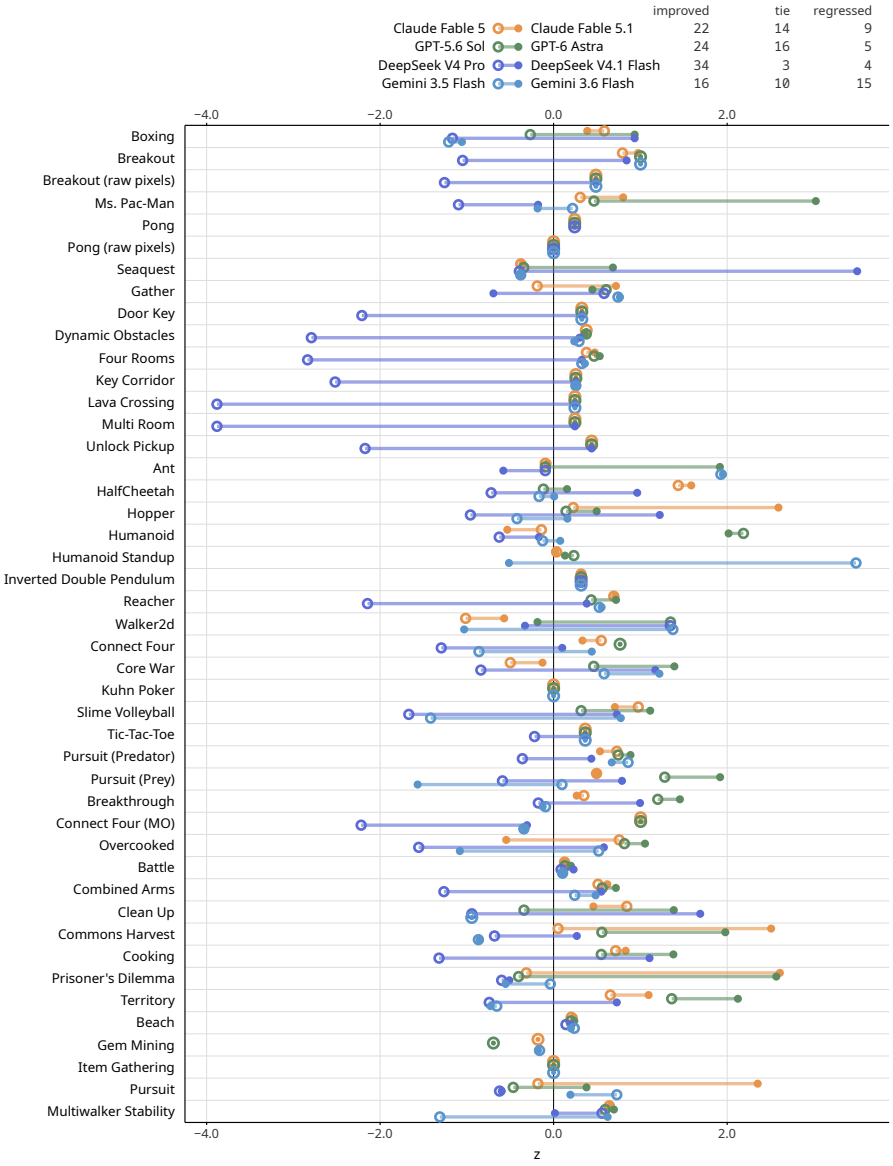


Figure 17: Model-version comparisons by task. Hollow circles represent older models, and solid dots represent newer models.

D.5 Routing-based Models

Setup. We evaluate Fugu and Fugu Ultra (Sakana AI, 2026) as examples of routing and orchestration over underlying models. Both use Codex at high reasoning with the same 30-minute budget and 12-attempt target as the primary comparison. Because their mechanism differs from that of the other models in our roster, we exclude both systems from the primary comparison. For this analysis only, we add them to the primary roster and refit the estimator in Section 3.3 over all 49 models. Figure 18 shows the joint ranking, and Table 6 compares both systems with the four strongest other models using selected metrics from Table 1.

Findings. Across tasks, Fugu Ultra and Fugu rank among the top three by posterior z mean. However, uncertainty limits claims of superiority, so we view them as similarly competitive with the strongest models. Their performance is generally strong, with some weaker tasks. Fugu's weaker cases include Unlock Pickup, Pursuit (Prey), and Clean Up. Fugu Ultra's lower-ranked cases include Core War, Prisoner's Dilemma, and Beach. Their strong aggregate performance suggests that adaptive routing and orchestration are promising, while the weaker cases suggest that their advantages may remain task-dependent.

Pricing. Fugu's cost is unavailable because the selected model is undisclosed. Fugu Ultra's recorded cost omits billed orchestration tokens and is a lower bound, preventing a complete cost comparison.

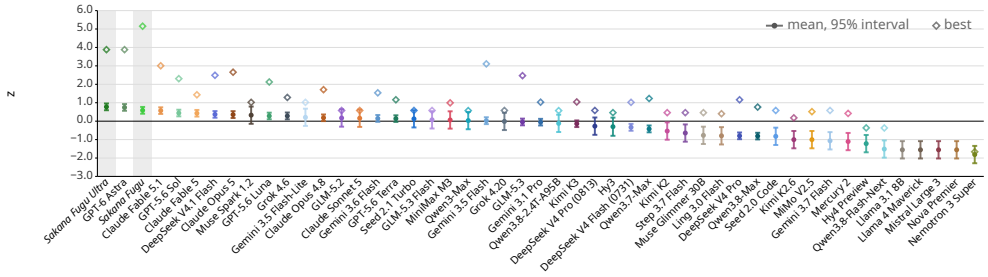


Figure 18: Joint ranking of 47 models and two Fugu models (shaded). Dots show posterior z means with 95% intervals. Diamonds show best single-run scores.

Table 6: Top six models with the Fugu models (shaded) added to the 47 models. All intervals are 95% intervals.

Model	Rank interval	z mean	z interval	Percentile	Percentile interval
Fugu Ultra	$[1, 4]$	$+0.776$	$[+0.59, +0.96]$	77.2	$[72.2, 82.2]$
GPT-6 Astra	$[1, 4]$	$+0.745$	$[+0.56, +0.93]$	78.8	$[73.9, 83.7]$
Fugu	$[1, 8]$	$+0.587$	$[+0.40, +0.77]$	69.8	$[64.8, 74.8]$
Claude Fable 5.1	$[2, 9]$	$+0.572$	$[+0.39, +0.75]$	67.3	$[62.4, 72.1]$
GPT-5.6 Sol	$[3, 13]$	$+0.441$	$[+0.26, +0.62]$	68.4	$[63.5, 73.3]$
Claude Fable 5	$[3, 14]$	$+0.428$	$[+0.25, +0.61]$	65.5	$[60.7, 70.4]$

D.6 TRAINED RL BASELINES DETAILS

Figures 8 and 9 in Section 4.4 compare code-policy champions with separately trained RL agents. We provide details of these trained RL baselines here.

Training. We use Stable-Baselines3 (Raffin et al., 2021) implementations of PPO on Hopper-v5 and SAC on Humanoid-v5, with Gymnasium over MuJoCo (Towers et al., 2025; Todorov et al., 2012). Both use the library's default hyperparameters without tuning, with 3 changes. The policy and value networks use 2 hidden layers of 256 units each, SAC warms up for 10,000 steps, and the training budgets are set per task. The SAC changes follow the RL Baselines3 Zoo entry for Humanoid (Raffin, 2020), while PPO keeps the library defaults rather than the Zoo's tuned Hopper entry. Both policies read the task's default proprioceptive observations, the same inputs code policies receive, and both use a learning rate of 0.0003 and a discount factor of 0.99. PPO trains for 10^6 steps using 8 parallel task instances, rollouts of 2,048 steps per instance, a batch size of 64, and 10 update epochs per rollout, with a GAE parameter of 0.95, a clipping parameter of 0.2, and an entropy coefficient of 0. SAC trains for 2×10^6 steps using 1 task instance, a replay buffer of 10^6 transitions, a batch size of 256, 10,000 warm-up steps, 1 gradient update per step, and a target-smoothing coefficient of 0.005. We evaluate the final policy after the full training run, without checkpoint selection.

Evaluation. Each trained policy is evaluated over 30 episodes. The figures report the mean and population standard deviation across episodes, not across training seeds. The random-policy reference uses uniformly sampled actions.

Reference lines. Hopper's solved threshold of 3,800 is the registered Gymnasium reward threshold. Humanoid has no registered solved threshold. The 6,000 reference is a rounded strong-agent score in the range reported by SAC and the Zoo benchmark (Haarnoja et al., 2018; Raffin, 2020). Both serve as performance references, not task-completion criteria.

D.7 Cross-Task Performance Similarity

Comparison. To assess task similarity, we compare model rankings for each task pair using Kendall's τ_b. Figure 19 shows the matrix of pairwise rank correlations. We exclude three tasks with no score variation and nine others where more than half the measured models tie, an empirical decision to limit sensitivity to individual rank swaps. To better understand individual pairs, Figure 20 examines four non-overlapping task pairs, showing each model's within-task standardized z scores rather than its ranks.

Findings. Task rankings in Figure 19 agree only partially. The 528 eligible pairs across 33 tasks have a median Kendall's $\tau_b = +0.298$, with 73 negative correlations. The task pairs in Figure 20 suggest that tasks have different characteristics and that performance is not uniformly related across them. These differences in models' responses to individual tasks favour evaluation across diverse tasks.

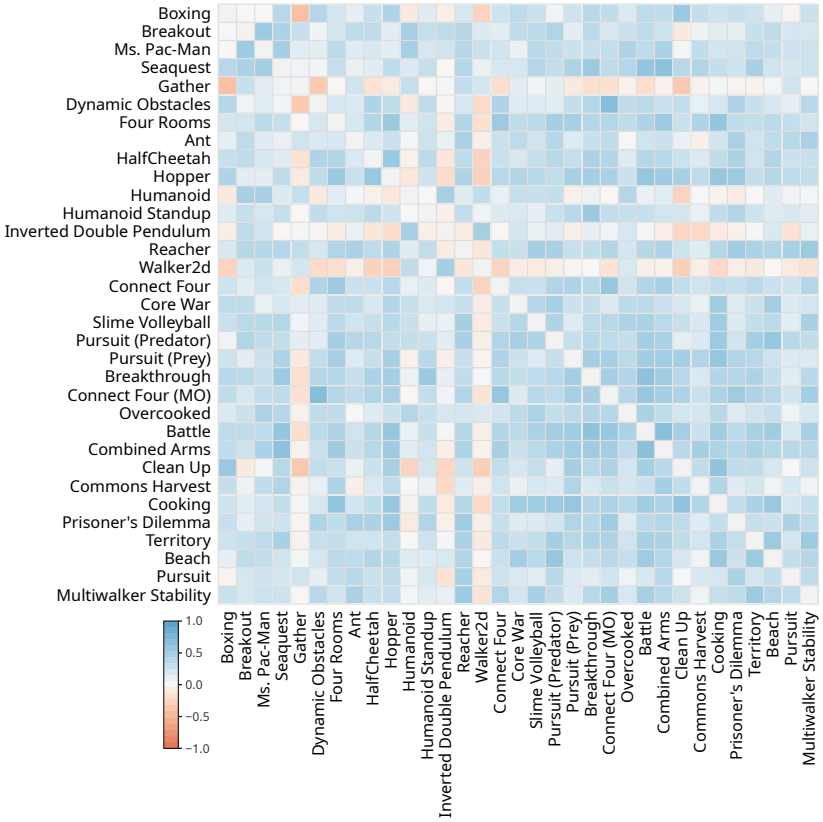


Figure 19: Cross-task Kendall's τ_b for selected tasks. Blue indicates agreement and red indicates disagreement.

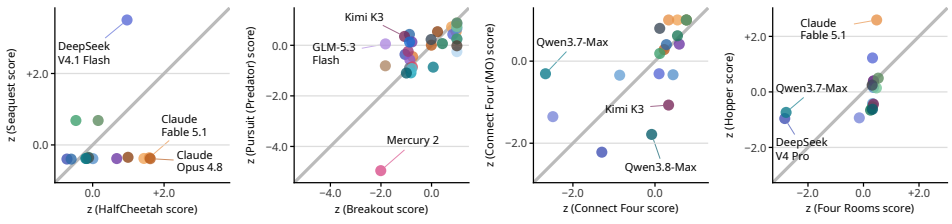


Figure 20: Comparisons of four task pairs selected from Figure 19. Points denote shared models. Labels mark the three largest deviations for each pair.

D.8 Comparison with Established Benchmarks

Comparison. We compare our ranking with the 15 benchmark rankings listed in Table 7. These extend beyond the works discussed in Section 2 to provide a broader benchmark comparison. Each comparison includes at least 5 shared models, matched by registry identifier. Scores are reused as-is without re-evaluation. To construct each ranking, we look for related scoreboards available at the time of writing, keep each model's highest listed score, and exclude disputed variants. The table identifies the online result tables, developer pages or repositories, and third-party aggregators from which the scores were collected. Given this processing of statistical data, the comparison is better considered a descriptive one that assesses agreement in model ordering and does not formally establish the benchmarks' sensitivity to differences in capability.

Findings. Table 7 reports Kendall's τ_b after reversing lower-is-better metrics. Figure 21 shows ALE-Bench, which has the largest overlap (37 models, $\tau_b = 0.570$). The rankings broadly agree, with many fine-grained differences. We argue that this suggests our benchmark shares broad similarities with established benchmarks while focusing on different aspects of agents' capabilities. We also note that limited overlap and differing harnesses, reasoning settings, and score sources limit interpretation.

Table 7: Established benchmarks' rank agreement with our ranking.

Benchmark	Score Source	# Shared Models	Kendall's τ_b
SEC-bench Pro (Lee et al., 2026) (Pass@1)[a]	Aggregator (LLM Stats)	5	+0.800
Agents' Last Exam (Sun et al., 2026) (Score)	Leaderboard (Snorkel AI)	15	+0.600
ExploitGym (Wang et al., 2026b) (Pass@1)	Aggregator (BenchLM)	6	+0.600
ALE-Bench (Imajuku et al., 2025) (mean performance, one self-refinement)	Leaderboard (Official)	37	+0.570
ProgramBench (Yang et al., 2026) (Almost@1)	Results (Official)	8	+0.500
Terminal-Bench 4.0 (Marten, 2026) (Pass@1)	Aggregator (LLM Stats)	12	+0.485
Terminal-Bench 3.0 (Marten et al., 2026) (Pass@1)	Results (Official)	11	+0.455
DeepSWE v1.1 (Huang & Jiang, 2026) (Pass@1)	Results (Official)	20	+0.400
Chartography (Garre et al., 2026) (Pass@1, no tools)	Leaderboard (Surge AI)	22	+0.399
CyberGym (WANG et al., 2026) (Pass@1)	Aggregator (BenchLM) Aggregator (LLM Stats) Leaderboard (Official)	13	+0.348
ZeroBench-main (Roberts et al., 2026) (Pass@5, with tools)	Leaderboard (External)[b]	9	+0.333
AutomationBench (Shepard & Salimans, 2026) (Pass@1, public tasks)	Results (Official)	10	+0.289
Terminal-Bench 2.1 (Merrill et al., 2026; The Terminal-Bench Team, 2026) (Pass@1)	Aggregator (LLM Stats)	23	+0.258
BabyVision (Chen et al., 2026) (reported accuracy)	Leaderboard (UniPat AI)	7	+0.238
NL2Repo-Bench (Ding et al., 2026) (Score)	Aggregator (LLM Stats)	13	+0.026

Metric labels follow the score source's reporting and may differ from the benchmark paper's name (e.g., success rate).
[a]The 5 scores used here are classified by LLM Stats as self-reported and unverified. They are absent from the official leaderboard at the time of checking.
[b]The ZeroBench scores come from the official site's externally reported leaderboard.

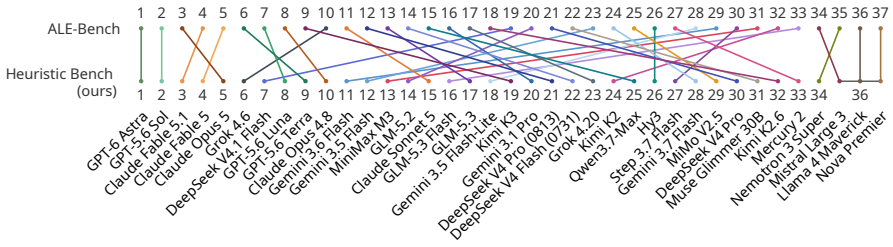


Figure 21: Our percentile ranks (left) versus those of ALE-Bench (right). Rank 1 is the best.