HIERARCHICAL EXPERIMENTALIST AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are increasingly used to act in the real world and augment human decision-making, yet most systems rely on parametric knowledge acquired through imitation, optionally improved by post-training, retrieval, or search over fixed data. This paradigm struggles in novel domains or on complex queries that require information unavailable from prior knowledge alone; knowing the laws of physics, for example, does not by itself enable reliable reasoning or long-horizon actions in a complex physical system. We argue that agents therefore need *active experimentation*: the ability to gather targeted query-specific evidence, discover general principles of unseen environments, and convert interaction experience into reusable skills. We introduce **Hierarchical Experimentalist Agents (HExA)**, an in-context, experiment-centric self-improvement framework that (1) iteratively designs and refines query-relevant experiments, (2) incrementally distills experience into reusable and composable skills that accelerate experimentation within and across tasks, and (3) integrates experimental evidence to act or answer queries. HExA can work with any LLM including frontier black-box models, and requires no external supervision, oracle solutions, or offline data. We evaluate HExA on multi-step reasoning, interactive partially observable games, and physics law discovery tasks. To further evaluate when active experimentation is especially necessary, we introduce INTERPHYRE, a set of hard procedural physics tasks where we enable tool-calling and intervention APIs. Current frontier LLMs including GPT-6 Astra fail in these settings: on the most challenging Interphyre tasks, we show that HExA can enable such models to near 100% success. Moreover, HExA improves token efficiency compared to naive agentic and memory baselines and enables generalization via hierarchical skill transfer from easier to harder tasks.

1 INTRODUCTION

Large language models (LLMs) and LLM agents have demonstrated remarkable capabilities across a wide range of knowledge-intensive and agentic tasks, from code generation and mathematical reasoning to multi-step planning (Gao et al., 2025; Jimenez et al., 2024; Yao et al., 2023b; Schick et al., 2023). Most deployed LLM systems, however, still rely primarily on parametric knowledge acquired through imitative pretraining and post-training over large collections of human-generated data or environments, augmented by static retrieval and test-time reasoning (Anthropic, 2026; OpenAI, 2025; Team et al., 2024; Team, 2026; Gao et al., 2024). This paradigm works well when the relevant information is well represented in training data or can be recovered through reasoning and in-context learning from zero- or few-shot prompts (Dong et al., 2024; Mei et al., 2025; Dou et al., 2026). But it breaks down in two important settings. First, an agent may encounter a novel environment whose dynamics, constraints, or effective strategies were absent from training corpus or are unknown even to humans. Second, even when the relevant general knowledge is encoded in the model's parameters, applying it reliably to a particular instance may require far more than recall and reasoning, like information that must be acquired through interaction: knowing the laws of physics, for example, does not by itself solve an experimental physics problem (Silver & Sutton, 2025; Ying et al., 2025).

In real world settings, given the vast space of novel tasks and queries, an agent's success likely cannot rely on pretrained parametric knowledge alone. Instead, we argue that, agents should, in principle, learn like scientists through active experimentation: this entails exploration, information-gathering, proposing falsifiable hypotheses, testing them through interaction and experiments, and reasoning over what the environment confirms or refutes (Spelke & Kinzler, 2007; Chuck et al.; Team et al., 2023; 2021; Chandra, 2025). This is precisely the information that parametric recall, longer chains of

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.

thought, and static retrieval cannot directly provide. Experimentation, however, is costly: solving each instance from scratch may require many interaction cycles, even when related instances share useful structure. An effective agent should therefore not only blindly re-explore, but *learn from its experiments*—distilling reusable skills from prior experience and transferring them across related instances to amortize the cost of future experimentation. These observations motivate the central question of our work: *"How can LLM agents efficiently learn to reason, act, and generalize in complex, novel domains via in-context experimentation, skill learning, and transfer?"*

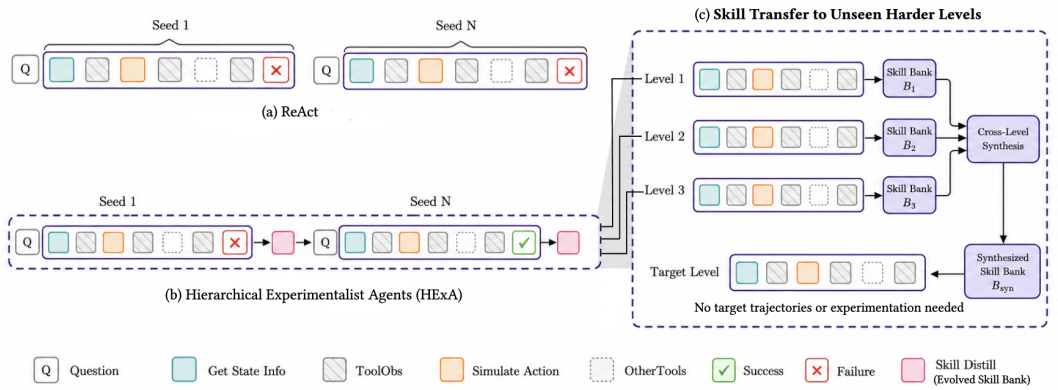


Figure 1: Overview of the HExA framework. (a) Baseline ReAct: each episode is solved independently with no cross-episode learning. Each episode consists of alternating tool calls for experimentation, terminating in success (✓) or failure (×). **(b) HExA:** episodes are processed in successive rounds. After each episode, an evolver distills the trajectory into a *Skill Bank* of reusable skills and recurring mistakes that condition subsequent episodes. Repeated interaction helps evolve and refine the task-specific Skill Bank. **(c) Transfer to unseen tasks:** each source task runs the HExA loop independently to build its Skill Bank. The resulting banks are synthesized by the evolver into a single generalizable Skill Bank for zero-shot transfer to harder target tasks, with no target trajectories used during bank construction.

We thus propose **Hierarchical Experimentalist Agents (HExA)** (Figure 1), a novel in-context reinforcement learning (ICRL) framework that extends in-context learning along three dimensions: (1) the actor LLM actively interacts with an environment through tools, proposing and testing hypotheses to gather query-relevant evidence through deliberate experimentation rather than passive observation; (2) the evolver LLM contrasts successful and failed experimental trajectories and distills their reusable knowledge into a persistent external skill bank; (3) a retriever selects relevant skills from the bank and injects them into the actor's context for subsequent episodes, providing stepping-stones for further experimentation progress while amortizing the cost of exploration. HExA is hierarchical in the operational sense that later trajectories are generated while conditioned on previously learned skills, allowing subsequent skills to refine, compose, delimit, or supersede earlier ones. Beyond task-specific strategies, this process can therefore form higher-level experimental skills that generalize across new seeds and configurations and transfer to structurally different tasks. Unlike trained ICRL approaches that first train a model on offline interaction histories to implement a learning algorithm in context (Laskin et al., 2022; Moeini et al., 2025) or skill based post-training methods (Xia et al., 2026b; Lu et al., 2026), HExA requires no reward or action labeled offline training data, additional meta-training, or parameter updates to the actor. Instead, its learning experience is self-generated online by the current skill-conditioned policy: the evolving bank guides hypothesis-driven experimentation, successful and failed trajectories update the skill bank, that then conditions the next round. Thus, learning and data collection form a closed in-context self-improvement loop: what the agent has already learned changes not only its subsequent actions, but also which experiments it performs, which evidence and skills it collects next (Section 2). We share other detailed related works in Appendix A.

Recent benchmarks for multi-step reasoning tasks (Shojaee et al., 2026; Opedal et al., 2025; Lin et al., 2025a; Nasvytis & Fan, 2026a) and interactive partially observable games (Liu et al., 2025) and analysis of data unseen during pretraining (Dou et al., 2026)—focus to some extent on evaluating the in-context learning ability of frontier LLMs, but rely mostly on static reasoning and factual analysis. We evaluate HExA on these easier task setups as well. But we instantiate and formalize HExA for a broader setting: tasks where solving an instance requires active experimentation to discover new knowledge or gather targeted, instance-specific evidence, rather than recall and reasoning alone.

This includes both settings where the model already knows the relevant general principles but must experiment to apply them precisely, and genuinely unfamiliar domains where useful knowledge must be acquired through interaction with unseen open-ended environments (Team et al., 2021; Hughes et al., 2024). Physical reasoning is a natural testbed for this harder setting, since models often have the relevant principles yet cannot apply them at the required precision without active experimentation. To study and evaluate this general setting concretely, we introduce Interphyre (Section 3.1 and Appendix D), a procedural 2D physics reasoning benchmark for LLM agents, built on top of PHYRE (Bakhtin et al., 2019), with programmatic experimentation tool interfaces via scene inspection, simulation, interventions, and intermediate-dynamics observations.We additionally evaluate HExA on DiscoverPhysics (Wiemann et al., 2026), another recent benchmark testing agent experimentation ability. Together, these allow us to evaluate how HExA improves capability LLM agents for self-improvement and generalization via hypotheses driven interactive learning and hierarchical skill composition in complex unseen experiment-centric tasks, all in a non-parametric in-context setup.

In summary, our main contributions are:

1. We formalize an **experiment-centric problem setting** and introduce **Interphyre**, a benchmark to better evaluate and improve generalist LLM agents for this problem setting specifically.
2. We propose **HExA**, in which an LLM actor, evolver and retriever, jointly work to design experiments, test hypotheses, distill new knowledge and experience into composable hierarchical skills that evolve over interactions and generalize to unseen tasks.
3. HExA improves both frontier and smaller LLMs, including black-box models. On the two hardest INTERPHYRE levels, where zero-shot success is near zero, HExA reaches **83%** and **96%** with Claude Sonnet 4.6, and **100%** and **76%** with GPT-6-Astra. Skills synthesized from easier levels transfer zero-shot to harder ones, raising `catapult` success from 11% (ReAct) to 44% (**+33** points). Beyond InterPhyre, HExA is best in 9 of 10 task-actor settings across interactive games, reasoning & planning, and DISCOVERPHYSICS.

2 HIERARCHICAL EXPERIMENTALIST AGENTS: LEARNING VIA EXPERIMENTATION IN-CONTEXT

We now formalize HEXA's problem setup and self-improvement objective, and describe the actor–evolver–retriever loop, experimentation feedback, hierarchical skill-bank, and cross-task transfer mechanism. Figure 1 gives the high-level framework. We use the `catapult` level of INTERPHYRE as a running example; all environment and task details are provided in Appendix D and E.

Problem setup. We consider a task family ℓ with instances $\{(\ell, s_j)\}$, where each s_j specifies an environment configuration (e.g., geometry, parameters, and initial conditions). For each family ℓ we hold a set of experimentation seeds $\mathcal{S}_\ell = \{s_1, \ldots, s_{|\mathcal{S}_\ell|}\}$, on which the agent experiments and the skill bank is evolved, and a disjoint set $\mathcal{S}_\ell^{\text{ev}}$ on which the resulting bank is evaluated. The actor interacts with each instance through an environment-provided tool interface for experimentation $\mathcal{F}$, which may expose scene inspection, simulation, interventions and hypotheses testing, and task-specific analysis (Appendix F). Each episode has at most T interaction turns. At turn t, the actor observes the history $h_t = (a_0, o_0, \ldots, a_{t-1}, o_{t-1})$ and selects a tool call

$$a_t \sim \pi_{\text{act}}(\cdot \mid \ell, s_j, \mathcal{G}, h_t; \mathcal{F}), \qquad o_t = \textsc{Env}(a_t), \quad (1)$$

where $\mathcal{G}$ is a skill context retrieved once at the beginning of the episode. The episode terminates after a terminal action, a successful simulation, or the T-turn is over, yielding trajectory $\tau_j = h_{T_j}$ with $T_j \le T$. a binary task $\textsc{Goal}_\ell$ a binary task scores, tells the terminal simulation outcome. Writing $\textsc{Sim}(\tau_j)$ for the terminal simulator state associated with trajectory τ_j, success is

$$y_j(\pi_{\text{act}}; \ell, \mathcal{G}) = \mathbb{I}[\textsc{Goal}_\ell(\textsc{Sim}(\tau_j))], \qquad \tau_j \sim \pi_{\text{act}}(\cdot \mid \ell, s_j, \mathcal{G}). \quad (2)$$

Equation 2 defines the task without method-specific assumptions: any agent can be evaluated against it, and skill-free baselines are the case where $\mathcal{G} = \emptyset$.

Algorithm 1: HEXA

Input: Actor π_{act}, evolver π_{ev}, level ℓ, seeds $\mathcal{S}_\ell$, rounds R, episodes/round x

```
// off2on warm start
K_0 ← π_ev(∅, T_0) or K_0 ← ∅
for n = 1, ..., R do
    // top-M skills + top-N mistakes
    G_n ← RETRIEVER(K_{n-1}, ℓ, M, N)
    for i = 1 + x(n-1), ..., xn do
        // task + tools + skills
        τ_i ← π_act(· | ℓ, s_i, G_n)
        // outcome × efficiency
        r(τ_i) ← REWARD(τ_i)
    // evolver: old bank + new trajectories + rewards
    K_n ← π_ev(K_{n-1}, T^(n))
return K_R
```

In-context self-improvement objective. Equation 2 is method-agnostic: skill-free agents have $\mathcal{G} = \emptyset$, while HEXA improves the same frozen actor by evolving only the external skill state that conditions it. Let $\mathcal{D}_\ell$ denote the distribution of future instances and $\mathcal{G}_{\mathcal{K},\ell} = \textsc{Retriever}(\mathcal{K}, \ell, M, N)$. For fixed π_{act}, we define

$$J_\ell(\mathcal{K}) = \mathbb{E}_{s\sim\mathcal{D}_\ell}\left[y(\pi_{\text{act}}; \ell, \mathcal{G}_{\mathcal{K},\ell})\right]. \tag{3}$$

HEXA keeps π_{act} fixed and evolves $\mathcal{K}$ from interaction on $\mathcal{S}_\ell$ to increase J_ℓ. Thus, self-improvement occurs entirely through context: $\mathcal{K}_R$ is useful when the same frozen actor performs better with $\mathcal{K}_R$ than with $\mathcal{K} = \emptyset$ on unseen instances.

The HEXA loop. HEXA alternates three components: an *actor* π_{act} that experiments in the environment, an *evolver* π_{ev} that distills the resulting trajectories into $\mathcal{K}$, and a *retriever* that selects skill bank entries for the actor's next episodes. Algorithm 1 repeats this loop for $R = \lceil |\mathcal{S}_\ell|/x \rceil$ rounds as shown in Appendix figure 8. At round n, the retriever constructs $\mathcal{G}_n = \textsc{Retriever}(\mathcal{K}_{n-1}, \ell, M, N)$. The actor then runs on a fresh batch of at most x experimentation seeds under the same $\mathcal{G}_n$, producing trajectories $\mathcal{T}^{(n)}$ with rewards. Importantly, $\mathcal{T}^{(n)}$ is generated under $\mathcal{K}_{n-1}$ rather than drawn from a fixed corpus, so each bank update changes the policy that collects the next round of experience. The evolver updates $\mathcal{K}_n = \pi_{\text{ev}}(\mathcal{K}_{n-1}, \mathcal{T}^{(n)})$. No model weights change; the evolving bank is the persistent state carried across rounds. $\mathcal{K}_0$ may be empty or warm-started from skill-free actor trajectories, as ablated in Section 3.3 (see Algorithm 2 for all variants). After the final round, $\mathcal{K}_R$ is frozen: the evolver is not invoked during held-out evaluation on $\mathcal{S}_\ell^{\text{ev}}$ (Algorithm 4 gives the full inference procedure).

Skill-augmented actor. At the beginning of each episode, the actor receives the task description, tools $\mathcal{F}$, and a skill context retrieved from the current bank:

$$\mathcal{G} = \textsc{Retriever}(\mathcal{K}, \ell, M, N), \qquad \tau_j \sim \pi_{\text{act}}(\cdot \mid \ell, s_j, \mathcal{G}). \tag{4}$$

The retriever selects at most the top-M strategy skills and top-N mistake records according to the scores described below and inserts them as a bounded textual prefix. This changes only the actor's context: its architecture, parameters, and decoding procedure remain unchanged.

Trajectory reward. Each trajectory may receive a scalar reward $r(\tau) \in [-1, +1]$, which provides the evolver with evidence about trajectory quality. The signal can come from any available scorer—for example, a programmatic environment reward, human feedback, an LLM judge, or a learned reward model. In our INTERPHYRE experiments, the reward combines task outcome and interaction efficiency; its exact definition is given in Appendix C.2. HEXA can also operate without explicit scalar reward labels, in which case the evolver assesses trajectory quality from the trajectories themselves.

Skill distillation and retrieval. After round n, the evolver receives the current bank $\mathcal{K}_{n-1}$ and $\mathcal{T}^{(n)}$, with each τ_j's reasoning traces and reward, and returns

$$\mathcal{K}_n = \pi_{\text{ev}}(\mathcal{K}_{n-1}, \mathcal{T}^{(n)}), \qquad |\mathcal{K}_n^{\text{sk}}| \le M_{\max}, \qquad |\mathcal{K}_n^{\text{mis}}| \le N_{\max}. \tag{5}$$

The bank contains two complementary record types. *Strategy skills* $\mathcal{K}^{\text{sk}}$ contain a *title*, a *principle* capturing reusable task-solving and interaction insights gained from testing hypotheses through experimentation, a *when-to-apply* condition, an optional demonstrative *example*, the *source-seed provenance*, and a *reward score* r_k. *Mistake records* $\mathcal{K}^{\text{mis}}$ capture recurring failures as triples $\mu = (\delta, \rho, \alpha)$: failure pattern δ, inferred root cause ρ, and corrective strategy α. The evolver also recovers *partial skills*—correct intermediate discoveries embedded in otherwise failed trajectories—so that a failed episode need not be treated as uninformative(Chandra et al., 2025). Thus, reward is not used merely to select successful demonstrations: unsuccessful experiments can contribute failure corrections and valid partial discoveries, allowing both positive and negative experience to improve the subsequent policy. Each skill receives a score r_k derived from the mean reward $\bar{r}_{\text{src}}$ of the trajectories that support it (Section 3.3). These scores serve two purposes: the retriever prioritizes the top-M strategy skills and top-N mistake records, and the evolver uses them when retaining or pruning entries under the bank capacity constraint. The resulting $\mathcal{K}$ is therefore a compressed abstraction of prior experiments rather than a store of raw trajectories. Moreover, round-n trajectories are generated while conditioned on skills learned in earlier rounds. Subsequent evidence can therefore refine an earlier principle, narrow its applicability, combine it with complementary discoveries, or replace it when contradicted. We refer to this repeated abstraction and composition of previously

conditioned experience as *hierarchical skill formation*. Full schemas, prompts, reward details, and retrieval budgets are provided in Appendix C and Appendix K.

Abridged evolved skill bank used on `catapult` seed 64

Bank evolved on the training seeds, tested on the held-out seed 64.

S1: DEFAULT ▷ Right of the pivot, drop low Place the red ball at $x \approx 0.3$ on the right half of the arm… y is the primary range control, $y \approx -0.5$ giving maximum range and a ± 0.25 change shifting landing by 1.5–2.0 units. **Use when:** opening placement on a new seed.

S2: DIAGNOSE ▷ Chaotic cliff Chaotic zones exist where tiny parameter changes cause large trajectory swings… the cliff boundary can be as narrow as $\Delta y = 0.01$. **Use when:** two consecutive traces with small Δy land > 1.5 units apart; bisect rather than iterate.

S3: CORRECT ▷ Bisect, then commit Bisect a detected cliff to its midpoint… at standard drop heights ($y \leq 0$) commit to simulate once the gap is under 0.8 units. **Use when:** a cliff has been bracketed, or the gap threshold is met.

AVOID ▷ Linear search in a chaotic zone The agent assumes a monotonic response and keeps incrementing… when the system is in fact chaotic. **Correction:** switch from linear search to bisection.

On seed 64. The default traces to $x = 4.78$ against a target at 3.50; $y = 0.0$ collapses it to -0.92… the bank reads a cliff, so the agent bisects to $y = -0.25$, lands at 3.29, and commits: six iterations (App. K.6).

default → cliff $4.78/-0.92$ → bisect $y{=}-0.25$ → success

Cross-task skill transfer and generalization. The same composable skills can transfer knowledge across task families. Instance-specific details such as coordinates or object identities may not transfer, but mechanism-level principles—for example, how general strategies work or contact geometry affects impulse direction, can remain useful across different tasks and configurations. Given evolved source banks $\mathcal{K}_{\ell_1}, \ldots, \mathcal{K}_{\ell_S}$ and only a textual description of an unseen target task ℓ^*, a evolver synthesizes, $\mathcal{K}_{\ell^*} = \pi_{\text{ev}}^{\text{tr}}\big(\{\mathcal{K}_{\ell_i}\}_{i=1}^{S}, \text{desc}(\ell^*)\big)$, with $\mathcal{T}_{\ell^*} = \emptyset$. The evolver (i) selects source skill that are structurally relevant, (ii) removes source-specific parameter and entity bindings, (iii) re-grounds each retained principle in the target's entities, constraints, and tools, and (iv) assigns a transfer score based on structural correspondence and corroboration across source banks to help the retriever during zero-shot evaluations (Algorithm 3). Because no target trajectories are available during synthesis, the transfer procedure is constrained to express general mechanisms and applicability conditions rather than target-specific coordinates or observed solutions. The resulting $\mathcal{K}_{\ell^*}$ can be injected through the retriever in Eq. 4 for target experimentation free skill transfer, or used as $\mathcal{K}_0$ for subsequent within-task evolution. Full synthesis prompts are provided in Appendix I.2.5; results appear in Section 3.4.

3 EXPERIMENTS, RESULTS AND ABLATIONS

We evaluate HEXA along three questions: (i) whether skills learned from the actor's own experimentation improve the same frozen actor on held-out instances; (ii) which components of experimentation, skill evolution, and reuse drive these gains; and (iii) whether the resulting skills generalize and transfer beyond the tasks on which they were learned. We test across reasoning, interactive games, and scientific discovery, and use INTERPHYRE as a controlled setting for detailed analysis of active experimentation and hierarchical skill formation.

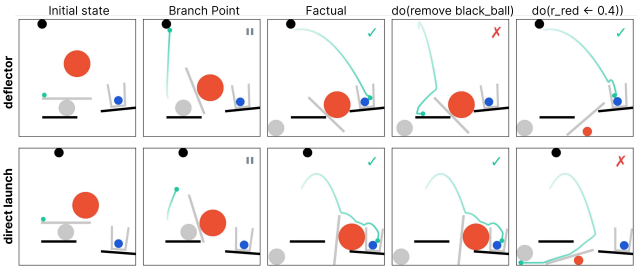


Figure 2: INTERPHYRE and its properties. (i) tasks whose solutions cannot be inferred from a static description and must be discovered through interaction and experiments; (ii) a tool-call programmatic interface for experimentation and hypothesis testing via scene inspection, interventions and intermediate-dynamics observations; and (iii) controllable difficulty through procedural generation.

3.1 EXPERIMENTAL SETUP

Benchmarks and Evaluation Setup. We introduce and use INTERPHYRE as the controlled testbed for analyzing how experimentation and skill acquisition improves self-improvement, and use three other task families to test whether the same mechanism generalizes broadly: multi-step reasoning with MathGAP (Opedal et al., 2024), ZebraLogic (Lin et al., 2025b), Tower of Hanoi (Shojaee et al., 2025), and Matchstick Arithmetic (Nasvytis & Fan, 2026b); interactive games including Wordle, Minesweeper, Mastermind, and Maze (Liu et al., 2025; Lin et al., 2026); and physics-law discovery with DISCOVERPHYSICS (Wiemann et al., 2026). Although HEXA yields significant performance and learning-efficiency gains on these tasks compared to baselines, these tasks are not designed to isolate the necessity of learning through active experimentation: zero-shot LLM performance

is already relatively high, indicating substantial prior familiarity, and the tasks do not explicitly support experimentation-centric interaction. INTERPHYRE is a benchmark of conceptually simple but difficult interactive physical-reasoning tasks built on PHYRE level designs (Bakhtin et al., 2019), with experimentation tools for active hypothesis testing. Unless stated otherwise, skill banks are learned on training instances, frozen, and evaluated on disjoint held-out instances for all tasks. Splits and task-specific protocols are in Appendix G.

Baselines and LLM agents. On INTERPHYRE, we evaluate GPT-6-Astra, Claude Sonnet 4.6, Qwen-2.5-{3B,7B}-Instruct, and GPT-OSS-{20B,120B}, each with at most 25 interaction turns per seed, together with a random valid-placement policy. Across the external benchmarks, we evaluate GPT-OSS-120B and Qwen3-8B on the interactive games, GPT-OSS-120B on reasoning and planning, and Claude Sonnet 5 on DISCOVERPHYSICS. We compare three baseline regimes: (i) memoryless agentic methods—ZERO-SHOT performance of base LLMs, REACT (Yao et al., 2023a), and Reflexion (Shinn et al., 2023); (ii) persistent episodic memory methods—Episodic RAG (Lewis et al., 2020) and MEMRL (Zhang et al., 2026b); and (iii) interaction budget matched parametric adaptation with GRPO (Shao et al., 2024). Retrieved memory is inserted in the same prompt as the HEXA skill bank for fair comparison; full configurations and details are in Appendix G. Table 1 reports each LLM's ZERO-SHOT solve rate across eight levels of Interphyre. On the two harder levels: `pass_the_parcel` and `catapult`, every model, including the random policy, scores 0%. These levels, where purely parametric reasoning is insufficient, are the primary testbed for our analysis and ablations of HEXA. We report *solve rate* (% of seeds satisfying GOAL_ℓ, Eq. 2) and average turns per seed.

Table 1: Base LLM agent's (ZERO-SHOT baseline) solve rate (%) across eight INTERPHYRE levels. † Pass the Parcel and Catapult evaluated over 50 seeds; all other levels evaluated over 100 seeds.

	Rand	Q-3B	Q-7B	O-20B	O-120B	Sonnet
Cliffhanger	0	0.0	0.0	0.0	0.0	0.0
Tipping Point	3	0.0	3.0	1.0	2.0	0.0
Basket Case	4	0.0	2.0	5.0	13.0	19.0
Down to Earth	8	3.0	7.0	6.0	34.7	68.0
Two Body	4	0.0	0.0	0.0	6.0	14.0
Falling in Place	1	0.0	1.0	3.0	29.0	33.0
Pass the Parcel	0	0.0	0.0	0.0	0.0	0.0†
Catapult	0	0.0	0.0	0.0	0.0	0.0†

3.2 LEARNING FROM EXPERIMENTATION YIELDS IN-CONTEXT SELF-IMPROVEMENT VIA HIERARCHICAL SKILL ACQUISITION

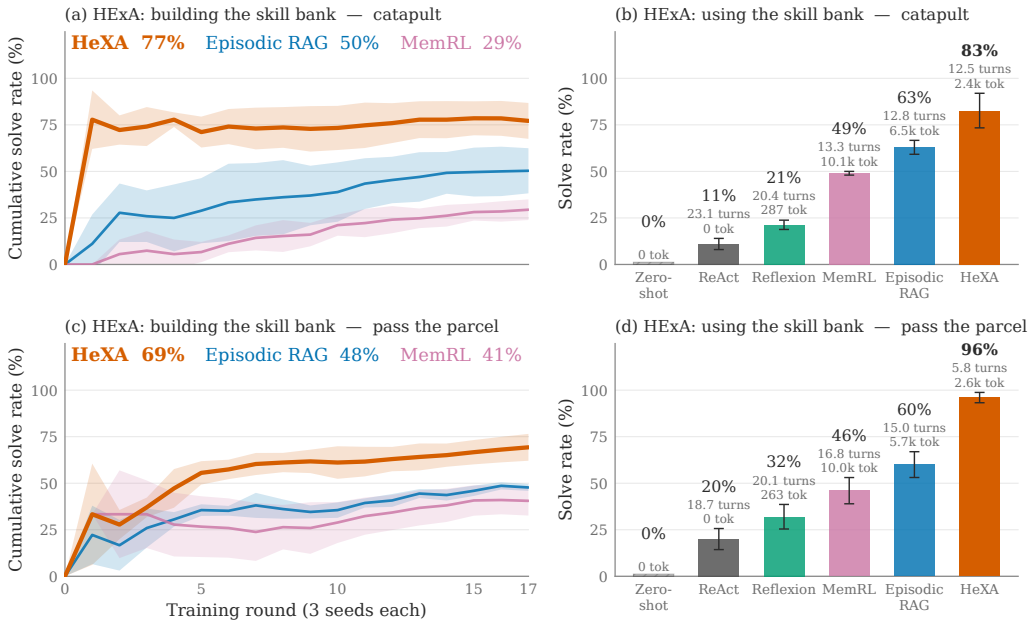
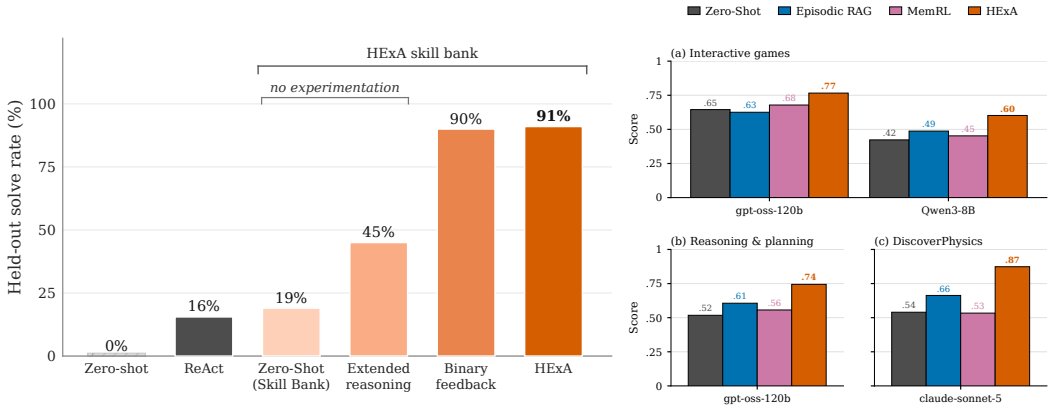


Figure 3: HEXA uses experimentation and learned skills for higher success with greater efficiency. Claude Sonnet 4.6 on `catapult` (top) and `pass_the_parcel` (bottom). *Left:* cumulative solve rate while skills are built over training seeds. *Right:* solve rate on held-out seeds using the final frozen skill bank; labels report mean turns per seed and injected tokens per episode. Curves and bars are means over three runs; bands and error bars denote ±1 s.d. Full results are in Table 8.

Across all benchmark families, the central pattern is that experimentation and interaction experience becomes substantially more useful when it is distilled into a compact reusable skill state rather than repeatedly rediscovered or carried forward as episodic memory. On the hardest INTERPHYRE levels, HEXA substantially outperforms the baselines while zero-shot LLMs, including frontier LLMs mostly fail (Figure 3; Table 8). Importantly, this improvement is not due to a larger inference-time budget: HExA solves more test instances while requiring fewer interaction turns and substantially fewer tokens and injected context than the episodic memory methods (Table 10). The same pattern persists across model scale: with GPT-6-Astra, HEXA reaches 76% and 100% on `pass_the_parcel` and `catapult`, compared with 56% and 92% under REACT, while reducing mean interaction turns per seed by 45% (Figures 15, 16); Figure 3 shows the same qualitative advantage with Claude Sonnet 4.6, and the gains extend to the open-weight models (Table 9).



(a) Importance of experimentation with skill bank **(b)** Other benchmarks results

Figure 4: (a) Effect of evaluation-time feedback. Held-out solve rate for Claude Sonnet 4.6, averaged over `catapult` and `pass_the_parcel`. All orange bars use the same frozen HEXA skill bank; lighter shades indicate less environment feedback. Without feedback, the bank yields a 19% solve rate, rising to 45% with extended reasoning. Providing only binary success or failure feedback raises the solve rate to 90%. Per-level results are in Table 14. **(b) HEXA improves frozen actors across all evaluated benchmark families.** Averaged scores across task families of interactive games, reasoning and planning, and DISCOVERPHYSICS; exact metrics differ, full details are in Appendix H.3.

The comparisons also distinguish *learning from interaction* from simply having access to interaction. REACT can experiment on every held-out instance but carries no learned state across instances, while Episodic RAG and MEMRL retain prior experience without compressing it into the same bounded set of reusable principles and failure corrections. HEXA improves over both regimes, supporting the hypothesis that its benefit comes from converting experimental evidence into reusable guidance and stepping-stones for subsequent behavior. Appendix H.2.2 further separates the two roles: the frozen skill bank improves the actor even when simulator feedback is removed, showing that it has learned reusable task structure, but reasoning with the bank alone remains substantially below full HEXA. Thus, skills amortize what has already been learned, while conditioned on skills, continued inference time experimentation further supplies the instance-specific evidence needed to apply and refine that knowledge on a new problem. This effect extends beyond INTERPHYRE: across interactive games, reasoning and planning, and DISCOVERPHYSICS, HEXA improves the corresponding LLM agent in every reported task–actor setting and achieves the highest success rates as shown in Figure 4b. Full results, evaluation setups and details are in Appendices H.1 and H.3.

3.3 INSTANTIATION OF HEXA AND ABLATIONS

The skill bank in HEXA has three orthogonal axes of instantiation (Algorithms 1 and 2): how it is *initialized* before experimentation, how it is *updated* between rounds, and whether we attach interaction reward labels to the skills.

Does skill-bank initialization and update matter? We compare three initialization regimes: *Offline*, which distills from a warm-start batch and then freeze the skill bank; *Off2On*, which uses the same warm start but continues evolving the bank from online interaction; and *Online*, which starts from an empty bank and bootstraps skills purely from the agent's own interactions.

For skill bank updates we compare *Evolution*, where the evolver receives both the accumulated bank and the newest trajectory batch, against *Iterative Replacement*, where it rebuilds the bank from the newest trajectory batch alone. Across these configurations of HExA, as shown in Figure 5 and Appendix Table 11, *Off2On + Evolution* gives the best accuracy–efficiency trade-off is therefore our default HExA configuration. The update comparison is particularly informative: Evolution outperforms Iterative Replacement on both levels. Because round-n trajectories are themselves generated while conditioned on $\mathcal{K}_{n-1}$, retaining the previous bank allows new evidence to refine, delimit, merge, or replace accumulated knowledge explicitly; rebuilding from the latest batch alone discards that explicit state and conditions the update only on the most recent trajectories. Figure 5 further shows that the higher-performing configurations also terminate in fewer turns.

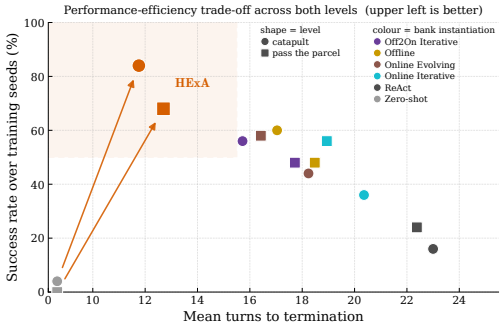


Figure 5: HExA variants accuracy–efficiency comparison. Solve rate vs. mean turns to termination for each skill-bank initialization and update strategy with ablations on Sonnet 4.6. Colour marks the strategy and shape marks the task; the upper left is better. On both levels, Off2On + Evolution is the strategy with highest accuracy and efficiency. Arrows show its gain over ZERO-SHOT. Details in Table 11.

How do reward labels help learning via experimentation? HExA is also flexible to ingest additional experimental evidence such as trajectory rewards, which are a function of outcome reward and efficiency (Appendix C.2), to further guide skill evolution and retrieval. The skill reward r_k is in turn a function of the mean of individual source trajectory rewards ($\bar{r}_{\text{src}}$) that invoked the skill. We define the reward in detail in Appendix H.2. HExA as a framework is agnostic to the choice of trajectory reward and skill reward and can function with no rewards. In our ablation without the reward labels, every success and failure trajectory, and every skill distilled out of them, is treated equally, and the LLM evolver implicitly updates the bank using its existing knowledge rather than environment feedback. Removing the labels costs accuracy as well as efficiency in terms of turns to solve the task on every level we ablated (details in Figure 6, Table 12). The reward labels thus provide environment-grounded feedback from the actor's own exploration that helps assign credit for effective and efficient skill acquisition, and better guides the evolver in composing, retaining or pruning skills.

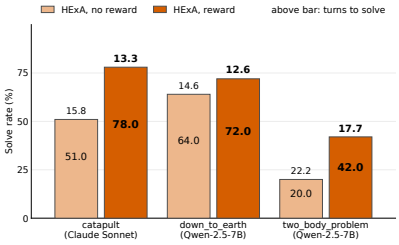
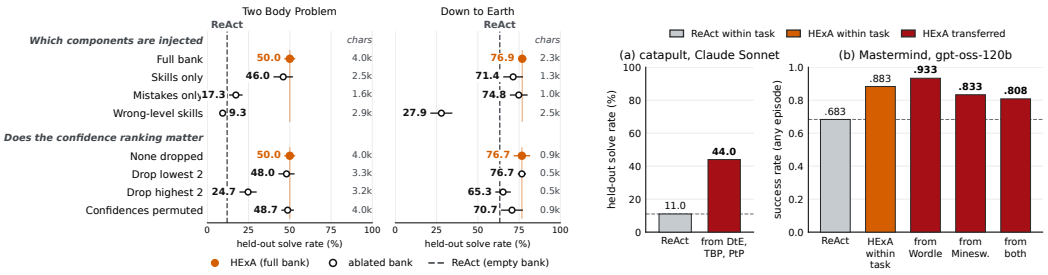


Figure 6: Solve rate (%) per level; the number above each bar is the mean turns to solve. The *no reward* bar treats every success and failure identically, so the evolver prunes from its own prior rather than from environment feedback. Details in Table 12.

How does skill bank content affect results? Frozen-bank controls rule out the possibility that HExA benefits merely from adding more or plausible-looking text to the prompt (Figure 7A). On `two_body_problem`, replacing task-specific skills with comparable-length wrong-level skills reduces success from 50.0% to 9.3%, near the empty-bank REACT baseline (12.0%); removing either skills or mistakes also degrades performance. The learned ranking is likewise informative: on `down_to_earth`, dropping the two highest-reward skills lowers success from 76.7% to 65.3%, while dropping the two lowest leaves it unchanged; permuting rewards also hurts despite retaining every skill. Thus, the gain depends on task-relevant distilled content and its learned prioritization, rather than context length alone. Full component, ranking, and context-length controls are reported in Appendix H.2.1. Thus, our default experimental setting for HExA is Off2On + Evolution with reward labeled skill bank having strategy skills, mistakes as well as general principal skills.

3.4 HIERARCHICAL SKILLS HELP ZERO-SHOT GENERALIZATION TO UNSEEN LEVELS

HExA's hierarchical structure arises because later trajectories are generated while conditioned on previously learned skills, allowing subsequent evidence to refine, delimit, combine, or replace earlier rules rather than accumulating as isolated observations. The transfer experiments test whether these abstractions remain useful beyond the task on which they were learned. The synthesized `catapult` bank makes this composition explicit: its 9 strategy skills contain 22 references to source-level skills (2.44 per synthesized skill), while its 4 mistake rules contain 9 source references (2.25 per synthesized



(A) Skill bank content ablations **(B)** Zero-shot transfer

Figure 7: (A) How the skill bank content drives the gains. *Top:* removing skills or mistakes hurts, while unnecessary wrong-level skills hurt and go below REACT baseline. *Bottom:* removing high-reward skills hurts substantially more than removing low-reward ones; permuting rewards also degrades performance. **(B) HExA on transferred hierarchical skill banks improve zero-shot performance on target-tasks.** *(a)* On `catapult`, skills synthesized from three easier levels raise Claude Sonnet to 44%. *(b)* On Mastermind, transferred banks can even improve over within task HExA reaching 93%. Full results are in Table 19.

mistake), details in Appendix H.4.1. The transfer evolver combines lower-level knowledge about impact direction, drop height, mass, geometric clearance, and failure recovery into target-grounded mechanism-level skills rather than overfitting source-specific settings. As show in Figure 7B, these abstractions transfer purely zero-shot without any target-task trajectories or interactions: combining three easier INTERPHYRE source skill banks raises Claude Sonnet performance on `catapult` from 11% to 44.0%. Transfer gains also appear on other INTERPHYRE levels and across Mastermind, Maze, and DISCOVERPHYSICS. On Mastermind, HExA with the transferred bank even beats within-task HExA. Full transfer results and the provenance analysis are given in Section C.6 and Table 19 in Appendix H.4. These results highlight how in-context learning via experimentation, enables hierarchical skill acquisition through HExA's skill evolution methods and also shows the necessity of learning reusable and transferable knowledge as compact skills as a step towards improving generalization to unseen complex domains.

3.5 IN-CONTEXT POLICY IMPROVEMENT VERSUS PARAMETRIC POST-TRAINING

To compare HExA's non-parametric adaptation with RL, we post-train Qwen-2.5-3B using GRPO on INTERPHYRE tasks (Figure 20). In the match interaction budget setup, HExA reaches 23.3% on `down_to_earth` and 12.0% on `two_body_problem`, while GRPO reaches 20% and 6% after its first training epoch. With continued training, GRPO subsequently matches HExA scores at 3.2 and 1.5 times more interaction budget respectively. The comparison highlights complementary adaptation timescales: HExA amortizes exploration and makes newly discovered strategies immediately available through in-context learning, whereas gradient-based RL can consolidate experience into model weights through further optimization. Full training curves and details are in Appendix H.5.

4 CONCLUSION, LIMITATIONS, & FUTURE WORKS

We introduced HExA, a non-parametric in-context RL framework in which a frozen LLM designs experiments, distills interaction experience into hierarchical skills, and reuses them across tasks without offline data or external supervision. To evaluate when active experimentation is necessary, we also introduced INTERPHYRE, a suite of hard physical reasoning tasks with tool APIs for scene inspection, intervention, and hypothesis testing. Across INTERPHYRE, DISCOVERPHYSICS, multi-step reasoning, and interactive games, HExA outperforms agentic and memory baselines with fewer turns and tokens. On the hardest levels, frontier models such as GPT-6-Astra fail zero-shot, HExA reaches up to 100% success, and its skills transfer zero-shot to unseen harder levels. At matched budget of interactions, it also outperforms GRPO, suggesting that reusing discovered strategies through context amortizes exploration more efficiently than weight updates when data is scarce.

Limitations and Future Work. Skill formation eventually saturates, and additional test-time compute alone does not help. Skill retrieval also relies on simple engineered rewards. Promising directions include unsupervised environment design to generate more diverse learning setups, interleaving skill learning with weight updates for continual learning, and automating reward design and skill retrieval.

AI USE STATEMENT

AI assistants were used to help with refinement of paper writing, creating plots, and writing code. We authors verified the data, analyses, and claims.

REFERENCES

Ossama Ahmed, Frederik Träuble, Anirudh Goyal, et al. CausalWorld: A robotic manipulation benchmark for causal structure and transfer learning. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2010.04296. 32

Kelsey R. Allen, Kevin A. Smith, and Joshua B. Tenenbaum. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 117(47):29302–29310, 2020. URL https://arxiv.org/abs/1907.09620. 32

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026. 20

Anthropic. Equipping agents for the real world with agent skills. 2025a. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills. 20

Anthropic. Introduction to agent skills. 2025b. URL https://anthropic.skilljar.com/introduction-to-agent-skills. 20

Anthropic. System card: Claude opus 4.6. 2026. URL https://www-cdn.anthropic.com/0dd865075ad3132672ee0ab40b05a53f14cf5288.pdf. 1

Tayfun Ates, Muhammed Ali Atesoglu, Cansu Yigit, et al. CRAFT: A benchmark for causal reasoning about forces and interactions. In *Findings of the Association for Computational Linguistics: ACL 2022*, 2022. URL https://aclanthology.org/2022.findings-acl.205/. 32

Anton Bakhtin, Laurens van der Maaten, Justin Johnson, Laura Gustafson, and Ross Girshick. PHYRE: A new benchmark for physical reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://arxiv.org/abs/1908.05656. 3, 6, 30, 31, 32, 35

Fabien Baradel, Natalia Neverova, Julien Mille, Greg Mori, and Christian Wolf. CoPhy: Counterfactual learning of physical dynamics. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://arxiv.org/abs/1909.12000. 32

Paul C. Bogdan, Uzay Macar, Neel Nanda, and Arthur Conmy. Thought anchors: Which LLM reasoning steps matter?, 2025. URL https://arxiv.org/abs/2506.19143. 32

Abhranil Chandra. Unifying foundation models with decision making: A path towards autonomous self-improving agents beyond rewards and human supervision. 2025. 1

Abhranil Chandra, Ayush Agrawal, Arian Hosseini, Sebastian Fischmeister, Rishabh Agarwal, Navin Goyal, and Aaron Courville. Shape of thought: When distribution matters more than correctness in reasoning tasks. *arXiv preprint arXiv:2512.22255*, 2025. 4

Yuxuan Chen, Piotr Piękos, Mateusz Ostaszewski, Firas Laakom, and Jürgen Schmidhuber. PhysGym. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets & Benchmarks Track*, 2025. URL https://arxiv.org/abs/2507.15550. 32

Anoop Cherian, Radu Corcodel, Siddarth Jain, and Diego Romeres. LLMPhy: Complex physical reasoning using large language models and world models, 2024. URL https://arxiv.org/abs/2411.08027. 32

Wei Chow, Jiajun Mao, Bowen Li, Daniel Seita, Vitor Guizilini, and Yue Wang. PhysBench: Benchmarking and enhancing vision-language models for physical world understanding. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2501.16411. 32, 35

Caleb Chuck, Supawit Chockchowwat, and Scott Niekum. Hypothesis-driven skill discovery for hierarchical deep reinforcement learning. in 2020 ieee. In *RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5572–5579. 1

Can Demircan, Tankred Saanum, Akshay K. Jagadish, Marcel Binz, and Eric Schulz. Sparse autoencoders reveal temporal difference learning in large language models. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.01280. 32

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, et al. A survey on in-context learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 1107–1128, 2024. 1

Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, Huaibing Xie, Jianglu Hu, Shaolei Wang, Weichao Wang, Yanling Xiao, Yiting Liu, Zenan Xu, Zhen Guo, Pluto Zhou, Tao Gui, Zuxuan Wu, Xipeng Qiu, Qi Zhang, Xuanjing Huang, Yu-Gang Jiang, Di Wang, and Shunyu Yao. CL-Bench: A benchmark for context learning. *arXiv preprint arXiv:2602.03587*, 2026. URL https://arxiv.org/abs/2602.03587. 1, 2

John Foley, Emma Tosch, Kaleigh Clary, and David Jensen. Toybox: Better atari environments for testing reinforcement learning agents, 2018. URL https://arxiv.org/abs/1812.02850. NeurIPS 2018 Workshop on Systems for ML. 31

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. Omni-MATH: A universal olympiad level mathematic benchmark for large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=yaqPf0KAlN. 1

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024. URL https://arxiv.org/abs/2312.10997. 1

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *arXiv preprint arXiv:2507.17746*, 2025. 24, 25

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou,

Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645 (8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z. 20

Zhengfu He, Xuyang Ge, Qiong Tang, et al. Dictionary learning improves patch-free circuit discovery in mechanistic interpretability: A case study on othello-GPT, 2024. URL https://arxiv.org/abs/2402.12201. 32

Edward Hughes, Michael Dennis, Jack Parker-Holder, Feryal Behbahani, Aditi Mavalankar, Yuge Shi, Tom Schaul, and Tim Rocktaschel. Open-endedness is essential for artificial superhuman intelligence. *arXiv preprint arXiv:2406.04268*, 2024. 3

Dongfu Jiang, Yi Lu, Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai Zou, Chao Du, Tianyu Pang, and Wenhu Chen. Verltool: Towards holistic agentic reinforcement learning with tool use, 2025. URL https://arxiv.org/abs/2509.01055. 52

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66. 1

Adam Karvonen. Emergent world models and latent variable estimation in chess-playing language models. In *Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2403.15498. 31, 32

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, et al. In-context reinforcement learning with algorithm distillation. *arXiv preprint arXiv:2210.14215*, 2022. 2

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 6, 49

Hao Li, Chunjiang Mu, Jianhao Chen, Siyue Ren, Zhiyao Cui, Yiqun Zhang, Lei Bai, and Shuyue Hu. Organizing, orchestrating, and benchmarking agent skills at ecosystem scale, 2026a. URL https://arxiv.org/abs/2603.02176. 20

Kenneth Li, Aspen K. Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2210.13382. 31, 32

Shiqian Li, Kewen Wu, Chi Zhang, and Yixin Zhu. I-PHYRE: Interactive physical reasoning. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2312.03009. 31, 32

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Binxu Li, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianyu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han chung Lee. Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026b. URL https://arxiv.org/abs/2602.12670. 20

Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Xin Xu, Tongtong Wu, Kun Wang, Yang Liu, Zhen Bi, Jungang Lou, Yuchen Eleanor Jiang, Hangcheng Zhu, Gang Yu, Haiwen Hong, Longtao

Huang, Hui Xue, Chenxi Wang, Yijun Wang, Zifei Shan, Xi Chen, Zhaopeng Tu, Feiyu Xiong, Xin Xie, Peng Zhang, Zhengke Gui, Lei Liang, Jun Zhou, Chiyu Wu, Jin Shang, Yu Gong, Junyu Lin, Changliang Xu, Hongjie Deng, Wen Zhang, Keyan Ding, Qiang Zhang, Fei Huang, Ningyu Zhang, Jeff Z. Pan, Guilin Qi, Haofen Wang, and Huajun Chen. Skillnet: Create, evaluate, and connect ai skills, 2026. URL https://arxiv.org/abs/2603.04448. 20

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv preprint arXiv:2502.01100*, 2025a. 2

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv preprint arXiv:2502.01100*, 2025b. 5, 40

Xiaofeng Lin, Sirou Zhu, Yilei Chen, Mingyu Chen, Hejian Sang, Ioannis Paschalidis, Zhipeng Wang, Aldo Pacchiano, and Xuezhou Zhang. Scaling in-context online learning capability of llms via cross-episode meta-rl. *arXiv preprint arXiv:2602.04089*, 2026. 5, 39, 94

Zichen Liu, Anya Sims, Keyu Duan, Changyu Chen, Simon Yu, Xiangxin Zhou, Haotian Xu, Shaopan Xiong, Bo Liu, Chenmien Tan, et al. Gem: A gym for agentic llms. *arXiv preprint arXiv:2510.01051*, 2025. 2, 5, 39

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. SKILL0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026. URL https://arxiv.org/abs/2604.02268. 2, 20

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning, 2025. URL https://arxiv.org/abs/2308.08747. 20

Yingwei Ma, Yue Liu, Xinlong Yang, Yanhao Li, Kelin Fu, Yibo Miao, Yuchong Xie, Zhexu Wang, and Shing-Chi Cheung. Scaling coding agents via atomic skills, 2026a. URL https://arxiv.org/abs/2604.05013. 20

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. Skillclaw: Let skills evolve collectively with agentic evolver, 2026b. URL https://arxiv.org/abs/2604.08377. 20

Michael Matthews, Michael Beukman, Chris Lu, and Jakob Foerster. Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.23208. Oral presentation. 31, 32, 34

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. A survey of context engineering for large language models, 2025. URL https://arxiv.org/abs/2507.13334. 1

Amir Moeini, Jiuqi Wang, Jacob Beck, Ethan Blaser, Shimon Whiteson, Rohan Chandra, and Shangtong Zhang. A survey of in-context reinforcement learning. *arXiv preprint arXiv:2502.07978*, 2025. 2, 21

Linas Nasvytis and Judith E Fan. Leveraging speech to identify signatures of insight and transfer in problem solving. *arXiv preprint arXiv:2605.12970*, 2026a. 2

Linas Nasvytis and Judith E. Fan. Leveraging speech to identify signatures of insight and transfer in problem solving. *arXiv preprint arXiv:2605.12970*, 2026b. 5, 39, 40

Linas Nasvytis, Simon Jerome Han, Ben Prystawski, Satchel Grant, Noah D. Goodman, and Judith E. Fan. Core: Contrastive reflection enables rapid improvements in reasoning. *arXiv preprint arXiv:2605.28742*, 2026. 49

Andreas Opedal, Haruki Shirakami, Bernhard Schölkopf, Abulhair Saparov, and Mrinmaya Sachan. Mathgap: Out-of-distribution evaluation on problems with arbitrarily complex proofs. *arXiv preprint arXiv:2410.13502*, 2024. 5, 40

Andreas Opedal, Haruki Shirakami, Bernhard Schölkopf, Abulhair Saparov, and Mrinmaya Sachan. Mathgap: Out-of-distribution evaluation on problems with arbitrarily complex proofs. In *International Conference on Learning Representations*, volume 2025, pp. 100465–100487, 2025. 2

OpenAI. Gpt-5 technical report. 2025. URL https://cdn.openai.com/gpt-5-system-card.pdf. 1

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022. 20

Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, Cambridge, 2nd edition, 2009. 35

Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. Autorefine: From trajectories to reusable expertise for continual llm agent refinement. *arXiv preprint arXiv:2601.22758*, 2026. 20

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: your language model is secretly a reward model. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc. 20

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 1, 20

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 6, 52, 69

Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. Continual learning of large language models: A comprehensive survey. *ACM Computing Surveys*, 58(5):1–42, 2025. 20

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023. 6, 47

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *arXiv preprint arXiv:2506.06941*, 2025. 5, 40

Parshin Shojaee, Iman Mirzadeh, Maxwell Horton, Samy Bengio, Mehrdad Farajtabar, et al. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *Advances in Neural Information Processing Systems*, 38:108018–108059, 2026. 2

Shuzheng Si, Wentao Ma, Haoyu Gao, Yuchuan Wu, Ting-En Lin, Yinpei Dai, Hangyu Li, Rui Yan, Fei Huang, and Yongbin Li. SpokenWOZ: A large-scale speech-text benchmark for spoken task-oriented dialogue agents. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=viktK3nO5b. 20

David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 1:11, 2025. 1

Elizabeth S Spelke and Katherine D Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007. 1

Alex F. Spies, William Edwards, Michael I. Ivanitskiy, et al. Transformers use causal world models in maze-solving tasks. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2412.11867. 32

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998. 20

Adaptive Agent Team, Jakob Bauer, Kate Baumli, Satinder Baveja, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, et al. Human-timescale adaptation in an open-ended task space. *arXiv preprint arXiv:2301.07608*, 2023. 1

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: A family of highly capable multimodal models. arxiv 2023. *arXiv preprint arXiv:2312.11805*, 2024. 1

Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, Nat McAleese, Nathalie Bradley-Schmieg, Nathaniel Wong, Nicolas Porcel, Roberta Raileanu, Steph Hughes-Fitt, Valentin Dalibard, and Wojciech Marian Czarnecki. Open-ended learning leads to generally capable agents, 2021. URL https://arxiv.org/abs/2107.12808. 1, 3

Qwen Team. Qwen3. 5: Towards native multimodal agents. *URL: https://qwen. ai/blog*, 2026. 1

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024. URL https://arxiv.org/abs/2407.17032. 35

Vijay Viswanathan, Shiqi Wang, Devamanyu Hazarika, Chirag Nagpal, Tongshuang Wu, Graham Neubig, and Yuning Mao. Discretizing reward models, 2026. URL https://arxiv.org/abs/2606.21795. 24, 25

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. SkillX: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026a. URL https://arxiv.org/abs/2604.04804. 20

Jiayu Wang, Yifei Ming, Zixuan Ke, Shafiq Joty, Aws Albarghouthi, and Frederic Sala. Skillorchestra: Learning to route agents via skill transfer, 2026b. URL https://arxiv.org/abs/2602.19672. 20

Yi R. Wang, Jiafei Duan, Dieter Fox, and Siddhartha Srinivasa. NEWTON: Are large language models capable of physical reasoning? In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023. URL https://arxiv.org/abs/2310.07018. 32

Zhaoyang Wang, Qianhui Wu, Xuchao Zhang, Chaoyun Zhang, Wenlin Yao, Fazle Elahi Faisal, Baolin Peng, Si Qin, Suman Nath, Qingwei Lin, Chetan Bansal, Dongmei Zhang, Saravan Rajmohan, Jianfeng Gao, and Huaxiu Yao. Webxskill: Skill learning for autonomous web agents, 2026c. URL https://arxiv.org/abs/2604.13318. 20

Matt L. Wiemann, Lindsay M. Smith, Peter Melchior, Siddharth Mishra-Sharma, Andrew Gordon Wilson, Pavel Izmailov, and Carolina Cuesta-Lázaro. Discoverphysics: Benchmarking llms for out-of-the-box scientific thinking, 2026. URL https://arxiv.org/abs/2605.26087. 3, 5, 40

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning, 2026a. URL https://arxiv.org/abs/2602.08234. 20

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026b. 2

Xinrun Xu, Pi Bu, Ye Wang, Börje F Karlsson, Ziming Wang, Tengtao Song, Qi Zhu, Jun Song, Zhiming Ding, and Bo Zheng. Deepphy: Benchmarking agentic vlms on physical reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 34160–34168, 2026. 32

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. AutoSkill: Experience-driven lifelong learning via skill self-evolution. *arXiv preprint arXiv:2603.01145*, 2026. URL https://arxiv.org/abs/2603.01145. 20

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*, 2022. 20

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023a. URL https://arxiv.org/abs/2210.03629. 6, 47

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*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X. 1

Lance Ying, Katherine M Collins, Prafull Sharma, Cedric Colas, Kaiya Ivy Zhao, Adrian Weller, Zenna Tavares, Phillip Isola, Samuel J Gershman, Jacob D Andreas, et al. Assessing adaptive world models in machines with novel games. *arXiv preprint arXiv:2507.12821*, 2025. 1

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. Coevoskills: Self-evolving agent skills via co-evolutionary verification, 2026a. URL https://arxiv.org/abs/2604.01687. 20

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, Bo Tang, and Muning Wen. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory, 2026b. URL https://arxiv.org/abs/2601.03192. 6, 50

Yanming Zheng, Mingyu Yan, Jingzhou Chen, et al. ContPhy: Continuum physical concept learning and reasoning from videos. In *International Conference on Machine Learning (ICML)*, 2024. URL https://proceedings.mlr.press/v235/zheng24l.html. 32

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences, 2020. URL https://arxiv.org/abs/1909.08593. 20

APPENDIX CONTENTS

A Related Works **20**

B More Background and Preliminaries **20**

C Additional Methodological Details for HEXA **22**

C.1 HExA as online non-parametric in-context reinforcement learning 22

C.2 Trajectory Reward 23

C.3 Skill Distillation from Experimental Experience 24

C.4 Skill Bank Construction and Retrieval 25

C.5 Initialization and Update Regimes 25

C.6 Cross-Task Skill Synthesis 26

C.7 Inference with a Learned Skill Bank 28

C.8 Design Principles 29

D Details of InterPhyre: A Testbed for Learning through Experimentation **30**

D.1 Introduction 30

D.2 Related Work 32

D.3 Design 32

D.4 Intervention API 35

D.5 Summary 37

E Other Benchmarks and Task Details **39**

E.1 Interactive Games 39

E.2 Reasoning and Planning Tasks 39

E.3 DISCOVERPHYSICS 40

F Tool Descriptions **43**

F.1 Shared Tools 43

F.2 Tools for Levels 44

G Baseline Implementation Details **46**

G.1 Zero-shot: One-Shot, No-Tool Baseline 46

G.2 ReAct: Inner Loop 47

G.3 Reflexion 47

G.4 Shared Harness for the External-Memory Baselines 48

G.5 Episodic RAG 49

G.6 MemRL 50

G.7 GRPO: Post-Training Baseline with Gradient based Weight-Update 52

G.8 Training and Evaluation Protocol 53

H Details of Experiments and Results **55**
H.1 INTERPHYRE . . . 55
H.2 Instantiation and ablations . . . 57
H.3 Other benchmarks and tasks . . . 63
H.4 Generalization and Transfer of Hierarchical Skills . . . 66
H.5 Non-Parametric HExA against parametric post-training with GRPO . . . 69

I InterPhyre Prompts **71**
I.1 ReAct System Prompts . . . 71
I.2 HExA Evolver Prompts . . . 82
I.3 Reflexion . . . 91
I.4 Episodic RAG . . . 92
I.5 MemRL . . . 92

J Prompts for the Other Benchmarks **94**
J.1 Interactive games: Mastermind and Maze . . . 94
J.2 Reasoning and planning tasks . . . 105
J.3 DISCOVERPHYSICS . . . 112

K Working Example: ReAct vs. HEXA on Level Catapult of InterPhyre **120**
K.1 Sparsified simulation frames (catapult seed 64) . . . 120
K.2 REACT system prompt (catapult seed 64) . . . 120
K.3 REACT trajectory (catapult seed 64, 25 iterations, FAILURE) . . . 121
K.4 Round-17 evolving skill bank used by HEXA . . . 128
K.5 HEXA system prompt (catapult seed 64) . . . 131
K.6 HEXA trajectory (catapult seed 64) . . . 133

L Working Example: Episodic RAG vs. HEXA on Level Pass the Parcel of InterPhyre **135**
L.1 HEXA retrieved skills (pass_the_parcel seed 57) . . . 136
L.2 Episodic RAG trajectory (pass_the_parcel seed 57) . . . 136
L.3 HEXA trajectory (pass_the_parcel seed 57) . . . 137

M Working Example: MemRL vs. HEXA on World Circle of DISCOVERPHYSICS **138**
M.1 Retrieved memory (circle seed 300) . . . 139
M.2 MemRL trajectory (circle seed 300) . . . 139
M.3 HEXA trajectory (circle seed 300) . . . 141

N Working Example: Episodic RAG vs. HEXA on Mastermind with Qwen3-8B **142**
N.1 Retrieved memory (Mastermind instance `59cedcb50e280335`) . . . 142
N.2 Episodic RAG trajectory (3 episodes, never solved) . . . 142
N.3 HEXA trajectory . . . 143

O Working Example: Episodic RAG vs. HExA on MathGAP **144**

O.1 Problem (MathGAP held-out item 22) 144

O.2 Retrieved memory 145

O.3 Episodic RAG reasoning 145

O.4 HExA reasoning 145

A Related Works

Learning from Interaction. LLM agents improve decision-making via *parametric fine-tuning*, such as reinforcement learning Ouyang et al. (2022); Rafailov et al. (2023); Guo et al. (2025), or *non-parametric adaptation*. While parametric updates yield strong performance, they often incur high computational costs and risk catastrophic forgetting or over-specialization Ziegler et al. (2020); Shi et al. (2025); Luo et al. (2025), driving the shift toward in-context agents as a more efficient alternative. HExA, our training-free framework, directly targets this challenge by self-improving LLM agents via in-context experimentation and self-evolution of reusable hierarchical skills.

Skills for LMs. Skills are natural-language modules that capture reusable procedural knowledge for augmenting LMs at inference time (Anthropic, 2025a;b), and skill augmentation has been validated across agent tasks (Si et al., 2023), including coding (Ma et al., 2026a) and web navigation (Wang et al., 2026c). Earlier skill libraries were built largely through human annotation (Li et al., 2026a;b; Liang et al., 2026; Wang et al., 2026b), which is effective but hard to scale in context-learning settings where contexts are long, technical, and domain-specific. Recent work has therefore moved toward automated skill construction. AutoSkill (Yang et al., 2026) extracts reusable behaviors from interaction traces as lifelong, versioned artifacts, while AutoRefine (Qiu et al., 2026) turns agent trajectories into reusable expertise. CoEvoSkills (Zhang et al., 2026a) builds multi-file skill packages through generator–verifier co-evolution; EvoSkill (Alzubi et al., 2026) performs failure-driven refinement into structured skill folders; and SkillX (Wang et al., 2026a) distills agent trajectories into a hierarchical skill knowledge base refined through execution feedback. However, these methods typically rely on external feedback signals, such as execution feedback, ground-truth comparison, or task-completion rewards, to evaluate and improve skill quality (Ma et al., 2026b), which are unavailable in context-learning scenarios without automatic feedback. A separate line of work internalizes skills through weight updates: SKILL0 (Lu et al., 2026) uses in-context reinforcement learning (RL) to absorb skills, while SkillRL (Xia et al., 2026a) builds a skill bank via RL-guided distillation from stronger teacher trajectories. These approaches require parameter access, limiting their use with closed-source models and sacrificing the interpretability of natural-language skill documents. We therefore propose HExA as a self-improving framework that discovers and evolves skills through in-context active experimentation.

B More Background and Preliminaries

Tool-Augmented Large Language Model (LLM) Agents. To interact with complex environments, LLMs are frequently augmented with external tools and programmatic execution capabilities. This allows them to deeply interleave reasoning with acting, enabling them to gather information, execute APIs, or manipulate simulators directly (Yao et al., 2022; Schick et al., 2023). By structuring interactions as a sequence of thoughts, actions, and observations, tool-augmented agents transcend the limitations of static parametric knowledge and can adapt their strategies based on real-time environmental feedback.

Reinforcement Learning and MDPs. We model the interaction between an agent and its environment as a Markov Decision Process (MDP) (Sutton & Barto, 1998), defined by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma, p_0 \rangle$. Here, $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p(s' \mid s, a)$ is the transition dynamics, $r(s, a)$ is the reward function, $p_0(s)$ is the initial state distribution, and $\gamma \in [0, 1)$ is the discount factor. At time t, the agent observes state $S_t \sim p_0$, selects an action $A_t \sim \pi(\cdot \mid S_t)$ according to its policy π, receives a reward R_{t+1}, and transitions to S_{t+1}. This generates a trajectory $\tau_t = (S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t)$.

The core tasks in RL are *policy evaluation* (estimating the expected return $v^\pi(s) = \mathbb{E}[\sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i} \mid S_t = s]$) and *control* (finding an optimal policy to maximize $J(\pi) = \sum_s p_0(s) v^\pi(s)$). By iterating between evaluating a policy and improving it, RL algorithms optimize a parameterized policy π_θ. This optimization can occur via *online pretraining* (interacting directly with the environment) or *offline pretraining* (learning from a static dataset of prior interactions $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}$).

Traditional methods for optimizing LLM policies rely on gradient-based parameter updates. For instance, Group Relative Policy Optimization (GRPO) avoids training a separate critic by using intra-group relative rewards to optimize the policy. For a given query x, the model samples G

responses $\{y^{(1)}, \ldots, y^{(G)}\}$, which are scored to obtain rewards $\{R_1, \ldots, R_G\}$. GRPO computes normalized advantages and updates the policy with a clipped objective:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{x,\{y^{(i)}\}} \left[\frac{1}{G} \sum_{i=1}^{G} \min\left(r_i A_i, \text{clip}(r_i, 1-\epsilon, 1+\epsilon) A_i\right) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right] \tag{6}$$

In-Context Reinforcement Learning (ICRL). To circumvent the need for continuous parameter updates, In-Context Reinforcement Learning (ICRL) conditions the policy on both the current state S_t and an evolving history or context C_t. Here, the action A_t is sampled according to $\pi_\theta(\cdot \mid S_t, C_t)$. The pre-trained weights θ remain frozen, yet the policy can achieve high rewards in test environments that differ from those seen during pre-training. This generalization arises because the forward pass of the neural network effectively implements an RL algorithm that learns from the context C_t. The performance of π_θ generally improves with the length and quality of C_t, a phenomenon referred to as *in-context improvement* (Moeini et al., 2025).

C ADDITIONAL METHODOLOGICAL DETAILS FOR HEXA

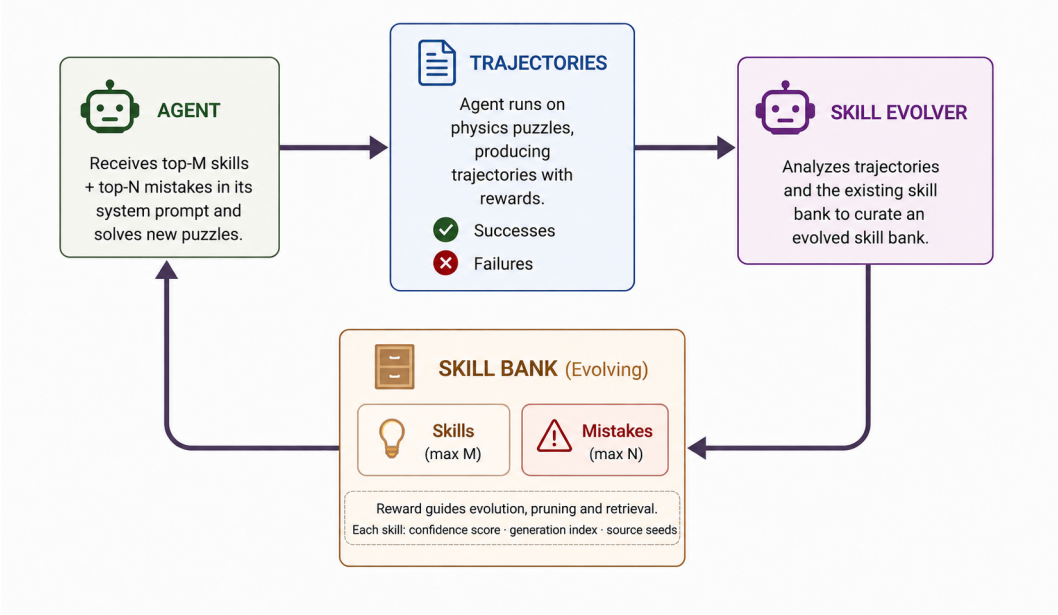


Figure 8: The HEXA actor-evolver loop. The actor receives distilled skills in its system prompt and runs on physics puzzles, producing trajectories annotated with rewards (successes ✓, failures ✗). The Skill Evolver analyzes these trajectories and curates the evolving Skill Bank: skills (max M) are merged, pruned, or promoted by reward signal, and common mistakes (max N) are updated accordingly. Each skill carries a generation index tracking its lineage across evolution rounds (Eq. 17). The updated bank is injected back into the actor's system prompt for the next round, closing the loop.

This appendix expands the HEXA framework introduced in Section 2. We use the notation and problem formulation defined in the main paper and focus here on methodological details omitted there for space: trajectory-level reward construction, the two-stage skill-distillation procedure, skill-bank construction and retrieval, initialization and update regimes, and cross-task skill synthesis. We additionally discuss the design choices that make HEXA applicable without parameter updates or externally supplied skill annotations.

C.1 HEXA AS ONLINE NON-PARAMETRIC IN-CONTEXT REINFORCEMENT LEARNING

In-context reinforcement learning (ICRL) broadly refers to policy improvement that occurs through an agent's context while its deployed model parameters remain fixed. A common approach is to first train or meta-train a sequence model on RL interaction histories so that, at deployment, the model can infer how its behavior should change from the actions, observations, and rewards appearing in its context. HExA shares the goal of policy improvement without test-time parameter updates, but differs in where the learning mechanism and learning state reside.

HExA requires no additional meta-training of the actor to implement a particular RL algorithm. Instead, given a frozen tool-augmented actor π_{act}, it constructs an explicit external learning state $\mathcal{K}$ from the actor's own online interaction experience. For a bank $\mathcal{K}$, define

$$\mathcal{G}_{\mathcal{K},\ell} = \textsc{Retriever}(\mathcal{K}, \ell, M, N), \tag{7}$$

and the effective bank-conditioned policy

$$\pi_{\mathcal{K}}(a_t \mid \ell, s, h_t; \mathcal{F}) := \pi_{\text{act}}(a_t \mid \ell, s, \mathcal{G}_{\mathcal{K},\ell}, h_t; \mathcal{F}). \tag{8}$$

The parameters of π_{act} are fixed, but its effective policy changes as $\mathcal{K}$ changes.

Let $\mathcal{D}_\ell$ denote the distribution of future instances of task family ℓ. We define the utility of a bank as the expected performance of the policy it induces:

$$J_\ell(\mathcal{K}) = \mathbb{E}_{s\sim\mathcal{D}_\ell,\, \tau\sim\pi_{\mathcal{K}}(\cdot|s)}\left[y_\ell(\tau)\right], \tag{9}$$

where $y_\ell(\tau)$ is the task-success predicate defined in Eq. 2. More generally, y_ℓ can be replaced by any task-level return or utility. Conceptually, HExA seeks a bounded external state

$$\mathcal{K}^* \in \arg\max_{\mathcal{K}\in\mathfrak{K}} J_\ell(\mathcal{K}), \tag{10}$$

where $\mathfrak{K}$ denotes the allowed class of bounded skill banks. Unlike gradient-based RL, this optimization does not update π_{act}; improvement is represented entirely by the evolving external state $\mathcal{K}$.

Crucially, HExA does not optimize this objective from a fixed dataset. Its learning data are generated online by the policy induced by the current bank. At round n,

$$\mathcal{G}_n = \textsc{Retriever}(\mathcal{K}_{n-1}, \ell, M, N), \qquad \mathcal{T}^{(n)} \sim \pi_{\mathcal{K}_{n-1}}, \tag{11}$$

after which the evolver constructs

$$\mathcal{K}_n = \pi_{\mathrm{ev}}\left(\mathcal{K}_{n-1}, \mathcal{T}^{(n)}\right). \tag{12}$$

Thus the current learning state determines the actor's subsequent behavior and therefore which evidence is collected next. Bank evolution and data collection form a closed loop:

$$\mathcal{K}_{n-1} \to \pi_{\mathcal{K}_{n-1}} \to \mathcal{T}^{(n)} \to \mathcal{K}_n.$$

In HExA, these trajectories arise from active environment interaction: the actor reasons over hypotheses, chooses tool calls or interventions, observes their consequences, and adapts its subsequent experiments.

The trajectories are also not retained merely as additional demonstrations. The evolver contrasts successful and unsuccessful experience and compresses it into reusable strategy skills, mistake corrections, and partial discoveries. Consequently, both positive and negative experience can change the effective policy: successful trajectories provide evidence for strategies to retain, while failures can identify invalid hypotheses, recurrent failure modes, and correct intermediate discoveries. Because trajectories in round n are generated under skills learned before round n, subsequent skill evolution can refine, compose, delimit, or replace previously acquired knowledge.

We therefore use *online* to refer to the skill-construction phase: experience collection and policy adaptation are interleaved, rather than separated into a fixed offline dataset followed by adaptation. Held-out evaluation is deliberately different: after the final round, $\mathcal{K}_R$ is frozen and no further skill evolution occurs on $\mathcal{S}_\ell^{\mathrm{ev}}$.

Finally, we define overall in-context self-improvement relative to the unconditioned actor as

$$\Delta J_\ell^{\mathrm{HExA}} = J_\ell(\mathcal{K}_R) - J_\ell(\emptyset). \tag{13}$$

For warm-started variants, the contribution of subsequent online evolution can additionally be expressed as

$$\Delta J_\ell^{\mathrm{evolve}} = J_\ell(\mathcal{K}_R) - J_\ell(\mathcal{K}_0). \tag{14}$$

Positive values indicate that experience gathered by the frozen actor has produced an external context state that improves its future effective policy. The trajectory reward $r(\tau)$ used by HExA is an internal credit-assignment signal for constructing this state; it should therefore be distinguished from the task-level objective J_ℓ itself.

C.2 Trajectory Reward

Interaction trajectories differ not only in whether they solve the task, but also in the amount and quality of experimental evidence they provide. A fast successful trajectory provides strong evidence for an effective strategy, whereas a failed trajectory that explores several hypotheses may still contain useful intermediate discoveries and diagnostic information. We therefore associate each trajectory τ with a scalar reward $r(\tau) \in [-1, +1]$ that combines *outcome* and *interaction efficiency*.

For the InterPhyre experiments, we use the following discrete reward categories:

$$r(\tau) = \begin{cases} +1.00 & \text{success},\ t \le 3, \\ +0.75 & \text{success},\ 4 \le t \le 7, \\ +0.50 & \text{success},\ 8 \le t \le 15, \\ +0.25 & \text{success},\ 16 \le t \le T, \\ -0.50 & \text{failure},\ t \ge 10, \\ -0.75 & \text{failure},\ t < 10, \end{cases} \tag{15}$$

where t is the number of interaction turns used before termination and T is the episode budget. Faster successful trajectories therefore receive larger positive rewards. Among failures, trajectories that continue experimenting receive a smaller penalty than trajectories that terminate early, since the former contain more interaction evidence from which the evolver may recover partial skills or diagnose recurring failure modes.

We use discrete rather than finely graded reward values because the reward is consumed by an LLM evolver as evidence about trajectory quality rather than optimized directly through gradient descent. The categories therefore encode qualitatively distinct outcomes—fast success, slower success, extended exploration without success, and early failure—while remaining straightforward for the evolver to contrast (Viswanathan et al., 2026; Gunjal et al., 2025). As shown in Section 3.3 and Table 12, reward labels are helpful but not required: HEXA can also evolve skills without them, leaving trajectory-quality assessment to the evolver itself.

C.3 SKILL DISTILLATION FROM EXPERIMENTAL EXPERIENCE

At round n, the actor produces a batch of reward-tagged trajectories $\mathcal{T}^{(n)}$. We partition this batch as

$$\mathcal{T}^{(n)} = \mathcal{T}^{(n),+} \cup \mathcal{T}^{(n),-},$$

where $\mathcal{T}^{(n),+}$ and $\mathcal{T}^{(n),-}$ denote successful and failed trajectories, respectively. Each trajectory contains the actor's reasoning, tool calls, resulting observations, and trajectory reward. The evolver π_{ev} uses these records to extract complementary forms of reusable knowledge through two stages.

Phase 1: Contrastive strategy extraction. The evolver receives the complete batch and contrasts high-reward trajectories against lower-reward trajectories to identify which strategies, experimental choices, or mechanistic insights distinguish effective behavior from ineffective behavior. Rather than retaining individual action sequences, it summarizes the relevant evidence into a small set of *strategy skills*.

Each strategy skill contains:

- a short *title*;
- a natural-language *principle* summarizing the mechanism or experimental insight;
- a *when-to-apply* condition describing when the principle is relevant;
- an optional demonstrative *example*;
- provenance identifying the *source seed(s)* supporting the skill; and
- a reward-derived score r_k, described in Section C.4.

The evolver is instructed to weight the evidence by trajectory quality: principles supported by efficient successful trajectories receive stronger support than principles inferred from marginal successes or partial evidence. In this way, distillation operates over the *contrast* between successful and unsuccessful experimentation rather than simply summarizing successful episodes.

Phase 2: Mistake and partial-skill extraction. Successful trajectories alone do not expose all useful learning signals. A failed trajectory can contain a correct hypothesis, a useful experimental observation, or a valid intermediate strategy even if a later decision causes the overall episode to fail. We therefore analyze the failed subset $\mathcal{T}^{(n),-}$ separately.

The evolver extracts *mistake records*

$$\mu = (\delta, \rho, \alpha),$$

where δ describes a recurring failure, ρ identifies its inferred root cause or broken causal belief, and α specifies an actionable correction. These records are intended to prevent repeated exploration failures rather than provide positive strategies directly.

The evolver also extracts *partial skills*: correct intermediate reasoning or useful discoveries embedded inside otherwise unsuccessful trajectories. For example, an actor may correctly identify the relevant physical mechanism or a promising parameter region but subsequently choose an ineffective adjustment. Preserving the correct intermediate observation allows later episodes to reuse that discovery without treating the entire failed trajectory as uninformative.

The two stages are complementary. Contrastive strategy extraction captures reusable mechanisms associated with successful behavior, while failure analysis captures recurring mistakes and useful intermediate discoveries that success-only distillation would discard. Because trajectories in later rounds are themselves generated while conditioned on previously distilled skills, newly extracted skills can refine, delimit, combine, or replace earlier ones rather than forming an independent list of episode summaries.

C.4 Skill Bank Construction and Retrieval

The distilled knowledge is stored in an external skill bank $\mathcal{K}$ indexed by task family. The bank contains two collections:

1. $\mathcal{K}^{\mathrm{sk}}$: reusable strategy skills and principles;
2. $\mathcal{K}^{\mathrm{mis}}$: recurring mistake patterns and their corrective actions.

Unlike episodic memory, $\mathcal{K}$ does not retain complete trajectories. Instead, it compresses repeated experimental evidence into reusable abstractions with explicit applicability conditions.

Trajectory grounded skill reward. Each strategy skill k carries a score derived from the trajectories that support it:

$$r_k = \mathrm{clamp}\left(\frac{\bar{r}_{\mathrm{src}} + 1}{2},\, 0.1,\, 1.0\right), \tag{16}$$

where $\bar{r}_{\mathrm{src}}$ is the mean reward of the source trajectories from which the skill was distilled. Skills supported by efficient successful trajectories consequently receive higher scores, whereas skills recovered from weaker or partial evidence can remain in the bank with lower scores. The score is used both to prioritize skills during retrieval and to guide retention and pruning when the bank reaches capacity.

We use discrete reward categories, which are natural for an LLM evolver to interpret (Viswanathan et al., 2026; Gunjal et al., 2025).

Bounded skill bank. To keep inference-time context bounded, the bank contains at most $M_{\max}$ strategy skills and $N_{\max}$ mistake records for each task family:

$$\left|\mathcal{K}^{\mathrm{sk}}\right| \le M_{\max}, \qquad \left|\mathcal{K}^{\mathrm{mis}}\right| \le N_{\max}.$$

The capacity constraint prevents the skill context from growing with the number of episodes and forces the evolver to consolidate, revise, or remove redundant and less useful entries as additional evidence arrives. This distinction is central to HEXA: interaction experience may grow indefinitely, but the representation made available to the actor remains a compact set of distilled skills.

Retrieval and prompt injection. At the start of an episode, the retriever constructs the skill context

$$\mathcal{G} = \textsc{Retriever}(\mathcal{K}, \ell, M, N)$$

by selecting the top-M strategy skills according to r_k together with the top-N mistake records for the current task family. The resulting structured block is prepended to the actor's system prompt alongside the unchanged task description and tool documentation.

The actor is not required to execute a retrieved skill verbatim. Instead, the bank provides reusable hypotheses, applicability conditions, and failure warnings that the actor may use or disregard based on the current instance. Retrieval therefore changes only the model's textual context; it does not modify model parameters, architecture, decoding, or the environment interface. This makes the same mechanism applicable to both open-weight models and API-served black-box LLMs.

C.5 Initialization and Update Regimes

The core loop admits several configurations that differ along two axes: (1) how the initial bank $\mathcal{K}_0$ is constructed, and (2) whether knowledge accumulated in previous rounds is retained explicitly when the bank is updated. These variants let us distinguish the benefit of warm-starting from the benefit of cumulative skill evolution.

Initialization. We consider three regimes.

- **Offline.** A warm-start batch $\mathcal{T}_0$ of x trajectories is collected from the base actor without any skill context and distilled into $\mathcal{K}_0 = \pi_{\text{ev}}(\emptyset, \mathcal{T}_0)$. The resulting bank is then held fixed.
- **Offline-to-Online (Off2On).** The bank is initialized from the same type of skill-free warm-start batch, but subsequent interaction rounds continue to update it using newly collected trajectories.
- **Online.** The bank starts empty, $\mathcal{K}_0 = \emptyset$, and all skills must be bootstrapped from the actor's subsequent online interactions.

Update strategy. For non-static banks, we consider two update rules.

- **Evolution.** The evolver receives both the current bank and the newly collected trajectory batch,
$$\mathcal{K}_n = \pi_{\text{ev}}\left(\mathcal{K}_{n-1}, \mathcal{T}^{(n)}\right),$$
and may add new skills, merge corroborating evidence, revise applicability conditions, or remove entries contradicted by subsequent experience.
- **Iterative Replacement.** The evolver receives only the new trajectory batch,
$$\mathcal{K}_n = \pi_{\text{ev}}\left(\emptyset, \mathcal{T}^{(n)}\right),$$
and reconstructs the bank from that round's trajectories. These trajectories were generated by an actor already conditioned on $\mathcal{K}_{n-1}$, so they may implicitly reflect earlier skills, but the previous bank itself is not supplied to the evolver.

The Offline regime is the corresponding static case, $\mathcal{K}_n = \mathcal{K}_{n-1}$. Crossing initialization and update choices gives the five configurations evaluated in Section 3.3: Offline Static, Off2On Evolution, Off2On Iterative Replacement, Online Evolution, and Online Iterative Replacement.

Evolution is designed to preserve useful accumulated knowledge while incorporating new evidence. This is particularly important because iterative skill learning can otherwise suffer from *skill drift*: repeatedly rebuilding a bank from a small recent batch may discard broadly useful principles in favor of narrow strategies specific to the most recently observed instances. Supplying $\mathcal{K}_{n-1}$ explicitly instead gives the evolver the opportunity to retain a useful rule, refine its boundary conditions, merge it with corroborating evidence, or replace it when later observations contradict it.

Algorithm 2 gives the complete procedure for all initialization and update regimes.

Unless otherwise stated, the configuration referred to as HEXA in our experiments is the reward-guided Off2On + Evolution variant.

C.6 CROSS-TASK SKILL SYNTHESIS

Within-task evolution produces banks specialized to individual task families, but many learned principles need not be tied to the particular entities, coordinates, or parameter values of their source task. Physical mechanisms such as momentum transfer, contact geometry, lever mechanics, and collision order—and more general procedural search strategies—may remain useful when the surface realization of a task changes.

HEXA therefore supports synthesis of a target skill bank from one or more source banks without using target-task trajectories during bank construction. Given evolved source banks
$$\mathcal{K}_{\ell_1}, \ldots, \mathcal{K}_{\ell_S}$$
and a textual description $\text{desc}(\ell^*)$ of an unseen target family ℓ^*, the transfer evolver produces
$$\mathcal{K}_{\ell^*} = \pi_{\text{ev}}^{\text{tr}}\left(\{\mathcal{K}_{\ell_i}\}_{i=1}^{S}, \text{desc}(\ell^*)\right), \qquad \mathcal{T}_{\ell^*} = \emptyset.$$

The synthesis procedure performs four operations:

1. **Select relevant source principles.** The evolver identifies source skills whose underlying physical or procedural mechanisms are structurally relevant to the target.
Algorithm 3 formalizes the cross-task synthesis procedure.

Algorithm 2: HExA: Initialization and Update Variants

Input: Actor π_{act}, evolver π_{ev}, task family ℓ, initialization $\mathcal{I} \in \{\text{empty}, \text{warm-start}\}$, update strategy $\mathcal{U} \in \{\text{evolution}, \text{iterative replacement}\}$, rounds R, episodes per round x, retrieval sizes M and N
Output: Final Evolved skill bank $\mathcal{K}_R$

```
// Initialize the skill bank
if I = warm-start then
    Collect skill-free warm-start batch T_0 of x trajectories from π_act
    K_0 ← π_ev(∅, T_0)
else
    K_0 ← ∅
// Main loop: collect experience and update bank
for n = 1, ..., R do
    // Retrieve top-M skills and top-N mistakes
    G_n ← RETRIEVER(K_{n-1}, ℓ, M, N)
    for i = 1 + x(n − 1), ..., x ∗ n do
        // Actor Prompt ... task + tools + skills
        τ_i ← π_act(· | ℓ, s_i, G_n)
        // outcome × efficiency
        r(τ_i) ← REWARD(τ_i)
    // Update bank based on strategy
    if U = evolution then
        // Evolve: keep old bank, incorporate new evidence
        K_n ← π_ev(K_{n-1}, T^(n))
    else
        // Iterative Replacement: rebuild from scratch each round
        K_n ← π_ev(∅, T^(n))
return K_R
```

Algorithm 3: HExA: Cross-Task Skill Synthesis

Input: Source banks $\mathcal{K}_{\ell_1}, \ldots, \mathcal{K}_{\ell_S}$, target description $\text{desc}(\ell^*)$, transfer evolver $\pi_{\text{ev}}^{\text{tr}}$, retrieval sizes M, N
Output: Synthesized target bank $\mathcal{K}_{\ell^*}$

```
// Invoke transfer evolver on source skill banks and target
   description
K_ℓ* ← π_ev^tr({K_ℓi}_{i=1}^S, desc(ℓ*))
// The evolver performs four operations internally:
// (1) select source skills structurally relevant to target
// (2) remove source-specific parameter and entity bindings
// (3) re-ground each principle in target's entities and tools
// (4) assign transfer score based on correspondence and corroboration
// Prepare synthesized bank for injection or initialization
if zero-shot transfer then
    // Inject directly through retriever for immediate use
    G ← RETRIEVER(K_ℓ*, ℓ*, M, N)
else
    // Use as initialization for within-task evolution
    K_0 ← K_ℓ*
    // Then run standard HExA loop (Algorithm 2)
return K_ℓ*
```

2. **Remove source-specific bindings.** Coordinates, object identities, and other parameterizations that are particular to the source task are stripped away when they are not transferable.

3. **Re-ground the principle.** The retained abstraction is expressed in terms of the entities, constraints, and available tools of the target task.

Algorithm 4: HEXA: Inference with a Learned Skill Bank

Input: Actor π_{act}, evolver π_{ev}, bank $\mathcal{K}^\star \in \{\mathcal{K}_R, \mathcal{K}_{\ell^*}\}$, task family ℓ, evaluation seeds $\mathcal{S}_\ell^{\text{ev}}$, mode $\mathcal{V} \in \{\text{static}, \text{within-seed}\}$, turn budget T, retrieval sizes M, N

Output: Evaluation trajectories and outcomes $\{(\tau_j, y_j)\}_{j=1}^{|\mathcal{S}_\ell^{\text{ev}}|}$

for $j = 1, \ldots, |\mathcal{S}_\ell^{\text{ev}}|$ **do**
  `// Fresh seed-local bank; K* is never modified`
  $\mathcal{K}_{j,0} \leftarrow \mathcal{K}^\star$
  $h_0 \leftarrow \emptyset$
  for $t = 0, \ldots, T-1$ **do**
    `// Retrieve top-M skills and top-N mistakes`
    $\mathcal{G}_t \leftarrow \textsc{Retriever}(\mathcal{K}_{j,t}, \ell, M, N)$
    `// task + tools + skills + history`
    $a_t \leftarrow \pi_{\text{act}}(\cdot \mid \ell, s_j, \mathcal{G}_t, h_t; \mathcal{F})$
    $o_t \leftarrow \textsc{Env}(a_t)$
    $h_{t+1} \leftarrow (h_t, a_t, o_t)$
    if a_t *is terminal* **then**
      break
    `// Update bank based on mode`
    if $\mathcal{V} =$ *within-seed* **then**
      `// Refine from feedback; consumes no turns`
      $\mathcal{K}_{j,t+1} \leftarrow \pi_{\text{ev}}(\mathcal{K}_{j,t}, h_{t+1})$
    else
      `// Static: bank unchanged`
      $\mathcal{K}_{j,t+1} \leftarrow \mathcal{K}_{j,t}$
  `// Score the episode (Eq. 2)`
  $\tau_j \leftarrow h_{T_j}$
  $y_j \leftarrow \mathbb{I}[\textsc{Goal}_\ell(\textsc{Sim}(\tau_j)) = \text{true}]$
return $\{(\tau_j, y_j)\}_{j=1}^{|\mathcal{S}_\ell^{\text{ev}}|}$

4. **Score the transferred skill.** The evolver assigns a transfer score according to the directness of the structural correspondence and the extent to which the principle is corroborated across source banks. The resulting score is used by the same retrieval mechanism that ranks within-task skills.

The evolver is explicitly constrained not to invent target-specific coordinates or solutions that are unsupported by the source banks and textual target description. Transferred entries are instead expressed as general mechanisms together with conditions specifying when they should apply. The purpose of synthesis is therefore to re-ground reusable abstractions rather than copy source-task solutions verbatim. Algorithm 3 formalizes the cross-task synthesis procedure.

The synthesized bank $\mathcal{K}_{\ell^*}$ can be used in two ways. First, it can be injected directly through the standard retriever, allowing the actor to begin interacting with the unseen target using transferred skills even though no target-task trajectory was used to construct the bank. Second, it can serve as $\mathcal{K}_0$ for a subsequent within-task HEXA run, after which target-specific evidence may further refine the transferred knowledge. The full cross-task synthesis prompts are provided in Appendix I.2.5, and the corresponding transfer experiments are reported in Section 3.4.

C.7 Inference with a Learned Skill Bank

Within-task skill evolution (Algorithm 2) returns a frozen bank $\mathcal{K}_R$, and cross-task skill synthesis (Algorithm 3) returns a bank $\mathcal{K}_{\ell^*}$. We write $\mathcal{K}^\star$ for either one. At inference time the actor is run on the held-out evaluation seeds $s_j \in \mathcal{S}_\ell^{\text{ev}}$ with the same budget of at most T turns per seed as during bank construction, and success on each seed is scored by Eq. 2. We consider two inference modes.

Static skill injection. The retriever constructs the skill context $\mathcal{G} = \textsc{Retriever}(\mathcal{K}^\star, \ell, M, N)$ once at the start of the episode, and every turn follows Eq. 1 with this fixed $\mathcal{G}$. The bank is not modified during or between episodes.

Within-seed experimentation. This mode relaxes the assumption in Eq. 1 that $\mathcal{G}$ is held fixed throughout the episode, enabling us to better leverage test-time compute using HExA and continue adapting from inference time experimentation as well. Each seed again starts from $\mathcal{K}^\star$, but after every turn t the evolver receives the current seed-local bank $\mathcal{K}_{j,t}$ and the interaction history h_{t+1} (the actor's tool calls and the resulting observations so far) and refines the bank,

$$\mathcal{K}_{j,t+1} = \pi_{\text{ev}}(\mathcal{K}_{j,t}, h_{t+1}), \qquad \mathcal{K}_{j,0} = \mathcal{K}^\star.$$

The skill context for the next turn is retrieved from the refined bank. In this way the agent can confirm, delimit, or discard a retrieved skill as soon as an experiment contradicts it, and can record instance-specific discoveries (for example, a promising parameter region) for the remaining turns. This inference time refinement does not require extra turns, so both modes have the same budget of at most T environment interactions. Each seed starts from a fresh copy of $\mathcal{K}^\star$ and its refined bank is discarded when the episode ends, so no information passes between evaluation seeds and $\mathcal{K}^\star$ is unchanged. Algorithm 4 gives both modes and our current results are evaluated using static skill injection.

C.8 Design Principles

The preceding components reflect four design properties of HExA.

In-context rather than parametric adaptation. Neither the actor nor the evolver is updated through gradient descent. Adaptation is represented explicitly in the external skill bank and affects behavior only by changing the context supplied to the actor. This allows HExA to operate with frozen and API-served models while avoiding the computational requirements and potential forgetting associated with repeated fine-tuning.

Domain-independent framework. The framework itself does not assume a particular environment or physical domain. The domain-specific inputs are the task description and the tools through which the actor interacts with the environment. The evolver consumes textual interaction trajectories and produces structured natural-language skills. Consequently, the same actor–evolver–retriever abstraction can be instantiated wherever an agent can interact with an environment and convert the resulting observations into reusable experience.

Reward-function flexibility. The particular reward function in Eq. 15 is an instantiation used for our InterPhyre experiments rather than a requirement of the framework. When an informative interaction reward is available, it can provide environment-grounded evidence for skill credit assignment and prioritization. When such feedback is unavailable, the evolver can still construct and update the bank from trajectories without explicit reward labels, as evaluated in our ablations. Thus, reward guidance supplements rather than defines the underlying skill-learning mechanism.

Hierarchical and compositional skill formation. The skill representation is hierarchical in two operational senses. First, distillation abstracts low-level actions, tool calls, and parameter settings into higher-level principles with applicability conditions. Second, later trajectories are produced while the actor already has access to skills learned in earlier rounds. Subsequent distillation can therefore refine an earlier principle, identify its limits, merge complementary observations, or replace a rule contradicted by new evidence. The resulting bank is not an explicit tree; its hierarchy arises through repeated abstraction and composition of experience across interaction rounds. Section 3.4 analyzes this compositional structure and its role in transfer to unseen tasks.

D DETAILS OF INTERPHYRE: A TESTBED FOR LEARNING THROUGH EXPERIMENTATION

Unlike benchmarks that primarily ask whether an agent can solve a task or answer a query, our setting requires an environment in which the agent can deliberately interact and gather evidence: inspecting the current instance, testing interventions, observing intermediate dynamics, diagnosing failures, and reusing what it learned across future seeds and levels in an experimentation loop. INTERPHYRE provides this in a controlled 2D physics domain with continuous actions and long-horizon causal structure. Each task requires placing a red ball at (x, y, r), after which the simulator evaluates whether the level-specific success predicate is achieved. INTERPHYRE builds on the level designs in the PHYRE benchmark (Bakhtin et al., 2019) by introducing APIs for level editing, test-time interventions on environment parameters, and logging traces from the simulation. These APIs can be used to customize the benchmark for evaluating agents on unseen levels, or provided to an LLM for active experimentation. In particular, our experiments use the INTERPHYRE API to expose a structured tool-calling interface for experimentation via interventional scene inspection, partial simulation, full rollouts, contact logs, and level-specific geometric analysis, for example as shown in Figure 14. However, successful actions often depend on contact chains, lever mechanics, collision timing, and strategy-dependent dynamics. Even strong LLMs struggle to anticipate such interactions reliably during inference, particularly when only given the scene description. Now using these intervention API and traces recovered from INTERPHYRE runs, HExA enables the LLM agent to reason over its proposed hypothesis and corresponding experimental observations, than merely guessing solutions from parametric prior knowledge. This lets the agent leverage the information acquired through exploration and interactions, to improve its performance in-context. INTERPHYRE therefore serves as both a benchmark and a diagnostic environment for experimentation based methods such as HEXA.

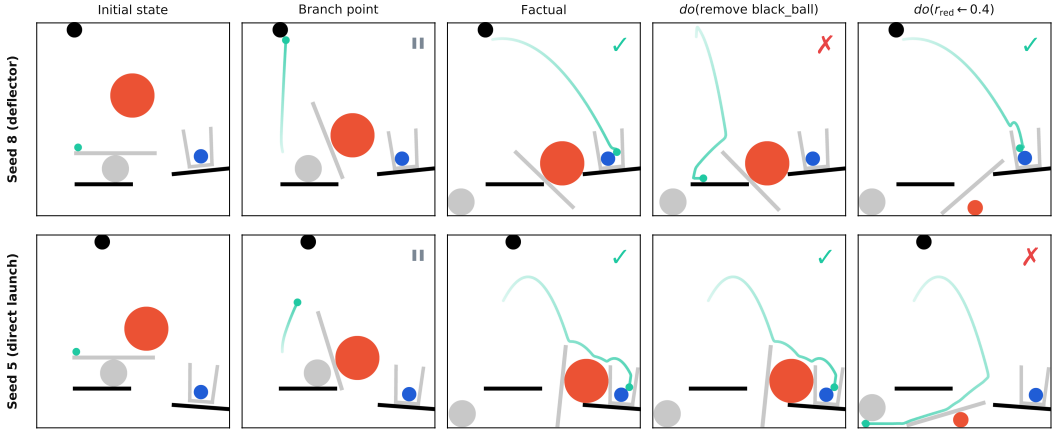


Figure 9: INTERPHYRE's snapshot/restore API. This branches a shared mid-trajectory state into a factual rollout and perturbed alternatives (Section D.4.1). Each row is one seed of the `catapult` level with a different oracle solution strategy. The leftmost column shows the initial scene for reference; the teal curve is the green ball's trajectory, shown up to the branch point in the *Branch point* column and continuing through the outcome in each subsequent column. Top (seed 8, deflector strategy): removing the deflector causes failure; shrinking the red action ball does not. Bottom (seed 5, direct-launch strategy): shrinking the red ball causes failure; removing the deflector does not. The same interventions have opposite causal relevance across seeds, showing that causal relevance is a property of an object relative to the active strategy, not of the object in isolation. See Section D.4.3 for the branching protocol.

D.1 INTRODUCTION

The board-reset primitive that unlocked world-model probing in chess and Othello has no equivalent in continuous physics. Those results relied on models trained on task-domain sequences and on the ability to specify any board state symbolically; physical state is continuous and contact-dependent, and counterfactual access requires rewinding a running simulation rather than resetting a board. We introduce Interphyre, a 2D physics environment whose snapshot/restore API captures complete simulation state at any event-triggered branch point and branches it into paired counterfactual trajectories that diverge under controlled perturbations. In a preliminary probing study, linear probes on Qwen3-8B residual-stream activations predict counterfactual outcomes above chance across

three levels, but only when conditioned on oracle solution strategy. On the catapult level, the same object removal is causally necessary under one strategy and irrelevant under another; pooling across strategies averages over this reversal. The same backend supports RL generalization research: physics parameters are independent experimental axes, so a policy trained under standard conditions can be tested under altered physics without changing scene structure. Interphyre's 25-level curriculum and open Python API are released to support interpretability work on physics-reasoning models and RL generalization research.

Existing 2D physics benchmarks (PHYRE (Bakhtin et al., 2019), I-PHYRE (Li et al., 2024), Kinetix (Matthews et al., 2025)) were designed to answer one question: can the agent solve the puzzle? Two research communities now need more from physics environments. Mechanistic interpretability of physics-reasoning LLMs requires mid-trajectory intervention and paired counterfactual rollouts from a shared physical state; without these, model activations cannot be paired with causal physical ground truth. RL generalization and behavioral analysis require independent control over physics parameters and white-box access to simulation state; without these, a researcher cannot test whether a learned strategy transfers across physics regimes or which scene features it relies on. No existing benchmark meets either requirement.

World-model discovery in board games relies on a property physics does not share: discrete, fully observable ground-truth states that make probe targets unambiguous and interventions exact. Language models encode Othello boards (Li et al., 2023) and chess positions (Karvonen, 2024) as structured world models; those results used models trained on task-domain sequences, where board state was the latent variable the training objective required the model to track. For physics-reasoning LLMs trained on general text, the analogous question is whether chain-of-thought reasoning produces structured physical representations, and answering it requires infrastructure those experiments did not need. Physical state is continuous and contact-dependent; the probe target is the outcome of a specific contact event, not a fixed board position, and counterfactual access requires rewinding a running simulation rather than resetting a board. The infrastructure that made board-game probing tractable does not transfer.

The RL gap has a specific shape. Testing whether a learned strategy generalizes across physics variations, or identifying which scene features a policy relies on, requires varying physics parameters while holding scene topology fixed. Kinetix varies both jointly (Matthews et al., 2025); PHYRE and I-PHYRE expose no parametric control at all. Without independent axes, a generalization result cannot be attributed to physics transfer rather than to a change in scene structure.

We introduce Interphyre, an extensible, intervention-aware 2D physics environment that addresses both gaps through three capabilities:

1. **An intervention API with snapshot/restore semantics.** Any running simulation can be snapshotted at an arbitrary point (on a physics event, a contact trigger, or a fixed timestep) and restored into two or more branches that diverge under controlled perturbations (impulses, forces, parameter changes, object additions or removals). Paired counterfactual evaluation follows: the same agent, on the same scene, under matched and perturbed physical conditions. No existing physical reasoning benchmark provides this programmatic primitive.
2. **Extensible level authoring as Python code.** Levels are Python classes built from a composable object API, with tunable physics parameters (gravity, friction, restitution, density) and user-defined success conditions. Like Toybox's reimplementation of Atari games (Foley et al., 2018), Interphyre reimplements the 2D physics puzzle paradigm established by PHYRE, extending it with white-box access, parametric control, and intervention infrastructure. A curriculum of 25 named levels, each parameterized by a random seed, ships with 250,000 pre-validated task instances; an oracle verification system certifies new seeds on demand, so the curriculum is editable and extensible rather than fixed.
3. **LLM-native interfaces and interpretability-ready data generation.** The simulator and intervention API are exposed as a tool-call surface that an LLM agent can invoke directly. The same interfaces support standalone generation of paired (factual, counterfactual) trajectory data for downstream interpretability and offline reinforcement learning pipelines.

Figure 9 shows the key result: the same structural intervention has opposite causal relevance across seeds that use different solution strategies, a confound any probing study not conditioned on strategy will average over.

D.2 Related Work

Physical reasoning benchmarks. PHYRE (Bakhtin et al., 2019) introduced 2D physics puzzles for evaluating sample-efficient learning, measuring agents by AUCCESS across within-template and cross-template generalization splits. Virtual Tools (Allen et al., 2020) developed a concurrent benchmark with similar Box2D-style mechanics, grounded in a Bayesian "sample, simulate, update" cognitive model. I-PHYRE (Li et al., 2024) extended this to temporally sequenced interventions, where agents eliminate obstacles at precise moments as simulations unfold. I-PHYRE's interventions are agent actions within a single trajectory; the environment produces one modified rollout, and paired counterfactual analysis requires two from a shared prefix. Kinetix (Matthews et al., 2025) takes a different direction: it is a JAX-accelerated physics engine supporting open-ended procedural generation for training general RL agents. Interphyre provides a designed curriculum whose solution structure is known, which is necessary for controlled interpretability comparisons. DeepPHY (Xu et al., 2026) unifies several of these environments under a single evaluation framework for agentic VLMs, but inherits the structural limitations of each backend (no parametric control, no paired counterfactuals, no activation-level instrumentation).

Counterfactual and causal evaluation environments. CoPhy (Baradel et al., 2020) introduced counterfactual learning of physical dynamics, requiring models to predict outcomes after do-interventions on object positions in 3D scenes. CausalWorld (Ahmed et al., 2021) exposed causal variables (masses, sizes, friction, gravity) for do-interventions in robotic manipulation, demonstrating the value of interventional evaluation for transfer learning. CRAFT (Ates et al., 2022) tested causal reasoning about forces and interactions through video question answering over Box2D scenes, including counterfactual question types. ContPhy (Zheng et al., 2024) extended counterfactual physics questions to continuum materials. Each introduces counterfactual or interventional structure into physics evaluation; none implements the programmatic snapshot/restore primitive that generates paired rollouts from a bit-identical physical prefix, which Interphyre's paired-data generation requires.

LLM-based physical reasoning. LLMs underperform humans on physical attribute reasoning (Wang et al., 2023) and struggle with physical dynamics even when strong on common-sense tasks (Chow et al., 2025). These systems evaluate behavior, not representations. PhysGym (Chen et al., 2025) uses LLM agents to actively probe interactive physics simulations and formulate hypotheses, the closest existing paradigm to Interphyre, though PhysGym targets discovery of physical equations rather than intuitive task-solving. LLMPhy (Cherian et al., 2024) demonstrated a simulator-in-the-loop framework where LLMs iteratively estimate physical parameters via code generation. None provides the pairing mechanistic interpretability requires: model activations matched to counterfactual physical outcomes from a shared simulation state.

Interpretability on controlled reasoning tasks. Recent work has discovered world models in Othello (Li et al., 2023), chess (Karvonen, 2024), and mazes (Spies et al., 2025), and circuit structure in board games (He et al., 2024) and in-context RL (Demircan et al., 2025). All these results rely on discrete, fully observable ground-truth states. Physics reasoning does not have this property. Physical state is continuous and high-dimensional; the quantity a probe should predict depends on which contact event matters for the task, not on a fixed board position. Counterfactual access requires rewinding a running simulation to a precise mid-trajectory event, not resetting a board to a saved position. Thought Anchors (Bogdan et al., 2025) identify important reasoning steps via black-box resampling of output traces; Interphyre supports a different analysis, pairing internal activations with causal physical ground truth obtained by intervening on the simulation directly rather than by resampling output traces.

D.3 Design

Four design decisions give Interphyre its research capabilities: levels are Python code (D.3.1), physics parameters are first-class inputs (D.3.2), the curriculum is certified by an oracle (D.3.3), and both RL and LLM interfaces share one simulator backend (D.3.4). 10 shows eight of the 25 canonical levels, spanning gravity-driven drops, collision chains, balancing tasks, and aperture navigation.

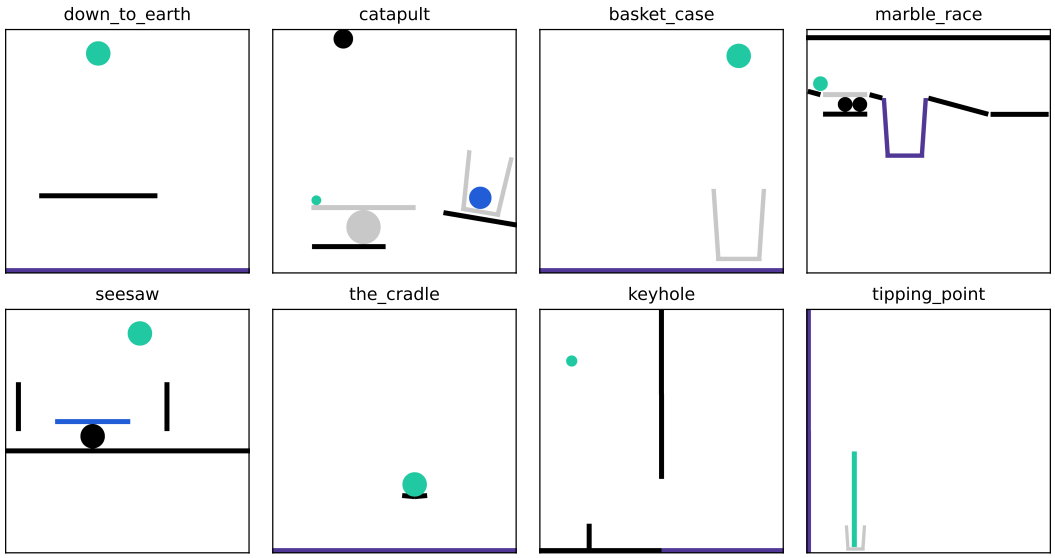


Figure 10: Eight of the 25 canonical levels in the Interphyre curriculum, shown at initial state. The curriculum is hand-authored to span a range of physical phenomena: `down_to_earth` (gravity-driven drop past an obstruction), `catapult` (impulse transfer), `basket_case` (container avoidance), `marble_race` (rolling dynamics along a track), `seesaw` (lever balance), `the_cradle` (pendulum), `keyhole` (aperture navigation), and `tipping_point` (unstable equilibrium). Each level is a Python class; each class is instantiable under a researcher-specified seed and physics configuration.

D.3.1 Levels as Python Code

Encoding levels as Python functions rather than data files means the geometry, action slots, and success conditions are ordinary code that any researcher can fork, modify, and register without rebuilding the package; Listing 1 shows a minimal example.

A researcher studying how behavior depends on scene topology writes a new subclass rather than waiting for a fixed benchmark to include the condition.

D.3.2 Physics Parameters as First-Class Inputs

Physics parameters (gravity, friction, restitution, density, solver step size) are fields on a `SimulationConfig` object passed to the environment constructor, not constants compiled into the

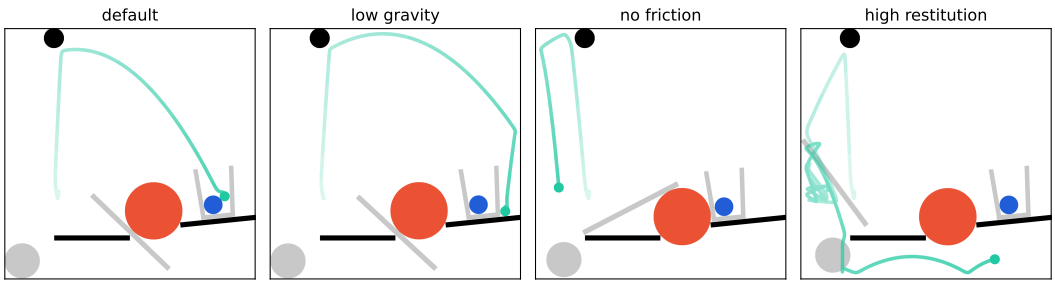


Figure 11: The `catapult` level (seed 8) under four physics configurations, each set by a single `SimulationConfig` field, with the same oracle action placed in every panel. Ghost trails show the green ball's in-flight trajectory. Default physics yields success. Reduced gravity preserves the qualitative mechanism but extends the arc, producing a near miss: the ball clears the basket opening rather than dropping in. Without friction the arm imparts less tangential force and the ball falls short; a higher-energy placement would compensate. With high restitution the elastic collision over-drives the arm and the ball overshoots; a softer placement would keep the launch on target. Holding scene and seed fixed, each parameter change shifts the required action.

```python
from interphyre import Level
from interphyre.levels import register_level
from interphyre.objects import Ball, Bar


def success_condition(engine):
    return engine.is_in_contact_for_duration(
        "green_ball",
        "purple_ground",
        3.0,
    )


@register_level
def build_level(seed=None, variant=0, scene=None) -> Level:
    ground = Bar(
        left=-5,
        right=5,
        y=-4.9,
        thickness=0.2,
        color="purple",
        dynamic=False,
    )

    green = Ball(
        x=0,
        y=4.0,
        radius=0.5,
        color="green",
        dynamic=True,
    )

    red = Ball(
        x=0,
        y=0.0,
        radius=0.4,
        color="red",
        dynamic=True,
    )

    return Level(
        name="falling",
        objects={
            "green_ball": green,
            "red_ball": red,
            "purple_ground": ground,
        },
        action_objects=["red_ball"],
        success_condition=success_condition,
    )
```

Listing 1: A minimal custom level. The geometry, the action slot, and the success condition are all ordinary Python; registering the builder makes it usable by the same evaluation pipeline as the bundled curriculum, with no package rebuild.

engine. The same level, seed, and action can be evaluated under any physically plausible parameter regime by swapping one configuration for another. Figure 11 shows the consequence: the same oracle action succeeds under default physics and fails in three distinct ways under reduced gravity, zero friction, and high restitution, each failure requiring a different corrective action.

Unlike Kinetix, which varies physics parameters jointly with level topology (Matthews et al., 2025), Interphyre keeps scene structure and parameter regime independent; a generalization study holds the level and seed fixed while sweeping gravity or friction.

D.3.3 Curriculum and Oracle Verification

The curriculum ships with 25 canonical levels, each paired with 10,000 pre-validated seeds, for 250,000 pre-certified task instances. Every seed in this distribution is guaranteed to admit at least one solving action placement, verified by an exhaustive oracle search over a discretized action grid with a bounded simulation budget per candidate rather than by stochastic sampling. The verification is decoupled from the environment runtime: a custom level or an extended seed range can be certified offline and the resulting seed list distributed alongside the environment code. Solvability is therefore a guarantee of the curriculum rather than of any particular bundled snapshot.

Oracle verification ensures that, up to the resolution of the action grid, every evaluation failure is attributable to the agent. PHYRE's stochastic-validity heuristic (Bakhtin et al., 2019) admits unsolvable instances, a confound controlled interpretability studies cannot afford; Kinetix trades solvability guarantees for scale, reasonable for general RL pretraining but not for paired-counterfactual evaluation.

D.3.4 Agent Interfaces and Observation Modes

Two interface surfaces share one simulator backend. The gymnasium-compatible interface (Towers et al., 2024) exposes the environment to standard RL training and evaluation code without modification: action spaces, step semantics, and reset behavior follow the conventions of the wider gymnasium ecosystem. The tool-call interface exposes the same environment as a set of named functions (`inspect_scene`, `place_action`, `advance_simulation`, `query_success`) an LLM can invoke directly, so that an LLM reasoner operating under a tool-use harness can drive the simulator in the same turn structure it would use for any other external tool.

Three observation modes expose the same physics to different experimental designs: `physics_state` returns a symbolic vector of positions, velocities, and contacts; `image` renders a frame in either RGB or a discrete-color palette (discrete colors remove the texture and shading confounds that bias VLM evaluation (Chow et al., 2025)); `both` returns both together for multimodal agents. Because all three run the same simulation, observation modality is a controlled variable, not a design constraint.

D.4 Intervention API

Paired counterfactual analysis requires two trajectories that share a bit-identical physical prefix but diverge under exactly one controlled perturbation. Forward rollout cannot produce these: without the ability to hold physical history constant while changing a single variable mid-trajectory, interpretability studies that depend on counterfactual contrast have no substrate. Interphyre supplies this primitive via snapshot/restore, which makes the simulation *causally interrogable*.

D.4.1 Snapshot/Restore and Counterfactual Branching

The environment is deterministic under a fixed seed: replaying the same action sequence on the same level, seed, and physics configuration yields a bit-identical trajectory, which is the property that makes a comparison genuinely paired.

The central primitive is the ability to capture an immutable snapshot of complete physics state at any point during simulation and restore it into two or more independent branches. A snapshot captures the complete Box2D world state: positions, velocities, fixture properties, solver configuration, active contacts, and contact-duration accumulators; the accumulators are required because many levels define success as sustained contact, and restoring position without restoring contact timing would break the pairing.

After `restore`, the simulation evolves identically to the original trajectory unless the researcher introduces a perturbation. This is what makes a comparison *paired*: the two branches share a bit-identical physical prefix, not merely an approximate distributional match. The branching protocol is: snapshot at a physically meaningful event, restore into branch A with no modification, restore into branch B with a controlled perturbation, and compare the divergent outcomes.

In causal terms (Pearl, 2009), the branching protocol produces a matched counterfactual pair: the shared physical prefix is the conditioning history, the perturbation is the treatment whose effect is being measured, and the divergent suffix yields the factual and counterfactual outcomes. What makes the comparison interpretable as evidence of causal effect rather than distributional difference is the bit-identical prefix: the only thing that differs between the two branches is the perturbation itself.

D.4.2 Event-Driven Branch Points

The usefulness of snapshot/restore depends on *where* to branch. Raw step indices are fragile: the step at which a ball contacts a platform varies across seeds, coupling the experimental design to seed-specific timing. Interphyre's declarative trigger language decouples branch points from timing:

Table 2: Core trigger factories. Each factory returns a trigger that the engine evaluates per step; the researcher specifies *what* physical event gates the intervention, not *which step* it corresponds to.

Factory	Fires when	Research use
`on_contact(a, b)`	Objects `a`, `b` touch	Branch at collision to isolate post-contact dynamics
`on_contact_duration(a, b, t)`	Sustained contact $\geq t$ s	Branch near success to test perturbation sensitivity
`on_position_threshold`	Object crosses position	Branch at spatial milestone, e.g., ball clears platform
`on_velocity_threshold`	Speed crosses threshold	Test whether trajectory divergence depends on velocity at branch
`at_step(n)`	Fixed timestep	Uniform temporal sampling for trajectory datasets
`when(predicate)`	Custom predicate on engine	Hypothesis-driven intervention on any state property

the researcher specifies a physical event, and the engine fires automatically when the condition is met. Table 2 lists the core factories.

Triggers compose: `on_sequence` fires an ordered list of triggers in sequence, and `on_any` fires when any child does.

D.4.3 Worked Example: Strategy-Dependent Causal Structure

Figure 9 runs this protocol on two seeds of the `catapult` level whose oracle solutions use different strategies: in seed 8, the green projectile bounces off a black deflector ball to reach the basket; in seed 5, it flies directly in.

Both seeds succeed in the factual branch. The two structural interventions, however, have opposite effects. Removing the black deflector ball causes seed 8 to fail (the deflector was necessary) and leaves seed 5 unaffected (the direct path does not require it). Shrinking the red action ball to radius r=0.4 changes the launch dynamics for seed 5 (failure) while leaving seed 8's deflector arc intact (success). No single intervention explains both outcomes. Each identifies the causal role of one object relative to one strategy.

Causal relevance, these results show, is not a property of an object in isolation. The same `remove_object` call is causally necessary for one strategy and causally irrelevant for the other. A probing study that pools across strategies averages over these opposite effects; if the two strategies occur at similar rates across seeds, the unconditional probe recovers neither.

Figure 12 shows the protocol skeleton. The snapshot is taken at the physical event that gates the experiment: the moment the red action ball contacts the catapult arm. The branch point is 30 steps later, once the green ball is airborne and the two seeds' trajectories have diverged. From that point, three branches diverge under different structural modifications. Section D.4.4 formalizes the split between kinematic and structural perturbations.

D.4.4 Perturbation Primitives as Experimental Treatments

Counterfactual branching requires not just rewinding but *perturbing*, and the space of perturbations determines the space of experimental questions. Kinematic perturbations (impulse, continuous force, velocity override, position override, and freeze) modify the state of existing objects. These support experiments asking whether an outcome depends on a specific kinematic variable (e.g., does a 5% change in post-contact velocity flip the result?). Structural perturbations (adding and removing objects) modify scene composition. These support experiments asking whether an outcome depends on the presence of a particular object (e.g., does removing a supporting surface change the agent's prediction?). The kinematic/structural distinction is the experimental design axis that makes causal findings interpretable: the reversal in 9 is visible only when intervention type is held constant and seed varied.

```python
def run_branch(seed, action, intervene=None):
    env = InterphyreEnv("catapult", seed=seed, enable_interventions=True)
    env.reset(); env.place_action(action)

    snap, _ = env.run_until(
        on_contact("red_ball", "gray_platform")
    )

    env.restore(snap)
    env.step_physics(30)  # branch point: green ball airborne

    if intervene:
        intervene(env)

    return env.run_until(on_success(), max_steps=450)


for seed in [8, 5]:
    action = oracle_action(seed)

    # col 2: factual
    run_branch(seed, action)

    # col 3: remove deflector
    run_branch(
        seed, action,
        lambda e: e.remove_object("black_ball")
    )

    # col 4: shrink red ball
    run_branch(
        seed, action,
        lambda e: shrink_red_ball(e, r=0.4)
    )

    # seed 8: success / failure / success
    # seed 5: success / success / failure
```

Figure 12: Branching protocol for Figure 9 (catapult level, seeds 8 and 5). Each call snapshots at the moment the red ball contacts the catapult arm, advances 30 steps to the branch point, where the green ball is airborne, and then applies the structural intervention. Seed 8: removing the black deflector causes failure; shrinking the red ball does not. Seed 5: shrinking the red ball causes failure; removing the deflector does not. The object's causal role depends on which strategy the seed uses, not on the object itself.

D.5 Summary

Conditioning on oracle solution strategy is necessary to recover probing signal in physics-reasoning models: pooling across strategies averages over opposite causal effects. Linear probes on Qwen3-8B residual-stream activations predict counterfactual outcomes above chance across three levels, but only when indexed to the oracle strategy for each seed. That result suggests probing without strategy conditioning conflates representations from different solution behaviors and understates what models encode about physical causality. The same conditioning requirement applies to all three questions motivating the platform: probing how representations encode physical structure, whether reasoning steps commit to correct physics, and whether those commitments are causally related to task success all require probe targets conditioned on the oracle strategy for each seed. Interphyre's snapshot/restore API makes this tractable: the two branches share a bit-identical physical prefix, so the probe's target is the perturbation's downstream effect rather than a distributional shift in scene statistics.

For RL research, the independent parameter axis is the primary contribution. Fixing scene topology while varying physics parameters isolates physics transfer from structural variation, a confound Kinetix cannot separate. The hand-authored curriculum provides concrete tasks with known solution structure, a substrate for curriculum learning and hypothesis-driven behavioral tests; a researcher can ask whether a strategy learned under one physics regime transfers to another, or whether it depends on properties specific to the training regime.

There are also limitations that motivate future work. The 25-level curriculum is hand-authored and does not scale indefinitely without additional authoring effort, though the oracle verification system applies to any new levels without modification to the runtime. We also see this in experiments where skill banks saturate and more test time compute does not help improve on the levels. So an interesting direction for future work is to integrate unsupervised environment design both to scale level complexity, experimentation and intervention tool design, and also to design unlimited newer

levels. This would enable us to research building agents that continually discover and learn new knowledge and consolidate and store them in an ever-growing repertoire of skills.

E Other Benchmarks and Task Details

InterPhyre is our proposed environment and is documented in full in Appendix D. The other three are popular task families existing in literature, so we describe what a reader needs to interpret our numbers, what an instance is, what the agent observes, what counts as success, and the configuration we ran and defer to the originating papers for everything else.

E.1 Interactive Games

All four games come from the suite of (Lin et al., 2026). Mastermind, Wordle and Minesweeper are Gem environments (Liu et al., 2025) reached through that work's environment adapter; Maze is its own implementation.

Cross-episode protocol. Every game is played under the same multi-episode protocol. A task instance is replayed for T episodes against a fixed total step budget; each episode resets the environment but leaves the hidden task unchanged, and the full interaction history stays in the context window. The agent is told this explicitly in its system prompt. Success is recorded per episode, so episode 1 measures first-attempt competence and episode 3 measures what the agent carried across resets. We use $T{=}3$ throughout, matching the published evaluation.

Mastermind (evaluation task). The agent must identify a hidden ordered code within a fixed number of attempts. Each guess returns structured feedback: black pegs count digits that are correct and in the correct position, white pegs count digits that are correct but misplaced. We use the published evaluation parameters: code length 3, digits drawn from 1 to 6, duplicates disallowed, and 3 turns per episode against a total step cap of 9. This makes the prompt character-identical to the one in the source paper's appendix. All three constructor arguments must be pinned together because the environment's `-random` variant otherwise redraws them at reset, and the outer loop terminates first, collapsing the run to a single episode.

Maze (evaluation task). A randomly initialised grid world with a start cell, walls, and a goal, generated so that the shortest path falls in a pre-specified length range. The agent observes only the four adjacent cells and emits one move per turn, with a cap of 9 steps per episode. An episode succeeds if the goal is reached within the cap. Partial observability is what makes the task cross-episode: the map can only be learned by traversing it.

Wordle and Minesweeper (transfer sources only). Neither is evaluated; both appear only as sources for the cross-task transfer experiments of Section 3.4. In Wordle the agent guesses a hidden five-letter word and receives per-letter feedback (correct and in place, present but misplaced, absent). Our candidate list is the five-letter subset of `en-basic`, 192 words in total; train and evaluation pools are drawn from one deduplicated stream so that disjointness is structural rather than incidental. In Minesweeper the agent reveals or flags cells on a grid with hidden mines, and revealing a safe cell displays the number of mines among its eight neighbours. We pin the opening reveal to a fixed cell in every episode and every arm: the generator excludes mines from the $3{\times}3$ region around the first reveal, so without pinning the layout is chosen by where the agent happens to open and episode 2 is a different board than episode 1 more than 90% of the time — which would contradict the protocol's own premise that the task is identical across episodes.

E.2 Reasoning and Planning Tasks

Four single-turn problem families, each with a generator and a programmatic verifier. We construct every task through a single task factory, so prompt templates, verifiers and per-task defaults have one source of truth, and we pin the arguments in Table 3 explicitly rather than relying on the factory's defaults.

Matchstick provenance. Matchstick arithmetic has no separate benchmark paper. It is a classic domain in the study of human insight problem solving, its specific problem types are adopted from (Nasvytis & Fan, 2026b), and the generator and verifier we run are those released with Core. Each puzzle presents an invalid arithmetic equation in Roman numerals rendered as matchsticks, and the model must move a single stick to make the equation valid.

Table 3: The four reasoning and planning tasks and the arguments we pin. All four use 100 held-out items; our runs use 60 training items.

Task	What an instance is	Pinned arguments
MathGAP (Opedal et al., 2024)	arithmetic word problems with controllable proof structure	none
ZebraLogic (Lin et al., 2025b)	logic-grid puzzles with controllable search complexity	`size_group="xlarge"`
Tower of Hanoi (Shojaee et al., 2025)	valid action sequences for the classic recursive planning problem	`variant="state"`, train disks [5], eval disks [6]
Matchstick (Nasvytis & Fan, 2026b)	move one stick to repair an invalid Roman-numeral equation	default dataset

Split behaviour. Three of the four evaluation sets are invariant to the training-set size: their items come from a fixed file, a fixed shuffle slice, or a disjoint disk count. MathGAP is the exception — its train and evaluation sets are consecutive slices `data[:n]` and `data[n:n+100]` of one shuffled 280-row pool, so raising the training size slides the evaluation window forward. The splits stay disjoint either way, so this is a comparability constraint across regimes, not a contamination risk, and we compare only within a regime.

E.3 DISCOVERPHYSICS

DISCOVERPHYSICS (Wiemann et al., 2026) asks an agent to discover the laws of motion of a simulated world whose physics deliberately deviates from our own, which is what separates discovery from recall of established science. The benchmark contains 22 such worlds (11 public, 11 private); each is generated on demand by an N-body simulator, so there is no static dataset to fit. We use three of the public worlds.

Experimentation through tools. DISCOVERPHYSICS is the second environment in our evaluation, after INTERPHYRE, in which the agent learns by running experiments through a tool interface rather than by reading a fixed input. Each round the agent emits exactly one action, as an XML-tagged tool call:

- `<run_experiment>` takes a JSON list of experiments. Each experiment sets the initial conditions of the particles the world lets the agent control and up to ten measurement times, and may set a total `duration`. The simulator integrates the hidden law and returns noisy positions and velocities of every particle at each requested time. A round may batch several experiments.
- `<final_law>` submits a Python function `discovered_law` that rolls the system forward under the inferred law. It may be accompanied by an optional `fit_parameters()` declaring up to five free parameters with initial values and bounds, which the evaluator fits with `scipy.optimize` against the agent's own trajectories. The submission must also include an `<explanation>` of the world's physics in prose.

Which controls `<run_experiment>` accepts depends on the world's topology (listed per world below). The agent must run at least two rounds of experiments before submitting and is forced to submit in the final round. The benchmark also offers an optional mid-round tool, `<run_mse_fit>`, that fits a candidate law to the data collected so far. It is not enabled in our configuration, so the agent's only feedback before submission is the trajectories it requests.

Scoring. Submissions are scored on two axes. The trajectory axis rolls out the fitted law on noise-free held-out initial conditions and reports mean position error normalised by the variance of the test trajectories (normalised MSE). The explanation axis is an LLM judge comparing the explanation against an expert-written per-world rubric on a 0–10 scale, which we divide by 10. Following the benchmark, every explanation we report is scored by `claude-opus-4-6`, a model different from the actor. An episode passes if normalised MSE is below 0.1 and the explanation score clears a gate. The benchmark's gate is 0.90; we also report 0.75. Our reward, used to rank episodes for distillation, is $0.5\, s_{\text{expl}} + 0.5\, \text{clip}(1 - \text{nMSE}/0.1)$, which lies in $[-0.5, 1]$.

Configuration. We run 10 experimentation rounds per episode, against the benchmark's 16, with observation noise σ=0.075 against its 0.05. A seed sets only the observation noise; the hidden law of a world is fixed. So within-world accumulation learns across noise realisations of one law, while cross-world transfer is the setting in which the instance genuinely changes. Every arm is evaluated on the same 15 held-out seeds per world.

The three worlds we use. `oscillator` is a two-particle world. Particle 1 is a fixed source at the origin and particle 2 is a mobile probe; the agent sets both scalar properties p_1, p_2, the probe's initial position and velocity, the measurement times and the duration. The spatial force is the 2D Poisson $(1/r)$ law, but its coupling varies in absolute time as $G(t) = G_0 \cos(\omega t + \varphi)$, so the interaction periodically reverses sign. Detecting this requires experiments run over a long enough timescale.

`circle` has one centre particle and ten ring particles, all of which source the field, under fractional gravitation derived from a fractional Laplacian $-(-\nabla^2)^\alpha$. The agent controls only two scalars, the ring radius and a tangential velocity shared by every ring particle, plus the measurement times. It cannot place probes, address individual particles or set masses.

`hubble` has 26 particles: a heavy central anchor that is the only field source, 20 ring orbiters whose masses cycle through $\{1, 2, 4\}$, and 5 probes whose positions, velocities and masses the agent sets. A radial, position-dependent Hubble flow $H(r)$ adds a mass-independent outward acceleration on top of the central law. The agent must therefore disentangle the central field-mediated force from a body force supplied by space itself.

`oscillator` and `circle` are within-world accumulation. `hubble` is the cross-world evaluation in Section 3.4: its bank is grown sequentially on nine other public worlds, with `hubble` and `ether`, the two worlds that share its central-law-plus-global-term structure, both held out.

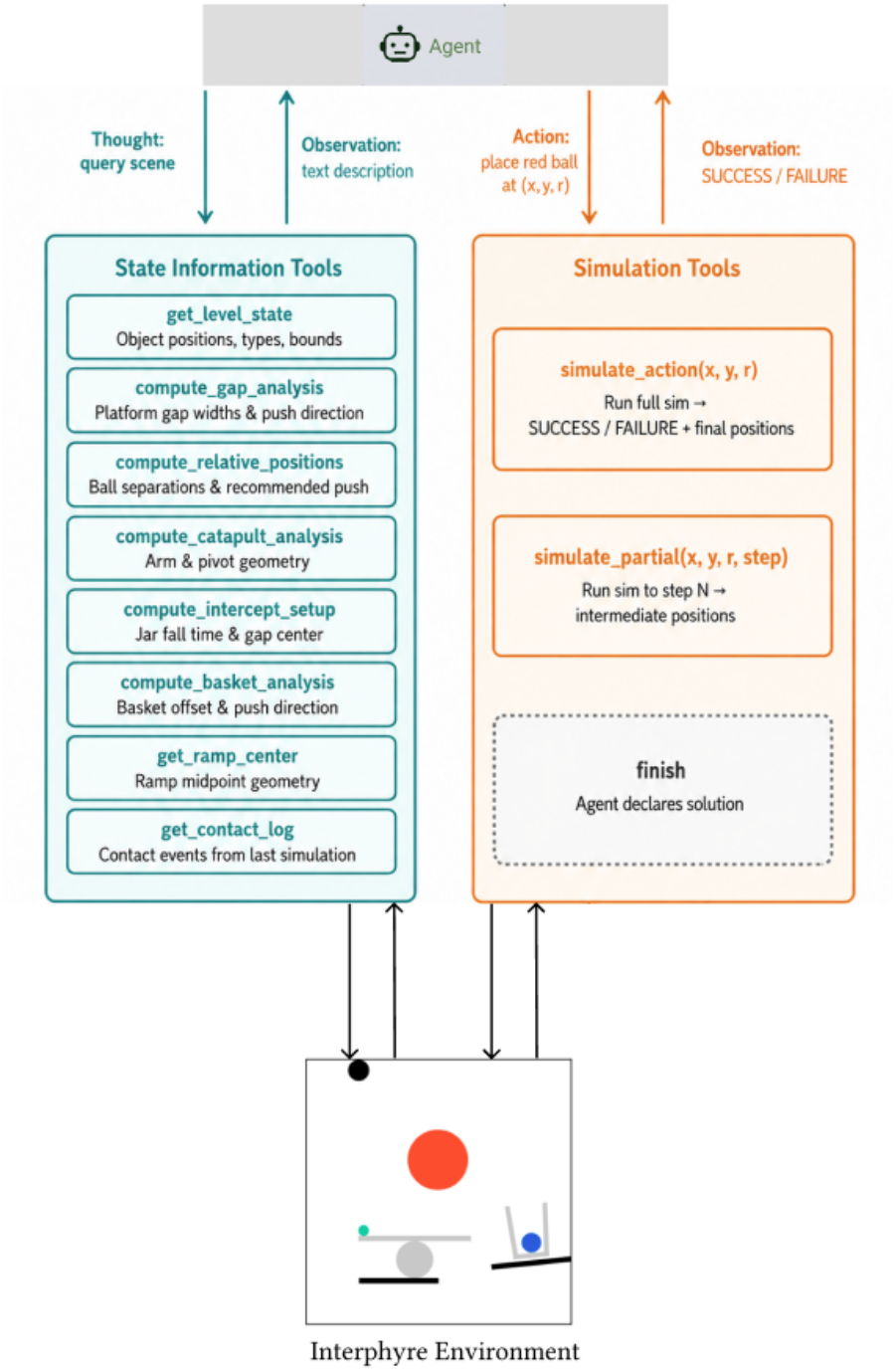


Figure 13: Tool interface between the HEXA agent and the Interphyre environment. The agent issues two classes of tool calls: **State Information Tools** (left) query the scene without advancing the simulation—returning object positions, gap widths, relative ball positions, and level-specific geometry—and **Simulation Tools** (right) place the red ball at (x, y, r) and run the physics engine, returning `SUCCESS`/`FAILURE` with final object positions (`simulate_action`) or intermediate state at a chosen timestep (`simulate_partial`). The `finish` action terminates the episode. Level-specific state tools (`compute_gap_analysis`, `compute_relative_positions`, `get_ramp_center`, etc.) are available only on their respective levels; shared tools are available on all levels. Selected tools presented here. Full tool signatures and usage instructions are provided in Appendix F.

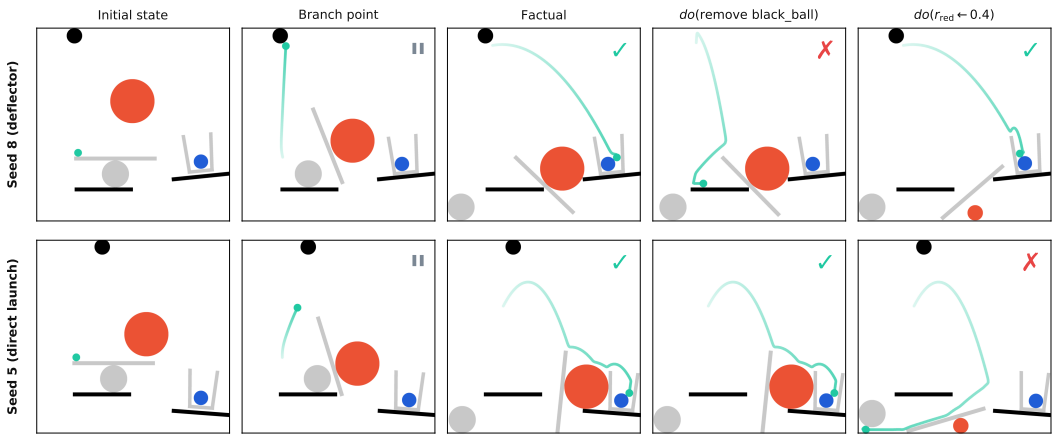


Figure 14: INTERPHYRE's snapshot/restore API. This branches a shared mid-trajectory state into a factual rollout and perturbed alternatives (Section D.4.1). Each row is one seed of the `catapult` level with a different oracle solution strategy. The leftmost column shows the initial scene for reference; the teal curve is the green ball's trajectory, shown up to the branch point in the *Branch point* column and continuing through the outcome in each subsequent column. Top (seed 8, deflector strategy): removing the deflector causes failure; shrinking the red action ball does not. Bottom (seed 5, direct-launch strategy): shrinking the red ball causes failure; removing the deflector does not. The same interventions have opposite causal relevance across seeds, showing that causal relevance is a property of an object relative to the active strategy, not of the object in isolation. See Section D.4.3 for the branching protocol.

F TOOL DESCRIPTIONS

This appendix provides complete descriptions of all tools available to the REACT agent. Tools are partitioned into *shared tools* (available on every level) and *level-specific tools* (exposed only on the corresponding puzzle).

F.1 SHARED TOOLS

`get_level_state()`. Returns a structured text description of the current puzzle scene: the name, position, radius (for balls), dimensions and angle (for bars), and dynamic flag of every object; the world bounds ($x, y \in [-5, 5]$); and the level-specific success condition.

`simulate_action(x, y, radius)`. Places the red ball at (x, y) with the given `radius` $\in [0.1, 2.0]$ and runs the full physics simulation (up to 2,000 steps at 60 Hz) to completion. Returns the outcome (`SUCCESS`/`FAILURE`), total step count, and final positions and velocities of all objects. Before launching the simulation the toolkit performs *pre-simulation validation*: if the placement violates world bounds, overlaps an existing ball, or intersects a platform, the call returns a detailed error specifying the minimum adjustment required rather than consuming a simulation attempt.

`simulate_partial(x, y, radius, stop_step)`. Identical to `simulate_action` but halts after `stop_step` physics steps (or earlier if the success condition is satisfied). Returns object positions and velocities at the chosen checkpoint, enabling mid-simulation diagnostics without committing to a full run.

`get_contact_log()`. Returns timestamped collision events from the most recent simulation: a list of `(step, object_A, object_B)` entries (capped at 20; a notice is appended if more exist), or `"No contact events recorded"` if no contacts occurred. Must be called after `simulate_action` or `simulate_partial`.

`finish(x, y, radius)`. Submits the agent's final answer: places the red ball at (x, y, r) and ends the episode. The episode is scored as a success if the physics engine confirms the goal condition.

F.2 TOOLS FOR LEVELS

F.2.1 DOWN TO EARTH — `COMPUTE_GAP_ANALYSIS()`

Analyses the gaps on each side of the platform that separates the green ball from the floor. Returns:

- platform span (left and right x-coordinates of the platform edges);
- green ball diameter;
- left gap width (distance from left wall to platform left edge) and right gap width (distance from platform right edge to right wall);
- `YES`/`NO` for whether the green ball can fit through each gap;
- a recommendation indicating which side has the larger viable gap, or a `WARNING` if neither gap is wide enough.

F.2.2 TWO BODY PROBLEM — `COMPUTE_RELATIVE_POSITIONS()`

Analyses the spatial relationship between the green and blue balls. Returns:

- position and radius of each ball;
- horizontal separation Δx, vertical separation Δy, and centre-to-centre distance;
- minimum contact distance (sum of radii);
- relative direction of the blue ball from the green ball (`LEFT`/`RIGHT`).

This information guides placement of the red ball to push the green ball into contact with the blue ball.

F.2.3 CATAPULT — FOUR ANALYSIS TOOLS

The Catapult level exposes four complementary tools to handle its multi-body lever mechanics.

`describe_scene_geometry()`. Returns a strategy-neutral geometric inventory of the scene with no prescriptive advice: all balls (position, radius, dynamic flag), all bars (centre, angle, length, dynamic flag), and all baskets (position, dynamic flag); plus the key pairwise distance (green–blue) and the success condition. Intended as a first-pass survey before committing to a simulation strategy. No arguments.

`simulate_with_trace(x, y, radius, object_names, stop_step)`. Places the red ball and runs a full (or truncated) simulation, then returns a per-object kinematic-extrema summary for every name in `object_names` (e.g. `["green_ball", "catapult_arm"]`). The summary per object includes: `peak_y` (maximum height reached), `min_y` (minimum height), `v_max` (peak speed), $\Delta\mathrm{pos}$ (net displacement), and, for bodies that rotate appreciably, peak angular speed $\omega_{\max}$ and angle range $[\theta_{\min}, \theta_{\max}]$. The most relevant contact events involving the red ball or any traced object are also returned (capped at 15). Used to verify lever mechanics and confirm that the green ball reaches the basket.

`trace_green_ball(x, y, radius)`. A lighter trajectory probe than `simulate_with_trace`: only the green ball is sampled. Places the red ball, runs the simulation up to a 600-step ceiling, and returns the green ball's (x, y) waypoints at fixed 30-step intervals together with a start/end/peak summary (start position, end position, Δ, peak y, peak speed). Stops early once the green ball settles (consecutive samples with $|\Delta\mathrm{pos}| < 0.10$) or once the success condition fires. Use this when the only quantity of interest is *where* the green ball travels rather than full per-object kinematics or contact events.

`predict_first_contact(x, y, radius)`. A cheap pre-simulation check (at most 90 physics steps, $\approx$1.5 s of simulated time) that identifies the first object the red ball contacts after release. Returns: placement validity, the name of the first object hit, the step at which contact occurs, approach speed, approximate contact point, and surface normal. Used to verify that the red ball reaches the intended catapult arm before committing to a full simulation.

F.2.4 FALLING INTO PLACE — `COMPUTE_INTERCEPT_SETUP()`

Computes the intercept geometry for the timed-interception puzzle. Returns:

- green ball position and which platform it rests on (`LEFT`/`RIGHT`);
- blue jar position and travel direction the green ball must take;
- the platform edge the green ball must cross and the gap centre x-coordinate;
- estimated time until the jar falls to platform height (in seconds and in 60 Hz physics steps).

F.2.5 BASKET CASE — `COMPUTE_BASKET_ANALYSIS()`

Analyses the spatial relationship between the green ball and the basket that the agent must avoid. Returns:

- green ball position and radius;
- basket position and scale, including opening half-width;
- purple ground position (if present);
- horizontal distance from the green ball to the basket centre;
- recommended push direction (`LEFT`/`RIGHT`) to deflect the green ball away from the basket.

.

F.2.6 PASS THE PARCEL — `GET_RAMP_CENTER()`

Analyses the ramp-and-basket mechanism in the Pass the Parcel puzzle. Returns the centre coordinates (x, y) of the ramp, together with inferred geometry including the top-basket position, bottom-basket position, platform position, ramp angle, and ramp bounds. This information guides placement of the red ball to roll it onto the ramp and interact with downstream mechanisms.

F.2.7 CLIFFHANGER — `COMPUTE_CLIFFHANGER_ANALYSIS()`

Analyses the cliffhanger geometry: a vertical green bar standing on a black platform with a ceiling above and the purple ground below. Returns:

- green bar centre, length, thickness, and angle, plus the (x, y) coordinates of its bottom point (resting on the platform) and top point (opposite end);
- black platform extents (left and right x, centre y, top-surface y, thickness);
- ceiling y and purple-ground y where present;
- the bar's bottom-point distance to the platform's left and right edges;
- closer platform edge (`LEFT`/`RIGHT`) — i.e. the edge the bar must fall past — and the width of the falling gap on that side (between the platform edge and the world wall);
- the implied tip direction the bar's top end must rotate (`LEFT`/`RIGHT`) for the bar's centre of mass to cross the closer edge.

F.2.8 TIPPING POINT — `COMPUTE_TIPPING_POINT_ANALYSIS()`

Analyses the tipping-point geometry: a vertical green bar pinned at its base by a small gray basket, with a static purple wall flush against either the left or right side of the box. Returns:

- green bar centre, length, thickness, and angle, plus the (x, y) coordinates of the free top end and the basket-pinned bottom end;
- gray basket centre (when present);
- purple wall x and its top/bottom y;
- the wall's side relative to the bar (`LEFT`/`RIGHT`);
- horizontal distance from the bar centre to the wall;
- approximate angle the bar must tip through, treated as a rigid stick pivoting at its base, to reach the wall ($\arcsin(\Delta x / L)$ when $\Delta x < L$, else `N/A`);
- suggested tip direction (`LEFT`/`RIGHT`).

G BASELINE IMPLEMENTATION DETAILS

This appendix documents every baseline used in our results for all environments : the ZERO-SHOT no-interaction control, the per-step REACT loop, the Reflexion wrapper (K=2 trials), the two external memory baselines (Episodic RAG and MemRL), and the GRPO fine-tune. We run open weight models using vLLM, and Huggingface on upto 4 A100 GPUs with 80GB VRAM each and the Claude Sonnet model using the Anthropic API. Together with the per-level system prompts (Appendix I) and tool descriptions (Appendix F), the contents below are sufficient to reproduce every baseline number reported in Section 3.

G.1 ZERO-SHOT: ONE-SHOT, NO-TOOL BASELINE

ZERO-SHOT removes the REACT loop entirely. The agent emits a single `Thought` and one `Action`, never iterates, never calls a simulator-write tool of its own, and never observes the consequence of a placement before committing. It is not, however, denied sight of the scene: the harness performs one scripted `get_level_state` call on the model's behalf and embeds the returned object table — exact positions, sizes, dynamic flags and any level-specific geometry — verbatim in the user message, so the model commits with the same numeric view of the initial state that REACT obtains on its first turn. What it lacks is not information about the scene but the ability to act on it and observe the result. It therefore serves as a lower-bound test for how much of each level is solvable from a single static observation, with no interaction.

Per-seed protocol. The harness performs exactly two operations per seed:

1. A single scripted call to `get_level_state` via the same INTERPHYRE toolkit used by REACT. The returned object table (positions, sizes, dynamic flags, and any level-specific geometry) is captured verbatim.
2. One non-streaming Claude Sonnet 4.6 call. The conversation has a level-specific system prompt and a single user message that embeds the scene table from step 1 and demands the final (x, y, r) placement as JSON. The harness parses the JSON, runs `simulate_action` once, and records success/failure.

The model is given no read tools, no probe tools (`simulate_partial`, `get_contact_log`), and no retries. There is one inspection of the scene and one committed placement.

Actor model and decoding. Actor model: Claude Sonnet 4.6 invoked via the `claude -p` CLI with `-max-turns 1` and `-no-session-persistence`, matching the actor used by the REACT and Reflexion baselines on the same level so the comparison isolates the harness, not the model. Temperature 0.3 and `max-new-tokens` 700, identical to REACT.

Answer prompt. The model reasons before answering: the answer block asks for a short `Thought` followed by the committed placement, so every recorded placement is preceded by explicit reasoning. The block appended to the user message is:

```
Submit your FINAL red ball placement now using the finish action.

Output EXACTLY this format and NOTHING else (no preamble, no commentary
outside the Thought line):
Thought: <2-3 short sentences explaining your placement choice>
Action: finish
Action Input: {"x": <X>, "y": <Y>, "radius": <R>}

Replace <X>, <Y>, <R> with your chosen floating-point numbers (subject to
the placement constraints above).
```

The reported base rates are therefore not an artefact of suppressing reasoning. On `catapult` — the level where every model scores 0% in Table 1 — all 50 held-out seeds produced a parseable placement and all 50 failed. Raising the reasoning effort of a frontier reasoning model on the same seeds does not change the outcome: $0/50$ again, this time spending 287K reasoning tokens, on 13 of which the model exhausted its token budget mid-reasoning and never committed to a placement (scored, as always, as failures). One inspection and one committed placement is the binding constraint on this level, not the amount of deliberation before the answer.

System prompt. The system prompt is self-contained and does not reuse the REACT level prompts, so the baseline cannot accidentally inherit ReAct framing. It states the world dimensions, gravity direction, placement constraints, and a per-level *elements* block enumerating the dynamic and static objects with one-line descriptions. The full per-level blurbs are in the released code (`react_agent/run_direct_answer_claude.py`).

Termination and scoring. A seed terminates after the single `simulate_action` call. Success is the same simulator predicate used by every other method (target pair in contact for ≥ 3 s). If the model's response cannot be parsed into a valid (x, y, r) tuple, the seed is recorded as a failure with no simulator call (these are rare: the answer block fixes a single syntactic shape for the placement).

G.2 REACT: INNER LOOP

Every method we evaluate runs the same REACT (Yao et al., 2023a) loop on top of the INTERPHYRE tool API. HEXA adds a skill block to the system prompt; Reflexion adds a reflection block; both leave the inner loop otherwise unchanged.

Per-step protocol. At each iteration the agent emits exactly one block of three lines:

```
Thought: <free-form reasoning over the conversation history>
Action: <tool name, drawn from the level's API>
Action Input: <JSON arguments, or empty for nullary tools>
```

The harness parses these three fields, dispatches the tool against the simulator, and appends the returned text as

```
Observation: <tool output>
```

The agent then produces the next `Thought`/`Action`/`Action Input` block, conditioned on the full conversation so far.

Termination, attempts, and scoring. A trial ends as soon as one of three things happens. (1) The agent emits `Action: finish` with arguments (x, y, r); the harness places the red ball and the simulator runs to completion. (2) The agent emits `Action: simulate_action` with arguments (x, y, r) and the simulator confirms the goal predicate (the named target pair stays in contact for at least 3 seconds, i.e. 180 physics steps at 60 Hz). (3) The iteration count reaches the cap of 25 without either of the above. Each `simulate_action` call counts as a placement attempt; an agent may make up to roughly 24 attempts within a single trial.

Decoding. All baselines share decoding: temperature 0.3, max-new-tokens 700 per agent turn, and a single tool call per turn (parsed greedily from the first `Action` line in the model's reply).

G.3 REFLEXION

Reflexion (Shinn et al., 2023) wraps the same single-trial REACT agent (above) in $K=2$ trials per seed each trial runs for 12 iterations. Trial 1 runs as a standard REACT loop. If trial 1 fails, the harness calls a separate Claude Sonnet instance to write a verbal reflection, prepends the reflection to the system prompt of trial 2, and runs trial 2 from a fresh 25-iteration budget. The episode is scored a success if either trial succeeds; iteration count and wall-clock time aggregate across the two trials.

Reflection model. The reflection step uses Claude Sonnet 4.6 (matching the actor for our `catapult` run), invoked with a single non-streaming completion. The reflection prompt is held fixed across seeds and levels.

Reflection prompt. The reflector receives the full REACT trajectory of the failed trial (`Thought`, `Action`, `Observation` per step) plus the final observation. Long observations are truncated to the first 300 and last 300 characters with an ellipsis to keep the prompt under context. The prompt template is:

```
SYSTEM:
You are analyzing a failed attempt at a 2D physics puzzle.
```

```
You will receive the full Thought/Action/Observation trajectory from a single failed attempt
     by a ReAct-style agent, followed by the final outcome and any prior reflections
     accumulated on this same task.

Your job: produce a short reflection (<=5 sentences) that the agent will read before its next
     attempt on the SAME task. Cover:
(a) which strategy/approach the agent pursued in this attempt;
(b) the specific kinematic, geometric, or procedural reason it failed (cite concrete
     coordinates, distances, or contact events from the observations);
(c) one concrete different action or strategy to try next -- be specific (object names,
     approximate (x, y, radius), expected mechanism).

Hard rules:
- Do NOT repeat lessons that already appear in the prior reflections list.
- Do NOT re-state the goal or the puzzle rules.
- Output ONLY the reflection text. No preamble, no headers, no markdown.

USER:
Outcome: FAILURE.
Final observation:
{final_observation}

Trajectory:
--- Step 1 ---
Thought: {thought}
Action: {action}
Observation: {observation_truncated}
... (one block per step) ...

## Prior reflections (do not repeat these)
1. {prior_reflection_1}
...

Write the reflection now. Be specific and concise (<=5 sentences).
```

Trial 2 conditioning. The returned reflection text is appended to the level's system prompt under a heading `## Reflexion memory` before trial 2 starts. The actor sees this block exactly once at trial 2's first turn, in addition to the unchanged level system prompt and tool list. Trial 2 then proceeds as a fresh REACT loop with the standard 25-iteration cap.

Empty / failed reflections. If trial 1 already succeeds, no reflection is generated and trial 2 is skipped (we still report the seed as a success but the reported iteration count is just trial 1's). If the reflection model returns an empty string (CLI error or empty output), trial 2 runs without a memory block, which makes it equivalent to a second REACT trial from scratch.

G.4 SHARED HARNESS FOR THE EXTERNAL-MEMORY BASELINES

Episodic RAG and MemRL differ only in what they store and how they select it. Everything else below is held identical between them, so any gap between the two is attributable to the memory representation rather than to the harness.

Online episode protocol. For each seed, the harness constructs and embeds the current intent, retrieves a memory block once, and appends that block to the system prompt before the agent begins its at-most-25-turn ReAct trajectory. After the environment returns the episode outcome, the harness first applies the method's feedback update and then writes the new episode to memory. The updated store and retrieval diagnostics are flushed after every seed. Thus memory can change *between* episodes, but retrieval is not repeated inside an episode.

Store. Our port uses a JSON-backed store rather than MemOS or an external vector database. Each record carries the level, seed, outcome class (SUCCESS/FAILURE), intent text, intent embedding, injectable experience text, utility Q, and update count. Retrieval computes cosine similarity over the stored embeddings. Episodic RAG only appends records and never changes Q; MemRL updates Q on the same record schema.

Retrieval key ("intent"). Both baselines key memory on an *intent*: the text that is embedded and compared against past episodes. MemRL calls this z (its ALFWorld runner uses the goal string); Episodic RAG calls it the problem text. The obvious construction — the level description plus the toolkit's state dump — does not work on INTERPHYRE: within a level it is roughly 95%

identical across seeds (same prose, same object names, same success condition, same world bounds, only coordinates differ), and measured over 105 seeds of `catapult` and `pass_the_parcel` its pairwise cosine similarity is 0.9994 ± 0.0003. At that spread MemRL's first retrieval phase is all-or-nothing and its similarity normalization is pure noise. We therefore use a *geometry* intent: only the instance-varying content, i.e. per-object coordinates and extents plus derived relations between the objects the goal predicate names, with no prose and no success condition. This is the honest analogue of MemRL's intent, since in ALFWorld the goal text differs per task while in INTERPHYRE what differs per instance is the geometry. Building it costs no LLM call and no rollout.

Embeddings. The source-faithful Episodic RAG configuration uses the local Jina v2 base English encoder, as in (Nasvytis et al., 2026); the source-faithful MemRL configuration uses OpenAI `text-embedding-3-large`. Text embeddings are cached on disk by (model, text hash), so the current intent is embedded once and the resulting vector is reused when the episode is written back.

Injection point and context accounting. The retrieved block is appended to the agent's system prompt at exactly the position where HEXA appends its `LEARNED PHYSICS SKILLS` block, so the base prompt is byte-identical across methods and only the contents of the injected block differ. Episodic RAG injects the outcome label and raw rollout content. MemRL injects, for each selected record, its outcome, source seed, current Q, update count, and experience text. The intent itself is only the retrieval key and is not repeated in the memory block.

Trajectory rendering. Both baselines render a stored trajectory as its last 12 ReAct steps. A step is a `Thought-Action-Observation` interaction; it is not a physics frame or simulator time step. The observation is the textual return from a tool call (e.g., scene state, placement error, or simulation outcome and final object positions). Each observation is truncated to 600 characters by retaining its head and tail.

G.5 EPISODIC RAG

Episodic RAG builds on retrieval-augmented generation (Lewis et al., 2020). Every completed episode is indexed by its problem intent. For a new problem, the method supplies a semantically nearby successful attempt and a nearby unsuccessful attempt, together with their verifier feedback. It has no learned value function: improvement can only come from adding more examples to the retrieval bank.

Retrieval. $k_{\text{success}} = k_{\text{failure}} = 1$. Candidates are ranked by cosine similarity against the query intent, then split by outcome class and the top entry of each class is taken. If either class is empty, the fallback in the quote above applies and the two most similar rollouts are taken regardless of outcome; the mode actually used is logged per episode

Prompt template. Episodic RAG makes no additional LLM call to summarize, rewrite, or otherwise transform a retrieved episode. The following block is constructed deterministically and appended to the unchanged REACT system prompt. Braced fields are filled at run time, and the rollout body repeats the step template for at most the last 12 steps. When two records are available, the entire `Past rollout` section is repeated once per record.

```
## Retrieved Past Rollouts (from your own past attempts on this level)

### Past rollout {i} -- {SUCCESS|FAILURE} (level {level}, seed {seed})
Level: {level}  |  Seed: {seed}
Verifier: {SUCCESS|FAILURE}
Iterations used: {iterations}
Simulator calls: {simulator_calls}
Shaped reward: {reward}
Final placement: x={x}, y={y}, radius={radius}
Final observation: {final_observation}

Rollout:
(showing last 12 of {total_steps} steps)   [only if truncated]
--- Step 1 ---
Thought: {thought}
Action: {action}
Observation: {observation}
...
```

```
These are complete past attempts with the environment's verdict. Use the successful rollout as
      a template where the geometry matches, and avoid repeating what the unsuccessful rollout
      did.
```

The footer above is reproduced from the implementation. In our port, "complete" means the complete stored representation: as noted below, that representation is capped at the final 12 steps rather than retaining the unbounded original rollout.

Write-back. Every episode is stored with its verifier outcome, iterations, simulator-call count, shaped trajectory reward, final placement, final observation, and raw Thought–Action–Observation trajectory. There is no summarization, utility update, or consolidation, so the bank grows monotonically.

Deviation from the published specification. We keep only the last 12 steps of each rollout rather than all of them. The rollouts are still raw — no model summarizes them — but they are no longer complete. Two reasons: `gpt-oss-120b` must run with eager attention in our setup and the untruncated block exhausts memory on a $4\times$A100 node during the first episode; and holding trajectory length equal to MemRL's is what makes the summarized-versus-raw contrast interpretable. Verbatim storage is recoverable by setting the step cap to zero.

G.6 MEMRL

MemRL (Zhang et al., 2026b) is value-aware episodic memory over (intent, experience, utility) triplets (z, e, Q). We port the official release to the INTERPHYRE episode interface using its **ALFWorld** configuration, the closest analogue among its published settings: sequential, exploration-heavy, and environment-verified. Table 4 lists the resolved values. In our store, (z, e, Q) maps respectively to the record's `intent`, `content`, and `q` fields; only z is embedded.

Phase A: similarity recall. Candidates are scored by cosine similarity against the query intent; those above a threshold δ are kept, capped at k_1. If the admitted set is empty, no memory block is injected at all and the agent falls back on the frozen model, as specified.

Phase B: value-aware selection. The admitted pool is re-ranked by $\text{score}_i = w_{\text{sim}}\tilde{s}_i + w_Q\tilde{Q}_i$ and the top k_2 are injected.

Utility update. Every memory injected during an episode receives a Monte-Carlo update $Q \leftarrow (1-\alpha)Q + \alpha\,(r + \gamma \max Q')$ with $\gamma = 0$, so the target is the terminal reward: $r = +1$ on success and $r = -1$ on failure.

Actor prompt template. The selected records are ordered by the Phase-B score and rendered into the following block, which is appended to the unchanged REACT system prompt. The `experience` field expands to either the successful or failed record format shown immediately below.

```
## Archived Memories (from your own past experiences)

### Memory {i} -- {SUCCESS|FAILURE} (seed {seed}, utility Q={Q}, updates={q_visits})
{experience_i}
...

These experiences were retrieved because they are similar to the current scene and have been
     useful in the past. Adapt them to this seed's geometry rather than copying coordinates
     blindly.
```

For a successful episode, `experience_i` has this form:

```
Task: {level} (seed {seed})

SCRIPT:
{generated_success_script}

TRAJECTORY:
{rendered_last_12_steps}
```

For a failed episode, it has this form:

```
TASK REFLECTION:
Task: {level} (seed {seed})

What went wrong:
{generated_failure_reflection}

Failed approach:
{rendered_last_12_steps}
```

Memory-building prompts. MemRL makes one additional LLM call after each episode to construct the experience text. A successful episode uses the following system prompt at temperature 0.0 on backends that expose temperature:

```
You are analyzing a SUCCESSFUL attempt at a 2D physics placement puzzle.

You will receive the task (level description and scene geometry) and the agent's Thought/
     Action/Observation trajectory.

Produce a concise, high-level script that captures the essential steps and decision points.
     The script must be:
1. Generic enough to apply to similar puzzle instances (different seeds, shifted geometry).
2. Specific enough to provide useful guidance (name objects, give approximate coordinates or
     offsets, state the mechanism).
3. At most 5 high-level steps.
4. Focused on strategy and key decisions, not on individual tool calls.

Output ONLY the numbered script. No preamble, no headers, no markdown fences.
```

A failed episode uses this system prompt at temperature 0.3 when supported:

```
You are analyzing a FAILED attempt at a 2D physics placement puzzle.

You will receive the task (level description and scene geometry) and the agent's Thought/
     Action/Observation trajectory.

This attempt failed. Analyze what went wrong and suggest improvements for future similar tasks.
      Cover:
1. Incorrect assumptions the agent made about the physics or geometry.
2. Steps to improve -- cite concrete coordinates, distances, or contact events from the
     observations.
3. What to avoid next time.

Keep it under 8 sentences. Output ONLY the reflection text. No preamble, no headers, no
     markdown fences.
```

Both calls receive the same user-message template:

```
Task:
{intent}

Trajectory:
{rendered_last_12_steps}
```

For the local/open-weight backend these are true system and user messages. The Claude CLI backend serializes the same content with literal `SYSTEM:`/`USER:`/`ASSISTANT:` role labels; as noted under deviations, that backend does not expose the requested temperature setting.

Write-back. A successful episode is *proceduralized* into a script of at most five high-level steps, stored together with the trajectory; a failed episode is stored as a reflection note plus the failed approach. The summarizer sees the intent and the rendered last-12-step trajectory; it does not generate the retrieval key. An intent whose nearest existing entry of the same outcome class exceeds the add-similarity threshold refreshes that entry in place rather than growing the bank. The update adopts the new content and episode provenance while retaining the old intent, embedding, identifier, Q, and visit count.

Calibration of δ. MemRL's threshold is explicitly dataset-dependent: its appendix determines δ from the pairwise cosine similarity distribution of task descriptions within each benchmark, taking the top-20% quantile, and its similarity normalization uses precomputed dataset statistics rather than per-pool ones. Both therefore have to be *measured* on our intents rather than copied from the ALFWorld configuration.

Table 4: MemRL hyperparameters. Values follow the official ALFWorld configuration except where noted; δ and the similarity normalization constants are dataset-dependent and calibrated per task.

Parameter	Value	Role
k_1	5	Phase-A recall cap
δ	calibrated	Phase-A similarity threshold
k_2	3	memories injected per episode
w_{sim} / w_Q	0.5 / 0.5	Phase-B score weights
ϵ	0.0	random-selection rate
$Q_{\min}$	-10	utility floor
α	0.3	utility learning rate
γ	0.0	discount (terminal reward)
r (success / failure)	$+1$ / -1	episode reward
Q_{init}	0.0	initial utility
add-similarity threshold	0.99†	update-in-place cutoff
novelty threshold	0.85	reward-attribution cutoff
reward merge gain	0.1	attribution scale
Trajectory steps kept	12	shared with Episodic RAG
Observation truncation	600 chars	shared with Episodic RAG

Numeric-geometry retrieval key. We include a labeled numeric-key variant. It parses the fixed-format geometry intent into coordinates, extents, angles, and pairwise offsets; standardizes every feature over the calibration pool; and applies the same cosine retrieval and two-phase MemRL selector. This changes only the retrieval key, not the stored experience, utility update, actor, or interaction budget. It is a documented departure from embedding a natural-language task description.

Deviations from the official release. First, INTERPHYRE uses a geometry-derived intent because the goal sentence is constant within a level. Second, merge-on-write uses 0.99 and same-outcome replacement, rather than the ALFWorld configuration's unrestricted 0.90 match. Finally, with the Claude actor, memory-building calls use the `claude -p` CLI, which exposes no temperature control; consequently the release's 0.0 proceduralization / 0.3 reflection split cannot be reproduced on that backend. Open-weight runs reuse the actor model and honor both temperatures.

G.7 GRPO: POST-TRAINING BASELINE WITH GRADIENT BASED WEIGHT-UPDATE

We use Group Relative Policy Optimization (Shao et al., 2024) to fine-tune Qwen-2.5-3B-Instruct on the same two levels reported in Section H.1 (`down_to_earth`, `two_body_problem`) using the VERL-TOOL framework (Jiang et al., 2025). GRPO is included purely as a sample-efficiency comparison for Section H.5; it is the only method we evaluate that updates model weights.

Setup. Per-step batch size 1, four rollouts per gradient step (n=4), binary $\{0, 1\}$ environment reward (the simulator's success predicate, identical to the scoring rule used by every other method). One *epoch* corresponds to a full pass over the 50 training seeds, i.e. 50 gradient updates. We train for 10 epochs (500 total gradient steps).

Hyperparameters. All runs share the configuration in Table 5; the only difference between the non-skilled and static-skilled variants is `max_prompt_length` (2048 vs. 3072), which is increased to accommodate the skill-bank prefix (≈2200 tokens vs. ≈1400 tokens without skills).

Prompt variants. We evaluate two prompt variants. The *non-skilled* variant uses the level's full system prompt (Appendix I) without any skill block; the *static-skilled* variant pre-pends the same skill bank HEXA ends with after its final training pass, baked in at data-generation time so the GRPO trainer sees a fixed augmented prompt across all rollouts. The static-skilled variant therefore tests whether the skill bank is also useful as a frozen prefix to gradient-based RL.

Results. Table 6 reports greedy val success rate (`mean@1`, 50 val seeds) at each epoch boundary. Figure 20 visualises the same data on a log-scaled x axis to emphasise the low-data region.

Table 5: GRPO hyperparameters shared across all four runs.

Parameter	Value
Model	`Qwen/Qwen2.5-3B-Instruct`
Algorithm	GRPO
Training seeds	1–50 (val: 51–100)
Batch size	1 seed / gradient step
Group size n	4 rollouts per prompt
Total epochs	10 (500 gradient steps)
Learning rate	1×10^{-6}
KL loss coefficient	1×10^{-4} (`low_var_kl`)
Entropy coefficient	1×10^{-3}
Temperature	1.0
Max prompt length	2048 (non-skilled) / 3072 (static-skilled)
Max response length	4096 tokens
Max observation length	512 tokens
Max turns	25
GPU memory utilization	0.3
Observation masking	True (observation tokens excluded from loss)
Hardware	1×A100 80 GB

Table 6: GRPO val success rate (greedy `mean@1`, 50 val seeds) per epoch. Epoch 0 is the pre-training baseline (no gradient updates). *ns* is non-skilled and *sk* static-skilled.

Epoch	DtE (ns)	DtE (sk)	TBP (ns)	TBP (sk)
0	6%	10%	4%	4%
1	20%	6%	6%	0%
2	16%	14%	22%	12%
3	18%	4%	20%	22%
4	54%	38%	26%	100%
5	100%	44%	22%	100%
6	100%	100%	50%	100%
7	100%	100%	56%	100%
8	100%	86%	48%	100%
9	100%	100%	50%	100%
10	100%	100%	40%	100%
HEXA	23.3%		12%	

Compute matching. The figure in Section H.5 marks the matched-sample-budget point at 50 unique seeds (one epoch). At this point, GRPO has performed 50 gradient updates and produced $50 \times 4 = 200$ rollouts; HEXA has generated 51 trajectories total (5 warm-start + 17×3 training). We compare on *unique seeds seen*, not on rollouts or gradient steps, since counting rollouts or gradient steps would artificially advantage the gradient-based method.

G.8 Training and Evaluation Protocol

This subsection states the training and evaluation budget behind every result in Section 3. In every environment the training and evaluation instances are disjoint, the memory-based baselines build their memory on the same training instances as HEXA, and every arm is scored on the same held-out instances. At evaluation the bank (or store) is frozen: nothing is distilled, evolved or written, and no state carries over between held-out instances. Table 7 summarises the budgets.

G.8.1 InterPhyre

Every level's seeds are split once, before any run, into three disjoint blocks that are never mixed:

1. **Warm start, 5 seeds.** Base-actor trajectories distilled into the initial bank $\mathcal{K}_0$ under *Off2On* initialisation. These seeds are never scored.

Table 7: Training and evaluation budget per environment. Warm start is the number of instances played by the base actor with an empty bank and distilled into the initial bank (round 0 in the games and the reasoning tasks); training is the number of instances seen by the remaining rounds of the evolution loop, as rounds × instances per round; evaluation is the number of held-out instances every arm is scored on. Maze counts are for `gpt-oss-120b`. Episodic RAG and MemRL build their memory on the same training instances as HEXA.

Environment	Warm start	Training	Held-out eval
INTERPHYRE (each level)	5	51 (17 × 3)	50
Mastermind	10	70 (7 × 10)	40
Maze	10	190 (19 × 10)	243
Reasoning and planning (each task)	2	58 (29 × 2, 10 attempts)	100
DISCOVERPHYSICS `oscillator`, `circle`	5	15 (5 × 3)	15
Transfer (no target-task trajectories)			
INTERPHYRE cross-level	source-level banks		50
Games cross-task	Wordle 10+120, Minesweeper 10+190		40 / 243
DISCOVERPHYSICS `hubble`	18	24 (8 worlds ×3)	15

2. **Training, 51 seeds.** The evolution loop runs here: 17 rounds of 3 fresh seeds each. Episodic RAG and MemRL build their stores on the same 51 seeds, so every method is given the same budget of experience on the same instances.
3. **Held-out evaluation, 50 seeds.** These seeds, produce the reported solve rates for ZERO-SHOT, REACT, Reflexion (K=2), Episodic RAG, MemRL and HEXA. No method has seen any of them during training.

Seeds are passed directly to the simulator's level generator and determine the randomised positions, masses and dimensions of the level's objects (Appendix D), so the three blocks are disjoint sets of *instances*, not merely disjoint indices.

The round-by-round loop is training, not evaluation. During training, round n attempts its 3 seeds with the bank $\mathcal{K}_{n-1}$ fixed, and only afterwards is the evolver allowed to update the bank from those trajectories; no attempt is ever made with a bank that has been distilled on the seed being attempted. Those per-round solve rates are a *training* diagnostic — they are what Figure 3 plots are computed from them. At evaluation the bank is frozen at the end of round 17.

Two protocols, kept separate. Table 11 (initialisation and update rule) and the reward-free rows of Table 12 report the *training-time* round-by-round rate on the training seeds rather than a frozen held-out rate, because their purpose is to compare how fast different update rules accumulate skill. Their numbers are paired across configurations but are *not* comparable to the held-out rates in Table 8 — which is why, for instance, Off2On Evolving reads 84% there and 82.7% as a frozen bank on the held-out pool. Every other INTERPHYRE number in the paper is a frozen bank on the 50 held-out seeds.

Cross-level transfer. The target bank is synthesised from the HEXA banks of the source levels, each evolved on its own level's warm-start and training seeds, plus a textual description of the target level. It is evaluated on the target level's 50 held-out seeds, the same seeds as that level's REACT baseline. The synthesis never sees a target-level trajectory, so the comparison is zero-shot at the trajectory level.

GRPO. GRPO updates weights and has no bank to freeze, so it uses its own split of 50 training and 50 validation seeds per level (Appendix G.7).

G.8.2 INTERACTIVE GAMES

An instance is one hidden task played for 3 episodes with a cumulative chat (Appendix E.1). One memory record is one three-episode instance, retrieved once at instance start.

Mastermind. The 120 possible codes are partitioned into 80 training and 40 held-out instances. HEXA warm-starts on 10 of the training instances (round 0, played by the base actor with an empty bank) and trains on the remaining 70 in 7 rounds of 10; Episodic RAG and MemRL build on the same

80 instances in batches of 10. Every arm, including the no-memory arm and Reflexion, is scored on the same 40 held-out instances with the bank or store frozen.

Maze. Training and evaluation layouts are drawn independently. HEXA uses 200 training instances. For `gpt-oss-120b` it warm-starts on 10 (round 0) and trains on the remaining 190 in 19 rounds of 10; for Qwen3-8B it warm-starts on 20 and trains on 180 in 9 rounds of 20. Episodic RAG and MemRL build on the same pool after removing duplicate layouts (193 instances). Every arm is scored on the same 243 held-out layouts.

Cross-task transfer. The source banks are HEXA banks trained on the source games: Wordle on 130 instances (13 rounds of 10) and Minesweeper on 200 instances (20 rounds of 10). A target bank is synthesised from one or both source banks plus a textual description of the target game, with no target-game trajectories, and is scored on the target's held-out pool: the same 40 Mastermind instances or 243 Maze layouts as the within-task arms.

G.8.3 REASONING AND PLANNING TASKS

Each of MathGAP, ZebraLogic, Tower of Hanoi and Matchstick uses 60 training items and 100 held-out items (Appendix E.2). HEXA runs 30 rounds of 2 problems with 10 attempts each, i.e. 600 training rollouts in which every training item is visited once. Round 0 is the warm start: its 2 problems are attempted by the base actor with an empty bank, and the remaining 29 rounds train on the other 58 items; a further 300 no-memory calls (5 per training item) estimate the per-item baseline, for a total budget of 900 rollouts. Episodic RAG and MemRL build on the same 60 items with the same 900-rollout budget. Every arm is scored on the same 100 held-out items with the bank or store frozen. We run no transfer experiments on these tasks.

G.8.4 DISCOVERPHYSICS

A seed sets only the observation noise; the hidden law of a world is fixed (Appendix E.3). Within a world the held-out seeds are therefore new noise realisations of the law the bank was trained on, while `hubble` is a world the bank has never seen.

Within-world (`oscillator`, `circle`). The initial bank is distilled from 5 warm-start episodes of the base actor. HEXA then trains for 5 rounds of 3 episodes (15 episodes), and the bank is frozen after round 5. Episodic RAG and MemRL build on the same 5 warm-start and 15 training episodes. Every arm is scored on the same 15 held-out seeds per world.

Cross-world (`hubble`). `hubble` is held out. The bank is initialised from 18 warm-start episodes pooled across five other worlds and grown online on 8 other public worlds with 3 episodes each (24 episodes). Episodic RAG and MemRL build on the same 18 warm-start and 24 training episodes. Every arm is scored on the same 15 held-out `hubble` seeds, and no `hubble` trajectory is seen during training.

H DETAILS OF EXPERIMENTS AND RESULTS

This appendix holds the full results behind Section 3: the per-level and per-task tables that the main-text figures summarise, together with per-episode metrics, iteration counts, injected-token budgets and significance tests.

H.1 INTERPHYRE

Table 8 gives the held-out solve rate and cost accounting (iterations and injected tokens) of every method on the two hardest levels with Claude Sonnet, and Table 9 gives the open-weight head-to-head solve rate on two levels with each open-weight model, with per-replicate standard deviations. The ZERO-SHOT base rate on every level is in the main body (Table 1).

Does active experimentation and skill creation help agents solve complex tasks? We test whether distilling interaction into an evolving skill bank helps on levels where a single REACT trajectory fails. We compare against two kinds of baseline: methods that accumulate no reusable

Table 8: A distilled bank beats storing the experience itself, at a third of the context. We report Claude Sonnet 4.6 results on the hardest levels of INTERPHYRE. We build the frozen bank using HEXA training loop and evaluate on same 50 held-out seeds. *Success*: solve rate (%), mean ± one standard deviation over runs where more than one exists. *Avg iters*: mean turns per seed. *Tokens/ep.*: mean memory block injected into the actor's prompt per held-out episode (`cl100k_base`), excluding the task prompt and transcript. HEXA improves across first two metrics and uses substantially less tokens compared to other baselines.

Method	Success (%)	Avg iters	Tokens/ep.
`catapult` — 50 held-out seeds			
No persistent memory			
ZERO-SHOT	0.0	1.0	0
REACT	11.0±3.0	23.1±0.2	0
Reflexion	21.3±2.5	20.4±0.8	287
External memory			
MemRL	49.0±1.0	13.3±0.1	10,149
Episodic RAG	62.9±3.7	12.8±0.2	6,513
HEXA	**82.7±9.3**	**12.5±0.9**	**2,392**
`pass_the_parcel` — 50 held-out seeds			
No persistent memory			
ZERO-SHOT	0.0	1	0
REACT	20.0±5.7	18.7±1.3	0
Reflexion	32.0±6.6	20.1±0.8	263
External memory			
MemRL	46.0±7.0	16.8±1.3	10,023
Episodic RAG	60.0±6.9	15.0±1.4	5,735
HEXA	**96.0±2.8**	**5.8±0.8**	**2,631**

Table 9: A distilled skill bank is the only memory that helps a frozen open-weight actor on both INTERPHYRE levels, at both model sizes. Actor and evolver are the same model in every row. DTE is `down_to_earth` and TBP is `two_body_problem`. Every arm builds its memory on 50 training seeds, freezes it, and is scored on a disjoint pool of 50 held-out seeds it never saw; no distillation happens during evaluation. REACT is the identical harness with an empty bank, so the only variable is what the retriever places in the prompt. Reflexion keeps no persistent store, carrying only a verbal lesson across the attempts on a seed; Episodic RAG retrieves past rollouts verbatim; MemRL maintains (intent, experience, utility) triplets. *Success* is the percentage of held-out seeds solved, ± one standard deviation over $n = 3$ replicates. Bold marks the best value in each column; dashes mark cells not yet run.

	Qwen2.5-7B-Instruct		Qwen2.5-3B-Instruct	
Method	**DtE**	**TBP**	**DtE**	**TBP**
No persistent memory				
REACT	63.3 ±3.4	12.0 ±0.0	8.3 ±0.9	7.3 ±0.9
Reflexion (K=2)	76.0 ±0.0	27.9 ±0.1	2.0 ±0.0	8.0 ±0.0
External memory				
Episodic RAG	42.7 ±11.4	18.7 ±7.6	4.0 ±4.0	5.3 ±3.1
MemRL (numeric key)	36.0 ±6.0	23.3 ±4.2	3.3 ±3.1	6.0 ±2.0
HEXA	**82.7** ±5.2	**50.0** ±3.3	**23.3** ±0.9	**12.0** ±0.0

skills (ZERO-SHOT, REACT, Reflexion), and external-memory methods that retain past episodes rather than distilling them (Episodic RAG, MemRL). On the two hardest levels with Claude Sonnet, where the base model solves nothing, HEXA is the strongest method by a wide margin, reaching 82.7% on `catapult` and 96.0% on `pass_the_parcel` (Table 8, Figure 3). The memory-free baselines stay low, and the memory baselines recover only part of the gain. Figure 17 shows a representative `catapult` seed where the retrieved skills guide HEXA to a solution in a handful of interactions, while REACT exhausts its budget.

The same pattern holds for smaller open-weight actors (Table 9). With Qwen-2.5-7B and Qwen-2.5-3B, HEXA is the best method on both levels at both sizes, and on `two_body_problem` it lifts the

7B model from 12.0% under REACT to 50.0%. Storing experience alone is not enough: Episodic RAG and MemRL often fall below memoryless REACT. Their retrieved rollouts come from seeds with different geometry, so a small actor tends to copy placements that do not transfer, and the long verbatim transcripts crowd out its reasoning about the current scene. Reflexion comes closest only on `down_to_earth` with the 7B model, where it needs a second attempt on every seed. The gain therefore comes from distilling prior interactions into reusable strategies, not from extra retries or from placing past experience in the prompt. A run with GPT-OSS-120B on `catapult` shows the same effect, with HEXA lifting it from 0% under REACT to 54%.

A frontier actor. We repeat the build-then-freeze protocol with GPT-6-Astra on the two hardest levels (Figures 15 and 16): the bank is built over 17 rounds of 3 training seeds and then frozen for 50 held-out seeds. Scale alone does not solve these levels without interaction. On `catapult`, ZERO-SHOT Astra scores 0% at both default and high reasoning effort, and at high effort it runs out of its token budget mid-reasoning without committing an action on 13 seeds. On `pass_the_parcel` it reaches 6%. The stronger actor does make better use of interaction: REACT reaches 92% and 56%, far above Claude Sonnet's 11.0% and 20.0%. HEXA still improves on it on both axes. On `catapult` it solves every held-out seed (100%) in 6.3 turns per seed against 14.9 for REACT, injecting 1,113 tokens. On `pass_the_parcel` it reaches 76% against 56%, in 11.4 turns against 17.6, injecting 3,190 tokens. The bank builds faster with the stronger actor. Its cumulative training solve rate passes 80% by the second round on `catapult` and ends at 98%. On `pass_the_parcel` it plateaus near 67% after the first round and ends at 71%. So the value of the distilled bank is not specific to a mid-sized actor. As the actor gets stronger, the gain shifts from solve rate alone toward interaction cost: at saturation on `catapult`, HEXA still halves the turns per seed.

Can HEXA leverage learned skills for more efficient exploration and experimentation? Spending more compute on interaction does not by itself raise task success, so we record the average number of turns per seed as an efficiency proxy alongside solve rate, Table 8. Accuracy and cost move together: HEXA lowers the average from 23.1 ± 0.2 turns under REACT to 12.5 ± 0.9 on `catapult`, roughly 46% fewer, and from 18.7 to 5.8 on `pass_the_parcel`, so the extra successes come with less interaction per instance rather than more. It is also the cheapest memory to carry, at 2,392 injected tokens per episode on `catapult` against 6,513 for Episodic RAG and 10,149 for MemRL (Table 8); the bank is a fixed size, so its cost is constant in the number of episodes played, whereas storing rollouts scales with the rollout cap rather than with anything learned. Converting past experience into a reusable skill bank therefore amortizes the cost of future experimentation instead of re-paying it each episode.

What does it cost to construct and use the memory? We report (Table 10) the total token cost used by the memory baselines, Episodic RAG and MemRL, and HEXA for constructing their memory during the training phase, which includes actor calls and evolver calls (per round; for MemRL, its summariser per episode), and for evaluation on held-out seeds using the final memories or skill banks. HEXA constructs its memory at less than half the cost of the memory baselines (4.9M tokens on `catapult` against 10.5M and 12.3M), and its 17 evolver calls are only about 5% of that. Evaluation is also cheaper, so in total HEXA uses 2–3.5× fewer tokens than either memory baseline. REACT and Reflexion construct no memory, so the fair comparison with them is the cost of a solved seed. Even with its entire memory construction included, HEXA needs 0.21M tokens per solved seed on `catapult` against 0.32M for Reflexion and 1.03M for REACT (0.14M against 0.19M and 0.43M on `pass_the_parcel`), making it the cheapest method per solved seed on both levels.

H.2 INSTANTIATION AND ABLATIONS

Section C.5 compares the five instantiations of the bank (initialization × update rule); Table 11 gives their training solve rate and mean turns per seed on both levels.

Reward labels. HEXA can ingest additional experimental evidence such as trajectory rewards, which are a function of outcome reward and efficiency (Appendix C.2), to further guide skill evolution and retrieval. The skill reward label r_k is in turn a function of the trajectory rewards of the trajectories

Table 10: Total LLM token cost, memory construction included. Claude Sonnet 4.6 on the two hardest INTERPHYRE levels, in `cl100k_base` tokens. *Memory construction*: building the skill bank or memory store on the 51 training seeds. *Actor*: prompts and outputs of every actor turn. *Memory calls*: LLM calls outside the actor (HEXA's evolver, one per round; MemRL's summariser, one per episode). *Eval*: actor tokens over the 50 held-out episodes, including the injected memory (for Reflexion, both trials and its reflections). *Per solve*: total divided by the number of held-out seeds solved. REACT and Reflexion construct no memory.

	Memory construction		Eval			
Method	**Actor**	**Memory calls**	**Actor**	**Total**	**Success (%)**	**Per solve**
catapult						
REACT	0	0	5.66M	5.66M	11.0	1.03M
Reflexion	0	0	3.43M	3.43M	21.3	0.32M
MemRL	12.14M	0.20M (51)	10.11M	22.45M	49.0	0.92M
Episodic RAG	10.49M	0	7.29M	17.78M	62.9	0.57M
HEXA	4.61M	0.27M (17)	3.71M	8.58M	**82.7**	**0.21M**
pass_the_parcel						
REACT	0	0	4.26M	4.26M	20.0	0.43M
Reflexion	0	0	3.02M	3.02M	32.0	0.19M
MemRL	10.72M	0.17M (51)	12.49M	23.38M	46.0	1.02M
Episodic RAG	8.34M	0	7.74M	16.08M	60.0	0.54M
HEXA	4.52M	0.25M (17)	**1.79M**	6.56M	**96.0**	**0.14M**

Table 11: Initialisation and update rule. Claude Sonnet on the two hard INTERPHYRE levels under the round-by-round protocol (Appendix G.8),on 50 training seeds of `catapult` and `pass_the_parcel`. *Solve*: solve rate (%); *Iters.*: mean turns per seed, with a failure counted at the 25-turn cap. *Evolving* passes the evolver both the current bank and the new round's trajectories; *Iterative* rebuilds from the current rollouts alone. *Off2On* warm-starts from base-actor trajectories; *Online* starts cold; *Offline* keeps its warm-start bank frozen. Off2On Evolving, written simply as HEXA elsewhere, is best on both levels on both metrics.

	Catapult		Pass the Parcel	
Configuration	**Solve**	**Iters.**	**Solve**	**Iters.**
Off2On Evolving (HEXA)	**84.0**	**11.8**	**68.0**	**12.7**
Off2On Iterative	56.0	15.7	48.0	17.7
Online Evolving	44.0	18.2	58.0	16.4
Online Iterative	36.0	20.4	56.0	18.9
Offline Static	60.0	17.0	48.0	18.5

Table 12: Reward labels on the skill bank. Solve rate (%) and mean iterations per seed, both arms scored on 50 **training** seeds . The *no reward* variant treats every success and failure identically, so the evolver decides what to retain and prune from its own prior rather than from environment feedback; everything else is held fixed The accuracy shows mean $\pm$ one standard deviation over n=3 independently built banks per arm.

Actor / Level	**Method**	**Solve rate (%)**	**Iters**
Claude Sonnet	HEXA, no reward	51.0 $\pm$10.1	15.8
`catapult`	**HEXA, reward**	**78.0** $\pm$10.7	**13.3**
Qwen-2.5-7B	HEXA, no reward	64.0 $\pm$1.7	14.6
`down_to_earth`	**HEXA, reward**	**72.0** $\pm$5.2	**12.6**
Qwen-2.5-7B	HEXA, no reward	20.0 $\pm$2.7	22.2
`two_body_problem`	**HEXA, reward**	**42.0** $\pm$3.3	**17.7**

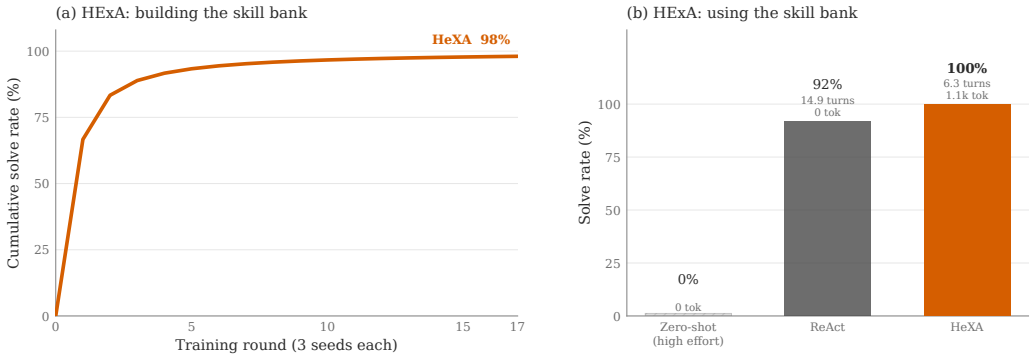


Figure 15: GPT-6-Astra on `catapult`. *(a)* Cumulative solve rate on the training seeds as the bank is built (17 rounds of 3 seeds). *(b)* Solve rate on the 50 held-out seeds with the bank frozen. Mean turns per seed and the injected memory per episode are printed above each bar. ZERO-SHOT is run at high reasoning effort; the default-effort run also scores 0%. HEXA solves every held-out seed in fewer than half the turns of REACT.

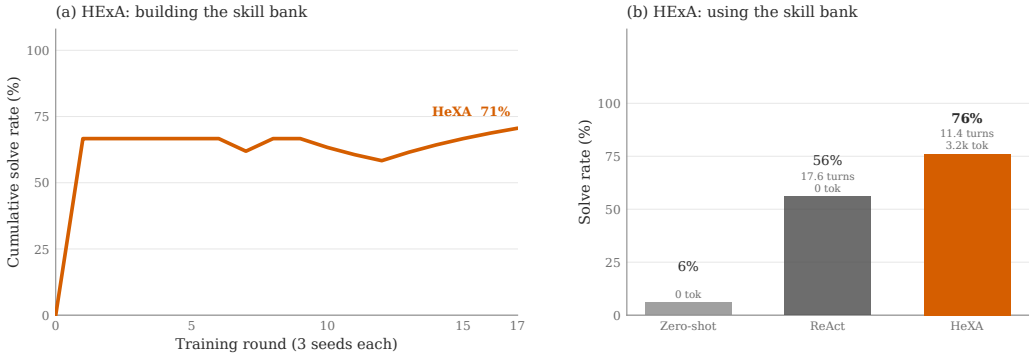


Figure 16: GPT-6-Astra on `pass_the_parcel`. Panels as in Figure 15; ZERO-SHOT is run at default reasoning effort. HEXA raises the held-out solve rate from 56% under REACT to 76% while cutting turns per seed from 17.6 to 11.4.

that invoked the skill,

$$r_k = \mathrm{clamp}\left(\frac{\bar{r}_{\mathrm{src}} + 1}{2},\, 0.1,\, 1.0\right), \tag{17}$$

where $\bar{r}_{\mathrm{src}}$ is the mean reward of those source trajectories. Users can define any trajectory and skill reward; HEXA as a framework is agnostic of these choices. We also run skill evolution without rewards, where every success and failure trajectory, and every skill distilled out of them, is treated equally, so the LLM evolver updates the bank from its prior knowledge rather than from environment feedback. Removing the labels costs accuracy on every level we test (Figure 6, Table 12): on `catapult` the reward-free variant drops Claude Sonnet from $78.0 \pm 10.7\%$ to $51.0 \pm 10.1\%$, and on Qwen-2.5-7B it reaches $64.0 \pm 1.7\%$ versus $72.0 \pm 5.2\%$ with rewards on `down_to_earth` and $20.0 \pm 2.7\%$ versus $42.0 \pm 3.3\%$ on `two_body_problem`. It also costs efficiency: the reward-free bank needs 15.8 iterations per seed on `catapult` against 13.3 with labels, 14.6 against 12.6 on `down_to_earth` and 22.2 against 17.7 on `two_body_problem`, so the labelled bank solves more seeds with fewer interactions. The reward labels thus provide environment-grounded feedback from the actor's own exploration that helps assign credit for effective and efficient skill acquisition, and better guides the evolver in composing, retaining or pruning skills. Our default setting for HEXA is therefore Off2On + Evolution with reward-labelled skills.

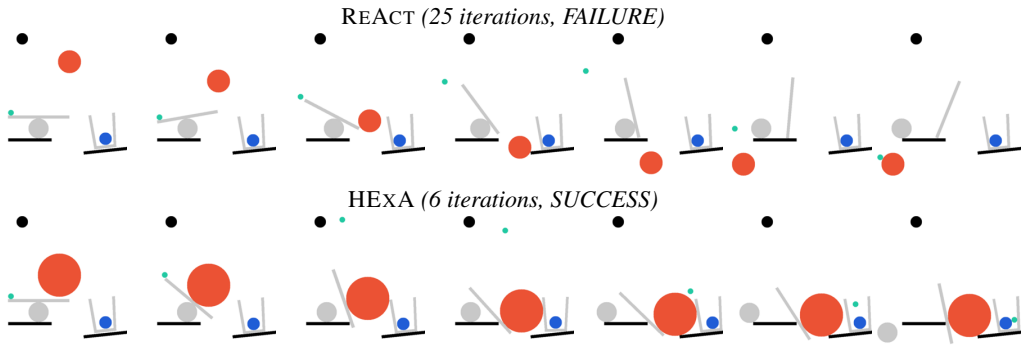


Figure 17: How the evolved bank guides the agent on a held-out `catapult` seed. Top: REACT, with no cross-episode memory, exhausts its 25-iteration budget on ineffective placements. Bottom: HEXA opens with the bank's learned default, reads the resulting failure through the bank's diagnosis rather than as a reason to keep tuning the same parameter, applies the corrective rule, and succeeds in six interaction iterations. Frames are sampled from the final simulations; full prompts, bank entries and trajectories are in Appendix K.4.

H.2.1 FROZEN-BANK CONTENT AND RANKING ABLATIONS

Setup. Because HEXA's skill bank is injected as a textual prefix, any improvement could in principle be attributed to prompt length, in-context formatting, or generic priming toward physics reasoning rather than to the specific knowledge the evolver distilled. We isolate this with frozen banks—no evolution during evaluation—under four content conditions and four ranking conditions (Figure 18; the numbers are those of Table 13 below).

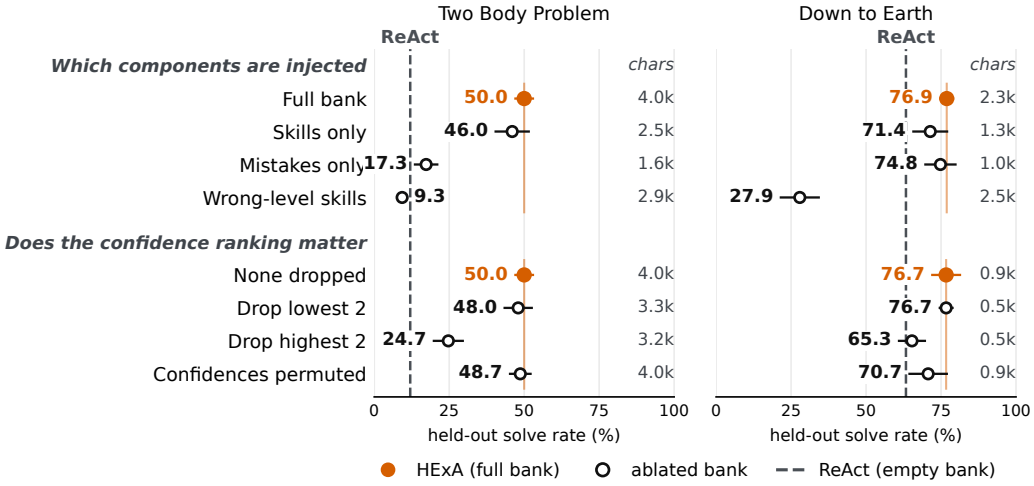


Figure 18: Frozen-bank content and ranking ablations, full results. Held-out solve rate (%) on 50 unseen seeds of `two_body_problem` and `down_to_earth` with Qwen-2.5-7B and a frozen skill bank; points are means over 3 independent runs and bars denote ± 1 s.d. The dashed line marks REACT with an empty bank ($\mathcal{K} = \varnothing$), the shaded orange region marks performance worse than injecting nothing, and the blue band marks ± 1 s.d. around the full-bank reference. The *injected context* column reports the character count of the retrieved prefix, so each condition's accuracy can be read against the amount of text it adds. *Top:* which components of the bank are injected—full bank, strategy skills only, mistake records only, or a length-matched bank evolved on a different level. *Bottom:* whether the evolver's reward-derived ranking carries information—all skills retained, the two lowest-rewards skills removed, the two highest removed, or rewards randomly permuted while retaining all content.

Strategy skills and mistake records are not interchangeable. Removing either component costs accuracy, but the two behave differently across levels. Strategy skills alone retain most of the gain on both levels (46.0% and 71.4%). Mistake records alone are nearly sufficient on `down_to_earth`

Table 13: Ablating the bank on INTERPHYRE, Qwen-2.5-7B throughout. *Success* is the percentage of 50 held-out seeds solved, ± one standard deviation over n=3 evaluations; *Chars* counts the injected block. Within a block every arm retrieves from a *single* frozen bank, so no difference can come from a different training run. **Top:** which components are injected — the wrong-level arm swaps in fluent skills distilled from the *other* level and keeps this level's own mistakes, at comparable injected length, so only content changes. **Bottom:** whether the reward ranking matters — the drop arms remove disjoint pairs of skills, on DTE at lengths within 0.9% of each other, and the permuted arm keeps every skill and only shuffles the rewards.The two blocks retrieve from different DTE banks, hence its two *None* rows; TBP uses one bank throughout. Bold marks the best value per block and level.

	Two Body Problem		Down to Earth	
Injected bank	**Chars**	**Succ.**	**Chars**	**Succ.**
Which components are injected				
Full bank	4004	**50.0** ±3.3	2335	**76.9** ±2.5
Skills only	2490	46.0 ±5.9	1345	71.4 ±6.0
Mistakes only	1564	17.3 ±4.1	1040	74.8 ±5.4
Wrong-level skills	2862	9.3 ±1.9	2476	27.9 ±6.7
Whether the reward ranking matters				
None dropped	4004	**50.0** ±3.3	892	**76.7** ±5.0
Drop *lowest* 2	3293	48.0 ±4.9	543	**76.7** ±2.5
Drop *highest* 2	3193	24.7 ±5.2	548	65.3 ±4.7
rewards permuted	4004	48.7 ±3.8	892	70.7 ±6.6
REACT (empty)	0	12.0 ±0.0	0	63.3 ±3.4

(74.8% of a 76.9% ceiling) yet collapse on `two_body_problem` (17.3%). We read this as a difference in failure structure: on the easier level most failures are recurring, locally correctable errors that a record of past mistakes prevents directly, whereas the harder level requires a positive multi-step strategy—bisection over a contact parameter, in the worked example of Appendix D.4.3—that descriptions of what went wrong do not supply. This supports maintaining both skill types rather than a single undifferentiated memory.

The evolver's rewards are informative. On `down_to_earth`, removing the two highest-reward skills costs 11.4 points (76.7% → 65.3%), whereas removing the two lowest leaves performance unchanged (76.7%) despite halving the injected context from 0.9k to 0.5k characters: the ranking identifies which entries carry the task, and the low-ranked tail is close to free to discard. Permuting rewards while retaining all content costs 6.0 points on `down_to_earth` (76.7% → 70.7%), indicating that retrieval order matters beyond mere inclusion, though on `two_body_problem` the same manipulation lies within run-to-run variance (50.0% → 48.7%). We therefore treat ranking quality as a secondary rather than primary driver of performance.

Relevance, not volume: the wrong-level control. We replace the bank with one evolved on a different INTERPHYRE level, matched in length and indistinguishable in surface form: fluent, well-structured physics guidance with the same schema of titles, principles and applicability conditions. On `two_body_problem` this reduces solve rate from 50.0% to 9.3%, below the 12.0% empty-bank REACT baseline, while injecting 2.9k characters against the full bank's 4.0k; the same ordering holds on `down_to_earth` (76.9% → 27.9%). Injecting plausible but misapplied procedural knowledge is therefore actively harmful relative to injecting nothing, which rules out length and priming explanations: a longer prompt that is wrong about the task is worse than no prompt at all. We attribute this to misdirected exploration—the actor commits early to a strategy whose applicability condition does not in fact hold, and spends its 25-turn budget refining the wrong hypothesis rather than forming a new one. This is also why cross-level transfer (Section C.6) requires the evolver's de-binding and re-grounding step rather than direct reuse of a source bank.

Summary. These ablations locate HExA's gain in task-relevant distilled content and its reward-derived prioritization, not in context length. Both skill types are load-bearing, their relative importance tracks the failure structure of the level, and the bank's value concentrates in a small number of high-reward strategies.

H.2.2 Is experimentation needed once the skill bank is built?

Motivation. HExA interacts with the environment in two places: while the bank is evolved and built, and again at evaluation, where the actor calls the tools with the frozen bank in context. One possibility is that, once built, the bank is sufficient on its own, so experimentation matters only for building it. The other is that the bank guides how the actor experiments, so the actor still needs experimentation at inference. To ablate this, we separate the two by giving the final skill bank to an agent for which inference time experimentation is disallowed.

Setup. We take the final skill bank of Claude Sonnet on catapult and pass_the_parcel levels and inject it exactly as in HExA's frozen evaluation. The agent receives the Zero-shot prompt: the task and a single description of the scene layout (object positions and sizes), with no experimentation tools. Only the skill bank is added. We evaluate it on the same 50 held-out seeds under three protocols:

- **Zero-Shot.** One call and one final placement.
- **Extended reasoning.** 25 turns with zero environment rewards or feedback. Turns 1–24 are a free scratchpad for forming hypotheses, working through the skills and eliminating options, and the agent commits one placement on turn 25. This matches HExA's 25-turn budget, but the turns are extended chain of thought rather than active experiments or interactions.
- **Binary feedback.** Up to 25 placements. After each one the agent is told only *success*, *failure* or *invalid* (overlap at t=0, which follows from the scene layout). There is no trajectory, no probe or trace tool, and no other simulator feedback. This tests whether a binary experiment decision is enough, or whether HExA's richer intervention tools are needed.

Table 14: The frozen bank without the tool interface. Claude Sonnet 4.6 on the 50 held-out seeds of the two hardest InterPhyre levels, with the final frozen HExA bank of one build per level injected exactly as in HExA's frozen evaluation. In the three middle rows the agent has no simulator tools; it is given the scene layout once and the prompt of the Zero-shot baseline. *Signal*: what the environment returns to the agent before its answer is scored. *Turns*: mean model calls per seed (placements for outcome-only feedback, including failed ones). The three middle rows are single runs over the 50 seeds. Zero-shot and ReAct are the rows of Table 8; HExA is the frozen evaluation of the same bank. (All numbers are on single run)

				Success (%) / Turns	
Arm	**Signal**	**Skills**	**Tools**	`catapult`	`pass_the_parcel`
Zero-shot	none	–	–	0.0 / 1	0.0 / 1
ReAct	full simulator	–	✓	11.0 / 23.1	20.0 / 18.7
Zero-Shot (Skill Bank)	none	✓	–	16.0 / 1	22.0 / 1
Extended reasoning (25 turns)	none	✓	–	42.0 / 25	48.0 / 25
Binary feedback (25 turns)	success / failure	✓	–	90.0 / 8.0	90.0 / 5.1
HExA (frozen, same bank)	full simulator	✓	✓	86.0 / 12.7	96.0 / 5.8

Distilled skills improve the actor's physical reasoning. Without any environment feedback the skill bank lifts the actor well above its baselines (Table 14). Zero-Shot reaches 16.0% on `catapult` and 22.0% on `pass_the_parcel`, against 0% for Zero-shot. With the 25-turn reasoning budget this rises to 42.0% and 48.0%, above ReAct with the full simulator (11.0% and 20.0%). Because every seed is held out and has a new layout, the gain cannot come from recalling past placements. The skills supply transferable physical understanding of the level: how the mechanism behaves (a drop right of the catapult pivot launches the green ball rightward), which parameters matter, and which errors to avoid. The actor applies this to a new scene and reasons its way to a correct placement on roughly half the seeds without touching the simulator. The gain also grows with the reasoning budget (16% $\rightarrow$ 42% and 22% $\rightarrow$ 48%), which is what one expects if the skills act as a scaffold for reasoning rather than as a lookup.

Experimentation at inference remains necessary. Reasoning alone stays 44 and 48 points below HExA with the same skill bank (86% and 96%). The only difference between extended reasoning and HExA is experimentation and environment feedback. Adding even binary success or failure feedback of the experiments after each placement, the actor reaches 90.0% on both levels, in 8.0 and 5.1 placements per seed. With extended reasoning, the actor typically settles on its answer within the first half of the budget and restates it for the remaining turns, because no new information arrives.

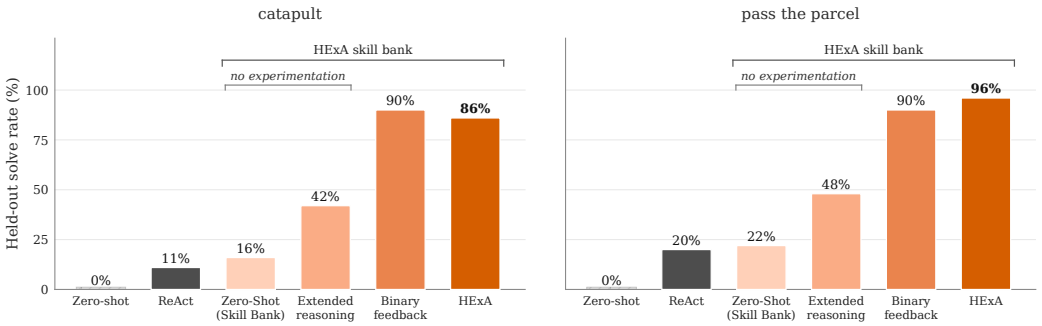


Figure 19: The HExA skill bank with and without experimentation tools. Claude Sonnet 4.6, solve rate on the 50 held-out seeds of each level. Greys use no skills: ZERO-SHOT (no interaction) and REACT. Oranges all use the same frozen HExA skill bank, with lighter shades receiving less environment feedback at evaluation. *Zero-Shot (Skill Bank)*: one call, one placement. *Extended reasoning*: 24 scratchpad turns without feedback, then one placement. *Binary feedback*: up to 25 placements, each answered only with success or failure. HExA: the full simulator. Numbers as in Table 14.

This ablation thus clearly shows that Experimentation matters in both phases. While the bank is built, it produces transferable skills that sharpen the actor's reasoning about the level: these alone take the actor from 0% to 42–48%. At inference, it lets the actor carry out the search the skills describe, and even outcome-only feedback is enough for this (about 90%). Neither ingredient suffices alone. Interaction without skills (REACT) reaches 11–20%, and skills without interaction reach 42–48%. Together they reach 86–96%.

H.3 OTHER BENCHMARKS AND TASKS

We repeat the same frozen-bank protocol on three external benchmark families: interactive games, reasoning and planning, and open-ended scientific discovery, against the same non-experimental and external-memory baselines. Figure 4b summarizes each family as one score per actor, averaged over that family's environments; Table 15 reports the per-task values it averages, and the three subsections below give token budgets, per-episode breakdowns and exact paired tests for each family.

Across all four actor groups, HExA is the best-performing method in 9 of 10 task-actor settings; on Maze with gpt-oss-120b, MemRL is marginally higher (0.666 vs. 0.649) at about 8× the injected context. EPISODIC RAG falls below ZERO-SHOT on the interactive games with `gpt-oss-120b`, while MEMRL falls below it on DISCOVERPHYSICS. Thus, each episodic-memory method imposes a cost in at least one family, whereas the distilled bank improves every reported actor group.

H.3.1 INTERACTIVE GAMES

Table 16 is the full head-to-head on Mastermind and Maze: four task/actor blocks, each arm playing an identical held-out pool, with first-episode, last-episode and `any`-episode success, the injected token budget, and a two-sided paired sign test on episode-3 outcomes against that block's ZERO-SHOT arm.

Do distilled skills help when each instance is replayed? For the interactive games, each instance is played for three episodes with the hidden task held fixed. Figure 4b reports `any`, the fraction of instances solved in at least one of the three episodes; Table 16 additionally reports *ep1* and *ep3*, success on the first and final episodes, which measure how the agent's behavior evolves across repeated attempts on the same instance.

The bank improves both actors and is also the cheapest persistent memory. On Mastermind with GPT-OSS-120B, HExA injects 971 tokens per episode compared with 1,957 for Episodic RAG and 3,416 for MEMRL, while reaching 88.3% `any`-episode success versus 68.3% for no memory. With Qwen3-8B the token budgets differ: HExA uses 359 tokens versus 1,965 for Episodic RAG and 3,192 for MEMRL, and achieves 64.0% `any`-episode success against 36.7% baseline. On Maze, token costs again vary by model: GPT-OSS-120B requires 945, 4,699, and 7,572 tokens respectively, with

Table 15: The bank lifts a frozen actor on all three external families. One row per task/actor cell, on the metric each block states. Every bank is frozen before evaluation and scored on instances it never saw, under each family's own split; values are the mean over n=3 replicates where more than one run exists. On the games and DISCOVERPHYSICS all arms are paired on identical instances. Bold marks the best value per row. Significance tests, token budgets and per-episode breakdowns are in Appendices H.3.1–H.3.3.

Task	Actor	Zero-Shot	Episodic RAG	MemRL	HEXA
Interactive games— success rate on the metric each block reports					
Mastermind (any)	gpt-oss-120b	.683	.650	.690	**.883**
Mastermind (any)	Qwen3-8B	.367	.492	.392	**.640**
Maze (any)	gpt-oss-120b	.607	.601	**.666**	.649
Maze (any)	Qwen3-8B	.479	.483	.513	**.564**
Reasoning & planning — accuracy on 100 held-out items					
MathGAP	gpt-oss-120b	0.510	0.560	0.600	**0.863**
ZebraLogic	gpt-oss-120b	0.560	0.720	0.650	**0.807**
Tower of Hanoi	gpt-oss-120b	0.263	0.377	0.273	**0.520**
Matchstick	gpt-oss-120b	0.737	0.767	0.703	**0.787**
DISCOVERPHYSICS — explanation score over 15 paired seeds					
oscillator	claude-sonnet-5	0.493	0.773	0.527	**0.860**
circle	claude-sonnet-5	0.587	0.553	0.540	**0.887**

HEXA at 65% any-episode success; Qwen3-8B requires 429, 5,457, and 7,027 tokens, with HEXA at 56.4% any-episode success against 47.9% baseline. Thus, the episodic baselines inject between 3.5× and 16× as much context as the skill bank across all settings, yet consistently underperform.

Storing episodes verbatim does not consistently transfer across instances: a retrieved transcript from a previous instance can displace the actor's own cross-episode observations rather than supplement them with a reusable abstraction. REFLEXION, which keeps only verbal lessons with minimal injected context, shows the lowest persistent memory cost but also the lowest any-episode success across most settings. In contrast, HEXA compresses prior experience into a much smaller set of strategies that can be reused across instances while leaving more context available for the current interaction.

H.3.2 REASONING AND PLANNING

Table 17 gives the per-task head-to-head on the four reasoning and planning tasks. Every arm is our own run on our provider at the same rollout budget as HEXA, so all rows are paired on identical items.

Do distilled skills help on reasoning and planning tasks? Every method receives the same rollout budget, so the comparison differs only in what information is carried between items. HEXA improves the frozen actor on all four tasks, raising mean accuracy from 0.518 with ZERO-SHOT to 0.744, and is the only memory method that improves every task rather than trading gains on one task against losses on another. Episodic RAG and MEMRL reach mean accuracies of 0.606 and 0.556, respectively. Storing previous rollouts therefore recovers part of the benefit, but not most of it, despite the equal rollout budget. The additional gain from HEXA comes from making distilled strategies immediately available to later items through context rather than carrying the original trajectories themselves.

H.3.3 DISCOVERPHYSICS

Table 18 gives explanation score, benchmark reward and pass-rate at the 0.90 gate for all three worlds, comparing a frozen skill bank against EPISODIC RAG, MEMRL, and no memory, all paired on the same 15 seeds.

Do distilled skills help when the task is open-ended scientific discovery? On DISCOVERPHYSICS, the skill bank contains experimentation strategies rather than task solutions. HEXA is the only method that improves explanation on all three worlds. On oscillator, both EPISODIC RAG (0.773) and HEXA (0.860) improve over the baseline (0.493), but HEXA reaches 11/15 pass-rate at the strict 0.90 explanation gate versus Episodic RAG's 6/15 and MemRL's 7/15. On

Table 16: HExA gives the largest gains in three of four settings at the lowest memory cost among memory baselines. Four task/actor settings on the two games, each a separate block. Each instance is played for three episodes: *ep1* and *ep3* are first- and last-episode success rates and `any` is success in at least one of the three. Every value is the mean ± one standard deviation over three independently built memories per arm. *Tokens* is the memory block injected into the actor's prompt each episode. Episodic RAG retrieves past rollouts verbatim; MemRL maintains (intent, experience, utility) triplets; Reflexion keeps no persistent store, carrying only a verbal lesson across episodes, and is the one arm that *removes* context by truncating the cross-episode transcript. A dash marks an arm that was not run at this block's budget. Bold marks HExA and the best value per block.

	Method	ep1	ep3	any	Tokens
Mastermind `gpt-oss-120b` n=40	Zero-Shot	.100 ± .035	.650 ± .020	.683 ± .031	0
	HExA	.058 ± .024	**.875 ± .020**	**.883 ± .012**	**971**
	Episodic RAG	.083 ± .031	.633 ± .085	.650 ± .094	1,957
	MemRL	.070 ± .021	.685 ± .250	.690 ± .244	3,416
	Reflexion	**.117 ± .047**	.133 ± .051	.283 ± .031	483
Mastermind `Qwen3-8B` n=40	Zero-Shot	.092 ± .012	.283 ± .083	.367 ± .072	0
	HExA	.079 ± .047	**.471 ± .080**	**.640 ± .082**	**359**
	Episodic RAG	.092 ± .012	.458 ± .043	.492 ± .031	1,965
	MemRL	.017 ± .024	.300 ± .054	.392 ± .043	3,192
	Reflexion	**.158 ± .012**	.325 ± .041	.433 ± .024	269
Maze `gpt-oss-120b` n=243	Zero-Shot	.244 ± .005	.610 ± .014	.607 ± .016	0
	HExA	.198 ± .040	.604 ± .033	.649 ± .045	**945**
	Episodic RAG	**.410 ± .037**	.549 ± .024	.601 ± .020	4,699
	MemRL	.341 ± .022	**.632 ± .026**	**.667 ± .024**	7,572
	Reflexion	.229 ± .028	.308 ± .018	.519 ± .043	358
Maze `Qwen3-8B` n=243	Zero-Shot	.243 ± .038	.472 ± .025	.479 ± .019	0
	HExA	.278 ± .010	**.542 ± .011**	**.564 ± .010**	**429**
	Episodic RAG	**.333 ± .037**	.379 ± .060	.483 ± .067	5,457
	MemRL	.296 ± .033	.425 ± .020	.513 ± .032	7,027
	Reflexion	.263 ± .018	.299 ± .032	.465 ± .022	194

Table 17: HExA improves a frozen model on all four reasoning and planning tasks. Actor and evolver are both `gpt-oss-120b`. All cells are accuracy, the fraction of held-out items answered correctly; where more than one run exists the value is the mean over three runs with ± one standard deviation. Every arm is our own run on our provider at the same rollout budget, so all rows are paired on identical items and compared by exact McNemar on per-item outcomes. Bold marks the best value in each column.

Method	MathGAP	ZebraLogic	Hanoi	Matchstick
Zero-Shot	0.510 ±.006	0.560 ±.026	0.263 ±.007	0.737 ±.023
Episodic RAG	0.560±.010	0.720±.030	0.377±.037	0.767±.009
MemRL	0.600±.015	0.650±.002	0.273±.027	0.703±.023
HExA	**0.863** ±.009	**0.807** ±.018	**0.520** ±.010	**0.787** ±.050
McNemar p, HExA *vs. no memory*	*4.7e-23*	*2.8e-14*	*7.4e-12*	*0.0081*

`circle`, neither episodic baseline improves explanation (Episodic RAG 0.553, MemRL 0.540), leaving HExA (0.887) as the sole improvement on the metric. On `hubble`, cross-world transfer, HExA achieves 0.840 explanation while the episodic baselines reach 0.740 (Episodic RAG) and 0.753 (MemRL).

The improvement also appears under the benchmark's strict success criterion. On `oscillator`, pass rate at the 0.90 gate rises from 0/15 to 11/15 with HExA, whereas Episodic RAG reaches 6/15 and MemRL 7/15. On `circle`, the benchmark reward does not improve for any method, yet the explanation-score gain is the most pronounced across all three worlds. This is not inconsistent: the evaluator re-optimizes the submitted parameters against the agent's own trajectories, so an incorrect functional form can still fit the observed data well. The explanation metric therefore exposes improvements in discovering the underlying law that the fit-based reward may not fully reflect, and distilled strategies succeed where verbatim-retrieval and learned-summarisation baselines do not.

Table 18: DISCOVERPHYSICS: distilled skills vs. episodic memory baselines. The agent designs experiments in a world governed by an unknown field equation, observes noisy particle trajectories, and submits the governing law as executable Python with a prose explanation. Actor and evolver are both `claude-sonnet-5`; every number is scored by `claude-opus-4-6`, a judge independent of the actor and the one the benchmark itself uses. All arms are paired on the same 15 held-out seeds per world, evaluated under a protocol registered before evaluation. *Expl.* is the judge's explanation score and *Reward* the benchmark's fit score, both higher-is-better; *pass@1*$_{.90}$ counts episodes with explanation ≥ 0.90 and normalised MSE < 0.1. EPISODIC RAG retrieves past rollouts verbatim; MEMRL maintains (intent, experience, utility) triplets with LLM summarisation; HEXA uses a frozen distilled bank. `oscillator` and `circle` are within-world accumulation where the bank encodes that world's own physics; `hubble` is cross-world transfer, grown on nine other worlds with `hubble` unseen. Bold marks entries that improve significantly over baseline ($p < 0.05$).

World	Design	Method	Expl.	Reward	pass@1$_{.75}$	pass@1$_{.90}$
`oscillator`	within-world	Zero-Shot	0.493	−0.060	1/15	0/15
		EPISODIC RAG	**0.773**	**+0.768**	**9/15**	**6/15**
		MEMRL	0.527	**+0.668**	**7/15**	**7/15**
		HEXA	**0.860**	**+0.859**	**11/15**	**11/15**
`circle`	within-world	Zero-Shot	0.587	+0.519	0/15	0/15
		EPISODIC RAG	0.553	+0.215	0/15	0/15
		MEMRL	0.540	+0.167	0/15	0/15
		HEXA	**0.887**	+0.767	**8/15**	**6/15**
`hubble`	cross-world	Zero-Shot	0.680	+0.618	5/15	1/15
		EPISODIC RAG	0.740	+0.466	9/15	4/15
		MEMRL	0.753	+0.675	7/15	2/15
		HEXA	**0.840**	**+0.852**	**12/15**	**10/15**

Exact paired tests vs. baseline (sign-flip permutation for continuous metrics, McNemar for pass-rate): `oscillator` HEXA expl. +0.367 ($p = .0001$), pass@1$_{.90}$ $p = .0010$; `circle` HEXA expl. +0.300 ($p = .0001$), pass@1$_{.90}$ $p = .0312$; `hubble` HEXA expl. +0.160 ($p = .0112$), pass@1$_{.90}$ $p = .0117$. On circle and hubble, EPISODIC RAG and MEMRL do not improve explanation significantly.

Table 19: Skills transfer with no target-task data, within an environment, across tasks and across worlds. In every row the target bank is synthesised by $\mathcal{M}_\theta$ from source banks plus a short textual statement of the target task, injected with no further evolution. The metric is stated per block and the baseline is that block's matched no-memory arm on the same instances. The italic row is the control: the same Wordle bank injected *verbatim*, without re-grounding, on the same instances as the row above it. Bold marks the transfer arms. Source-bank sizes, synthesised bank sizes, per-episode breakdowns and replicate-level tests are in Appendix H.4.

Target	Source(s)	Actor	Zero-Shot	HEXA Cross-Transfer	Δ
Across levels within INTERPHYRE — solve rate (%) on 50 held-out seeds					
`catapult`	DtE + TBP + PtP	Sonnet	11.0	**44.0**	+33.0
`falling_into_place`	DtE	Qwen-7B	20.0	**32.0**	+12.0
`two_body_problem`	DtE	Qwen-7B	12.0	**34.0**	+22.0
Across tasks in the interactive games — `any`-episode success rate					
Mastermind	*(within-task)*	`oss-120b`	.683	.883	+.200
Mastermind	Wordle	`oss-120b`	.683	**.933**	+.250
Mastermind	Minesweeper	`oss-120b`	.683	**.833**	+.150
Mastermind	Minesw. + Wordle	`oss-120b`	.683	**.808**	+.125
Maze	*(within-task)*	`oss-120b`	.607	.649	+.027
Maze	Minesweeper	`oss-120b`	.607	**.610**	+.003
Maze	Wordle	`oss-120b`	.607	**.671**	+.064
Maze	Minesw. + Wordle	`oss-120b`	.607	**.708**	+.101
Across worlds in DISCOVERPHYSICS — explanation score over 15 paired seeds					
`hubble`	9 other worlds	`claude-sonnet-5`	0.680	**0.840**	+.160

H.4 GENERALIZATION AND TRANSFER OF HIERARCHICAL SKILLS

Table 19 summarises every transfer result shown in Figure 7B. Table 20 reports the source-bank and synthesised-bank sizes for cross-level transfer within INTERPHYRE.

H.4.1 HOW SOURCE SKILLS COMPOSE INTO A HIERARCHICAL CATAPULT BANK

We analyze the provenance of the zero-shot `catapult` bank to make the hierarchical composition in cross-level transfer concrete. The transfer evolver receives banks from Down to Earth, Two Body

Table 20: Cross-level meta-skill transfer results. For each *target* level we report (i) the source skill banks consumed by $\mathcal{M}_\theta$, with their per-bank skill counts; (ii) the size of the synthesised target bank (n_k skills / n_μ mistakes); (iii) the agent that solves the target with *only* the synthesised bank injected; (iv) the resulting target success rate; and (v) the matched REACT baseline. No target-level trajectories are seen at any stage of synthesis. †Mean over REACT runs, hence no seed count.

Target	Sources (skills)	Synth. (k/μ)	Solver	Acc. (%)	Baseline (%)	Δ
Catapult	DTE (8) + TBP (8) + PTP (10)	9 / 4	Sonnet	**44.0** (22/50)	11.0†	+33.0
Falling Into Place	DTE (8)	4 / 2	Qwen-7B	**32.0** (16/50)	20.0 (10/50)	+12.0
Two Body Problem	DTE (8)	3 / 1	Qwen-7B	**34.0** (17/50)	12.0	+22.0

Problem, and Pass the Parcel, together with a textual description of `catapult`, but no target-level trajectory. The three banks contain 26 source skills in total (8, 8, and 10, respectively), listed below before the synthesized target entries. In the identifiers, `sk` denotes a distilled skill, `fsk` a failure-derived skill, `ev` an evolved skill, and `err` a recorded mistake.

Source-level skills. Tables 21–23 enumerate the full skill vocabulary available to the transfer evolver from each source level.

Table 21: The eight source skills in the `down_to_earth` bank.

Entry ID	Source-level skill
`dow_sk_000`	Gap Analysis Determines Push Direction
`dow_sk_001`	Place Red on the Opposite Side of the Target Gap
`dow_sk_002`	Horizontal Separation Must Be Within Collision Range
`dow_sk_003`	Use Radius 0.6 for Larger Collision Cross-Section
`dow_sk_004`	Jump to Minimum Separation, Not Tiny Increments
`dow_sk_005`	Place Red Slightly Below Green's Height
`dow_fsk_000`	Lateral Offset Drop Produces Horizontal Push
`dow_fsk_001`	Gap Exit Side Dictates Red Ball Starting Side

Note. The text in the second column is the title stored in the corresponding source-bank skill entry.

Table 22: The eight source skills in the `two_body_problem` bank.

Entry ID	Source-level skill
`two_sk_000`	Place Red on Opposite Side from Blue
`two_sk_001`	Small Horizontal Offset Creates Glancing Push
`two_sk_002`	Use Moderate Radius to Avoid Overlap and Wall Collisions
`two_sk_003`	Place Red Just Above Green, Minimize Drop Height
`two_sk_004`	Jump Geometrically Past Overlap, Not Incrementally
`two_sk_005`	Confirm Push Direction Before Simulating
`two_fsk_000`	Clearance Error as Direct Placement Calculator
`two_fsk_001`	Collision Angle Determines Push Direction

Note. The text in the second column is the title stored in the corresponding source-bank skill entry.

Table 23: The ten source skills in the `pass_the_parcel` bank.

Entry ID	Source-level skill
`pas_sk_000`	Ramp Launch: Upper-Mid Zone Placement
`pas_ev_12_001`	$y = 4.2$ as Preferred Default Launch Height
`pas_ev_10_002`	Safe Radius Range: $r = 0.45$–0.52
`pas_sk_002`	`get_ramp_center` Suggested Placement Trust
`pas_ev_10_004`	Ramp Overlap Avoidance: Rightward x Boundary
`pas_sk_003`	Overshoot Detection: Binary Threshold Behavior
`pas_ev_8_005`	Basket-Fall Coupling: Velocity-Match Signals Active Re-Trapping
`pas_ev_6_006`	X-Bisection at Fixed Radius: Opposite-Direction Overshoots
`pas_ev_7_008`	Ceiling-Bounded Y Placement for High Ramp Entries
`pas_ev_13_009`	Low Ramp Position (`ramp_y` < 1.0) Creates Unsolvable Energy-Deficit Geometry

Note. The text in the second column is the title stored in the corresponding source-bank skill entry.

The synthesis audit may also use a source-bank mistake as negative evidence. Table 24 defines the six source mistakes that occur as parents of the synthesized Catapult entries; source mistakes that were not selected by synthesis are omitted for clarity.

Table 24: Source mistakes selected as parents during Catapult synthesis.

Entry ID	Source-level mistake
dow_err_001	Escapes overlap through many tiny fixed steps instead of using the reported clearance.
dow_err_002	Increases radius without accounting for the resulting larger exclusion zone.
two_err_000	Micro-steps away from overlap despite being given the exact required displacement.
two_err_004	Places the red ball on the side that pushes the green ball away from the goal.
pas_err_000	Bisects one parameter inside a non-monotonic, direction-flipping regime.
pas_err_003	Repeats placements that overlap the ramp or ceiling and therefore never simulate.

Note. Source mistake entries do not have a title field; the text in the second column is a concise description.

Here, below a *parent entry* is an reference from one synthesized target entry to one cited source-bank entry. A source entry is counted once for every target entry that uses it. Thus the nine synthesized skills have 22 parent links (2.44 per skill) but 21 unique source entries, because pas_sk_000 contributes to both cat_xl_003 and cat_xl_007. The four synthesized mistakes have nine parent links (2.25 per mistake), all to distinct source entries within the mistake analysis.

Table 25: The nine synthesized hierarchical skills and their parent links in the zero-shot Catapult bank.

Entry ID	Synthesized Catapult skill	Parent source entries
cat_xl_000	Right-Side Arm Strike Generates Launch Torque	dow_sk_001, two_sk_000, two_fsk_001
cat_xl_001	Drop Height Governs Impact Velocity on the Arm	dow_sk_005, two_sk_003, pas_ev_12_001
cat_xl_002	Red Ball Mass Scales Arm Torque as r^3	dow_sk_003, two_sk_002, pas_ev_10_002
cat_xl_003	Moment Arm Length Multiplies Rotational Effect	dow_fsk_000, pas_sk_000
cat_xl_004	Ceiling Blocker Constrains Green Ball Arc	pas_ev_7_008, pas_err_003
cat_xl_005	Query Level State Before First Placement	two_sk_005, pas_sk_002
cat_xl_006	Geometric Clearance: One-Step Overlap Resolution	dow_sk_004, two_sk_004, two_fsk_000
cat_xl_007	Sustained Contact Requires Green Ball Entry into Basket	pas_ev_8_005, pas_sk_000
cat_xl_008	Switch Parameter Axis at Opposite Failure Modes	pas_sk_003, pas_ev_6_006

Note. The text in the second column is the title stored in the synthesized skill entry; the third column lists its audited parent entries.

Table 26: The four synthesized hierarchical mistake rules and their parent links in the zero-shot Catapult bank.

Entry ID	Synthesized Catapult mistake	Parent source entries
cat_xl_err_000	Striking the left side of the pivot rotates the arm in the wrong direction.	two_err_004, dow_sk_001
cat_xl_err_001	Resolving overlap through tiny positional steps exhausts the interaction budget.	dow_err_001, two_err_000, two_fsk_000
cat_xl_err_002	Increasing radius without recomputing clearance invalidates an otherwise useful placement.	dow_err_002, two_sk_002
cat_xl_err_003	Continuing to tune one parameter after opposite failures ignores the need to switch search axes.	pas_err_000, pas_sk_003

Note. Synthesized mistake entries do not have a title field; the second column is a concise description and the third column lists the audited parent entries.

Three examples illustrate the abstraction step:

1. **Direction becomes torque.**

   dow_sk_001, two_sk_000, and two_fsk_001 establish that the side and angle of contact determine the direction of the transferred impulse. The evolver re-grounds this shared principle as cat_xl_000: strike the arm to the right of its pivot so that the opposite end launches the green ball.

2. **Local overlap feedback becomes a general correction rule.**

   dow_sk_004, two_sk_004, and two_fsk_000 independently favor an analytic jump beyond the exclusion boundary over incremental motion. Their synthesis yields cat_xl_006, which applies the same one-step geometric correction to overlaps with the arm, pivot, ledge, or ceiling blocker.

3. **A source search failure becomes a target-level search policy.**

`pas_sk_003` and `pas_ev_6_006` describe opposite outcomes across nearby settings in a non-linear regime. The evolver abstracts this into `cat_xl_008`: when one Catapult parameter produces opposite failure modes, stop fine-tuning it and change an independent control axis.

This shows the hierarchy at the level of individual bank entries: simple, task-local rules and failure observations are selected across source levels and composed into target-grounded mechanisms.

H.5 Non-Parametric HExA against parametric post-training with GRPO

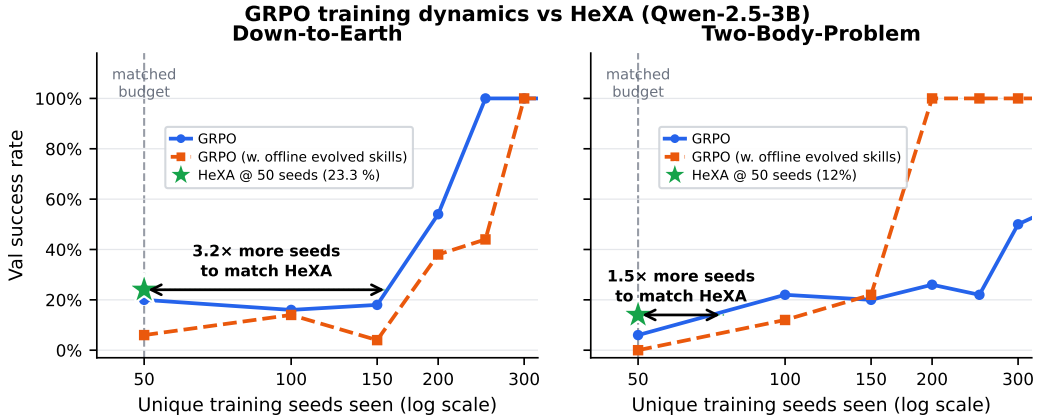


Figure 20: In-context skill evolution learns faster than GRPO at the same interaction budget. We compare HExA with GRPO fine-tuning on Qwen-2.5-3B using 50 training seeds. The green star marks HExA's success rate after the same 50-seed budget (Down to Earth: 23.3%, Two Body Problem: 12%), while the dashed line marks the matched-budget GRPO checkpoint. In this low-data regime, HExA outperforms both GRPO variants, suggesting that reusable in-context skills provide faster early adaptation; with many more updates, GRPO can eventually close the gap through direct reward optimization.

To situate HExA in the broader RL landscape, we train the same Qwen-2.5-3B model on the same two levels with Group Relative Policy Optimization (GRPO) (Shao et al., 2024), using batch size 1, n=4 rollouts per step, and a binary $\{0, 1\}$ reward from the environment. With 50 training seeds and batch size 1, one epoch of GRPO corresponds to 50 gradient updates — the same number of unique puzzle instances seen by HExA in a single pass. Figure 20 shows validation accuracy across 6 epochs for two variants: GRPO, and GRPO with a previously evolved skill bank baked into the prompt at data-generation time.

At the matched sample budget (50 unique seeds, vertical dashed line), GRPO without skills reaches 20% on `down_to_earth` and 6% on `two_body_problem`, against HExA's 23.3% and 12.0%. This gap is consistent with the established finding that in-context methods are more sample-efficient in new domains: skills distilled from early trajectories are immediately available to subsequent episodes via context injection, whereas GRPO must accumulate gradient signal over many rollouts before the policy weights encode the same knowledge.

Given sufficient training, however, GRPO's exact environment reward eventually dominates: `down_to_earth` reaches 100% by epoch 5 (GRPO) and epoch 6 (GRPO with skills), while `two_body_problem` peaks at 56% and plateaus around 40–50% for GRPO and converges to 100% by epoch 4 with skills. The skilled variant additionally exhibits a pronounced epoch-1 dip (6% on `down_to_earth`, 0% on `two_body_problem`) before recovering, consistent with the model needing several epochs to re-adapt its generation format to the longer skill-augmented prompt ($\sim$2200 vs. $\sim$1400 tokens) before it can leverage the injected skills. In-context skill evolution is therefore the preferred approach when the sample and interaction budget is small, while gradient-based RL with an exact reward remains a strong option to bootstrap further after an in-context warm start.

The gap reflects the core advantage of in-context skill evolution: strategies distilled from early trajectories are immediately available to later episodes through context, whereas gradient-based methods must accumulate signal over many rollouts before the weights encode equivalent knowledge—so in harder domains where early successes are rare, HExA can still enable experimentation guided learning from failure and partial-success trajectories. With many more updates, GRPO can eventually close the gap through direct optimization on the environment reward. But combining learning from

experimentation in-context to bootstrap initial progress on complex domains with later stages of gradient based RL post-training can be a robust learning method.

I INTERPHYRE PROMPTS

This appendix gives the prompts used on INTERPHYRE. All methods share the per-level REACT actor system prompt (Appendix I.1). HEXA adds its skill bank to that prompt, built by the evolver prompts of Appendix I.2; Reflexion, Episodic RAG and MemRL append their own memory block to it (Appendices I.3–I.5), and Reflexion and MemRL additionally use the prompts that build that memory. Long passages are abbreviated with `...`; the mechanics of each baseline are described in Appendix G.

I.1 REACT SYSTEM PROMPTS

The prompt contains the factual scene description, the success condition, the placement constraints, the full tool list (*including* the level-specific analysis tool), and the standard REACT format instructions; we give the agent no privileged information such as strategy tips, directional or geometry hints, radius guidance or physics rules. We show the full prompts for Down to Earth, Two Body Problem, Pass the Parcel and Catapult; for the other four levels, the parts shared with these (the generic simulator tools, the `finish` tool and the REACT format instructions) and long tool descriptions are abbreviated with `...`.

I.1.1 DOWN TO EARTH

```
You are an expert physics reasoning agent solving a 2D physics puzzle.
You have access to a physics simulator and can test your ideas before
submitting a final answer.

**Puzzle: Down to Earth**

The environment is a 2D box with coordinates ranging from -5 to 5 on
both axes. Gravity pulls objects downward.

**Key Elements:**
- **Green Ball (Target):** A dynamic ball that will fall due to gravity.
- **Black High Platform:** A static horizontal platform below the green
ball. Without intervention, the green ball lands on this platform and
stays there.
- **Purple Ground:** The floor at the very bottom of the box (y ~ -5).

**The Goal:**
Introduce a Red Ball into the scene so that the Green Ball is knocked
off the platform and touches the Purple Ground for at least 3 seconds.

**Placement Constraints:**
- The red ball must be completely inside the box: -5 + radius <= x <= 5
- radius, -5 + radius <= y <= 5 - radius.
- The red ball must NOT overlap with the green ball at t=0: distance
between centers > sum of radii.
- The red ball must NOT overlap with the black platform at t=0.

You have access to the following tools to interact with the physics
simulation:


1. get_level_state
   Description: Get the current level layout including all object
positions, sizes, and properties.
   Arguments: None
   Usage: Action: get_level_state

2. simulate_action
   Description: Place a red ball at (x, y) with the given radius and run
 the full physics simulation to completion. Returns whether the goal was
 achieved, final positions of all objects, and total simulation steps.
```

```
If the placement is invalid (out of bounds or overlaps), returns a
detailed error with how far to move the ball.
   Arguments: x (float), y (float), radius (float)
   Usage: Action: simulate_action
          Action Input: {"x": 0.5, "y": 4.0, "radius": 0.6}

3. get_contact_log
   Description: After running a simulation, returns the contact events:
which objects touched and when.
   Arguments: None
   Usage: Action: get_contact_log

4. simulate_partial
   Description: Place a red ball and run the simulation only up to the
specified step. Returns object positions and velocities at that point.
Useful for observing mid-simulation dynamics.
   Arguments: x (float), y (float), radius (float), stop_step (int)
   Usage: Action: simulate_partial
          Action Input: {"x": 0.5, "y": 4.0, "radius": 0.6, "stop_step":
 50}

5. compute_gap_analysis
   Description: Analyze the gaps on each side of the platform.
    Returns the left gap and right gap, and whether the green ball can
fit through each gap.
   Arguments: None
   Usage: Action: compute_gap_analysis

6. finish
   Description: Submit your final answer. Use this when you are
confident in your solution.
   Arguments: x (float), y (float), radius (float)
   Usage: Action: finish
          Action Input: {"x": 0.5, "y": 4.0, "radius": 0.6}


To solve this puzzle, you will reason step-by-step and use tools to test
 your ideas.

At each step, you MUST follow this exact format:

Thought: <your reasoning about what to do next>
Action: <tool name>
Action Input: <JSON arguments, or leave blank for tools with no
arguments>

After you take an action, you will receive:

Observation: <result from the tool>

Then you continue with another Thought/Action cycle.

When you are confident in your answer, use the "finish" tool to submit
it.

Important rules:
- Always start with a Thought before taking an Action.
- Only call ONE tool per step.
- Parse observation results carefully before your next thought.
- You can simulate multiple different actions to compare results.
- Each simulation resets the environment, so previous simulations don't
affect new ones.
```

I.1.2 TWO BODY PROBLEM

```
You are an expert physics reasoning agent solving a 2D physics puzzle.
You have access to a physics simulator and can test your ideas before
submitting a final answer.

**Puzzle: Two Body Problem**

The environment is a 2D box with coordinates ranging from -5 to 5 on
both axes. Gravity pulls objects downward.

**Key Elements:**
- **Green Ball:** A dynamic ball.
- **Blue Ball:** A dynamic ball, separated horizontally from the green
ball.
- Both balls fall under gravity from rest.

**The Goal:**
Place ONE Red Ball at t=0 so that the Green Ball collides with the Blue
Ball and stays in contact.

**Placement Constraints:**
- The red ball must be completely inside the box: -5 + radius <= x <= 5
- radius, -5 + radius <= y <= 5 - radius.
- The red ball must NOT overlap with the green ball or the blue ball at
t=0.
- 0.1 <= radius <= 1.5

You have access to the following tools to interact with the physics
simulation:


1. get_level_state
   Description: Get the current level layout including all object
positions, sizes, and properties.
   Arguments: None
   Usage: Action: get_level_state

2. simulate_action
   Description: Place a red ball at (x, y) with the given radius and run
 the full physics simulation to completion. Returns whether the goal was
 achieved, final positions of all objects, and total simulation steps.
If the placement is invalid (out of bounds or overlaps), returns a
detailed error with how far to move the ball.
   Arguments: x (float), y (float), radius (float)
   Usage: Action: simulate_action
          Action Input: {"x": 0.5, "y": 4.0, "radius": 0.6}

3. get_contact_log
   Description: After running a simulation, returns the contact events:
which objects touched and when.
   Arguments: None
   Usage: Action: get_contact_log

4. simulate_partial
   Description: Place a red ball and run the simulation only up to the
specified step. Returns object positions and velocities at that point.
Useful for observing mid-simulation dynamics.
   Arguments: x (float), y (float), radius (float), stop_step (int)
   Usage: Action: simulate_partial
          Action Input: {"x": 0.5, "y": 4.0, "radius": 0.6, "stop_step":
 50}

5. compute_relative_positions
```

```
   Description: Analyze the positions of the green and blue balls.
   Returns their coordinates, distance, on which side the blue ball is
    relative to green, and recommended red ball placement direction.
   Arguments: None
   Usage: Action: compute_relative_positions

6. finish
   Description: Submit your final answer. Use this when you are
confident in your solution.
   Arguments: x (float), y (float), radius (float)
   Usage: Action: finish
          Action Input: {"x": 0.5, "y": 4.0, "radius": 0.6}


To solve this puzzle, you will reason step-by-step and use tools to test
 your ideas.

At each step, you MUST follow this exact format:

Thought: <your reasoning about what to do next>
Action: <tool name>
Action Input: <JSON arguments, or leave blank for tools with no
arguments>

After you take an action, you will receive:

Observation: <result from the tool>

Then you continue with another Thought/Action cycle.

When you are confident in your answer, use the "finish" tool to submit
it.

Important rules:
- Always start with a Thought before taking an Action.
- Only call ONE tool per step.
- Parse observation results carefully before your next thought.
- You can simulate multiple different actions to compare results.
- Each simulation resets the environment, so previous simulations don't
affect new ones.
```

I.1.3 PASS THE PARCEL

```
You are an expert physics reasoning agent solving a 2D physics puzzle.
You have access to a physics simulator and can test your ideas before
submitting a final answer.

**Puzzle: Pass The Parcel**

The environment is a 2D box with coordinates ranging from -5 to 5 on
both axes. Gravity pulls objects downward.

**Key Elements:**
- **Top Basket (Gray, inverted):** A dynamic basket sitting on the black
 platform with its opening facing DOWNWARD. It traps the green ball
underneath it.
- **Green Ball:** A small dynamic ball trapped beneath the inverted top
basket on the platform.
- **Bottom Basket (Gray, upright):** A dynamic basket below the platform
 with its opening facing UPWARD. It holds the blue ball.
- **Blue Ball:** A dynamic ball sitting inside the bottom basket. This
is the target --- the green ball must touch it.
```

```
- **Black Platform:** A static horizontal bar. The top basket and (
initially) red ball sit on it.
- **Ramp (Black):** A static angled bar rising from the left edge of the
 platform upward to the right. Useful for rolling the red ball down onto
 the top basket.

**The Goal:**
Place ONE Red Ball so that the Green Ball contacts the Blue Ball for at
least 3 seconds.

**Placement Constraints:**
- The red ball must be completely inside the box: -5 + radius <= x <= 5
- radius, -5 + radius <= y <= 5 - radius.
- The red ball must NOT overlap with existing objects at t=0.
- 0.1 <= radius <= 2.0


You have access to the following tools to interact with the physics
simulation:


1. get_level_state
   Description: Get the current level layout including all object
positions, sizes, and properties.
   Arguments: None
   Usage: Action: get_level_state

2. simulate_action
   Description: Place a red ball at (x, y) with the given radius and run
 the full physics simulation to completion. Returns whether the goal was
 achieved, final positions of all objects, and total simulation steps.
If the placement is invalid (out of bounds or overlaps), returns a
detailed error with how far to move the ball.
   Arguments: x (float), y (float), radius (float)
   Usage: Action: simulate_action
          Action Input: {"x": 0.5, "y": 4.0, "radius": 0.6}

3. get_contact_log
   Description: After running a simulation, returns the contact events:
which objects touched and when.
   Arguments: None
   Usage: Action: get_contact_log

4. simulate_partial
   Description: Place a red ball and run the simulation only up to the
specified step. Returns object positions and velocities at that point.
Useful for observing mid-simulation dynamics.
   Arguments: x (float), y (float), radius (float), stop_step (int)
   Usage: Action: simulate_partial
          Action Input: {"x": 0.5, "y": 4.0, "radius": 0.6, "stop_step":
 50}

5. get_ramp_center
   Description: Analyze the pass_the_parcel setup. Returns the center of
 the ramp.
   Arguments: None
   Usage: Action: get_ramp_center

6. finish
   Description: Submit your final answer. Use this when you are
confident in your solution.
   Arguments: x (float), y (float), radius (float)
   Usage: Action: finish
          Action Input: {"x": 0.5, "y": 4.0, "radius": 0.6}
```

```
To solve this puzzle, you will reason step-by-step and use tools to test
 your ideas.

At each step, you MUST follow this exact format:

Thought: <your reasoning about what to do next>
Action: <tool name>
Action Input: <JSON arguments, or leave blank for tools with no
arguments>

After you take an action, you will receive:

Observation: <result from the tool>

Then you continue with another Thought/Action cycle.

When you are confident in your answer, use the "finish" tool to submit
it.

Important rules:
- Always start with a Thought before taking an Action.
- Only call ONE tool per step.
- Parse observation results carefully before your next thought.
- You can simulate multiple different actions to compare results.
- Each simulation resets the environment, so previous simulations don't
affect new ones.
```

I.1.4 CATAPULT

```
You are an expert physics reasoning agent solving a 2D physics puzzle.
You have access to a physics simulator and can test your ideas before
submitting a final answer.

**Puzzle: Catapult**

The environment is a 2D box with coordinates ranging from -5 to 5 on
both axes. Gravity pulls objects downward.

**Key Elements (factual --- no implied approach):**
- **Green Ball:** A small dynamic ball sitting on the LEFT end of a gray
 bar.
- **Gray Bar (Catapult Arm):** A dynamic lever resting on a gray ball (
pivot). The green ball sits on its left end.
- **Gray Ball (Pivot):** A dynamic ball acting as the fulcrum. It sits
on the left black platform.
- **Black Ball (Ceiling Blocker):** A static ball near the top of the
scene.
- **Black Platform (Left):** A static horizontal platform on the left
side.
- **Black Ledge (Right):** A static (possibly angled) platform on the
right side.
- **Basket (Gray):** A dynamic basket sitting on the right ledge.
- **Blue Ball (Target):** A dynamic ball inside the basket.

**The Goal:**
Place ONE Red Ball somewhere in the box so that, once the simulation
runs, the green ball contacts the blue ball for at least 3 seconds. The
success condition is ONLY the green-blue contact --- how you achieve it
is your choice.

**Placement Constraints:**
```

```
- The red ball must be completely inside the box: -5 + radius <= x <= 5
- radius, -5 + radius <= y <= 5 - radius.
- The red ball must NOT overlap with any existing object at t=0.
- 0.1 <= radius <= 2.0


You have access to the following tools to interact with the physics
simulation:

1. describe_scene_geometry
   Description: Return strategy-neutral geometry: every ball (position,
radius, dynamic flag), every bar (position, angle, length, dynamic flag)
, every basket (position, dynamic flag), and the key distance (green <->
 blue). No prescriptive advice; you interpret the layout to form a
strategy.
   Arguments: None
   Usage: Action: describe_scene_geometry


1. get_level_state
   Description: Get the current level layout including all object
positions, sizes, and properties.
   Arguments: None
   Usage: Action: get_level_state

2. simulate_action
   Description: Place a red ball at (x, y) with the given radius and run
 the full physics simulation to completion. Returns whether the goal was
 achieved, final positions of all objects, and total simulation steps.
If the placement is invalid (out of bounds or overlaps), returns a
detailed error with how far to move the ball.
   Arguments: x (float), y (float), radius (float)
   Usage: Action: simulate_action
          Action Input: {"x": 0.5, "y": 4.0, "radius": 0.6}

3. get_contact_log
   Description: After running a simulation, returns the contact events:
which objects touched and when.
   Arguments: None
   Usage: Action: get_contact_log

4. simulate_partial
   Description: Place a red ball and run the simulation only up to the
specified step. Returns object positions and velocities at that point.
Useful for observing mid-simulation dynamics.
   Arguments: x (float), y (float), radius (float), stop_step (int)
   Usage: Action: simulate_partial
          Action Input: {"x": 0.5, "y": 4.0, "radius": 0.6, "stop_step":
 50}

6. trace_green_ball
   Description: Lightweight trajectory probe --- only the green ball is
sampled. Places a red ball, runs the simulation, and returns the green
ball's (x, y) waypoints at fixed step intervals plus start/end/peak
summary. Stops early once the green ball comes to rest (capped at ~600
steps). Use this when you only care about WHERE the green ball travels,
not contact events or other objects --- much cheaper than
simulate_with_trace.
   Arguments: x (float), y (float), radius (float)
   Usage: Action: trace_green_ball
          Action Input: {"x": 1.2, "y": 3.5, "radius": 0.6}

7. predict_first_contact
   Description: Cheap pre-simulation check (<=90 physics steps, ~1.5s of
 sim time). Runs just long enough to find the FIRST object the red ball
```

```
touches after it is released, and reports: placement validity, the other
 object's name, the step of impact, approach speed, approximate contact
point, and surface normal. Use this to verify that your red ball
actually reaches the object you intended to hit BEFORE burning a full
simulation budget.
   Arguments: x (float), y (float), radius (float)
   Usage: Action: predict_first_contact
          Action Input: {"x": 1.2, "y": 3.5, "radius": 0.6}

8. simulate_with_trace
   Description: Place a red ball and run the simulation. Returns:
success flag, contact events involving the red ball or YOUR chosen
objects (via object_names), and per-object kinematic extrema (peak_y,
min_y, max_speed, displacement, and angular stats for moving bars/
baskets). You choose which objects to trace---e.g., ["green_ball"] to
see if it launches, ["basket","blue_ball"] to see if the basket is
disturbed, ["catapult_arm","green_ball"] to see how the lever moves the
green ball.
   Arguments: x (float), y (float), radius (float), object_names (list
of strings), n_samples (int, optional, unused), stop_step (int, optional
, default = run to completion)
   Usage: Action: simulate_with_trace
          Action Input: {"x": 1.2, "y": 3.5, "radius": 0.6, "
object_names": ["green_ball", "catapult_arm"]}

9. finish
   Description: Submit your final answer. Use this when you are
confident in your solution.
   Arguments: x (float), y (float), radius (float)
   Usage: Action: finish
          Action Input: {"x": 0.5, "y": 4.0, "radius": 0.6}

To solve this puzzle, you will reason step-by-step and use tools to test
 your ideas.

At each step, you MUST follow this exact format:

Thought: <your reasoning about what to do next>
Action: <tool name>
Action Input: <JSON arguments, or leave blank for tools with no
arguments>

After you take an action, you will receive:

Observation: <result from the tool>

Then you continue with another Thought/Action cycle.

When you are confident in your answer, use the "finish" tool to submit
it.

Important rules:
- Always start with a Thought before taking an Action.
- Only call ONE tool per step.
- Parse observation results carefully before your next thought.
- You can simulate multiple different actions to compare results.
- Each simulation resets the environment, so previous simulations don't
affect new ones.
```

I.1.5 FALLING INTO PLACE

```
You are an expert physics reasoning agent solving a 2D physics puzzle.
You have access to a physics simulator and can test your ideas before
submitting a final answer.

**Puzzle: Falling Into Place**

The environment is a 2D box with coordinates ranging from -5 to 5 on
both axes. Gravity pulls objects downward.

**Key Elements:**
- **Green Ball:** A dynamic ball sitting on one of the two black
platforms (left or right side).
- **Left Platform / Right Platform (Black Bars):** Two static horizontal
 platforms with a gap between them in the center.
- **Bottom Ramp (Black Bar):** A slightly angled static bar near the
bottom of the scene.
- **Blue Jar (dynamic Basket):** A dynamic basket positioned above with
its opening facing DOWNWARD. It falls due to gravity.

**The Goal:**
Place ONE Red Ball so that the Green Ball touches the Blue Jar for at
least 3 seconds.

**Placement Constraints:**
- The red ball must be completely inside the box: -5 + radius <= x <= 5
- radius, -5 + radius <= y <= 5 - radius.
- The red ball must NOT overlap with any existing object at t=0.
- 0.1 <= radius <= 2.0

You have access to the following tools to interact with the physics
simulation:

1. get_level_state ... 2. simulate_action ... 3. get_contact_log ... 4.
simulate_partial ...

5. compute_intercept_setup
   Description: Computes intercept geometry for the falling_into_place
level. ...
   Arguments: None
   Usage: Action: compute_intercept_setup

6. finish ...

To solve this puzzle, you will reason step-by-step and use tools to test
 your ideas. ...
```

I.1.6 BASKET CASE

```
You are an expert physics reasoning agent solving a 2D physics puzzle.
You have access to a physics simulator and can test your ideas before
submitting a final answer.

**Puzzle: Basket Case**

The environment is a 2D box with coordinates ranging from -5 to 5 on
both axes. Gravity pulls objects downward.

**Key Elements:**
- **Green Ball:** A dynamic ball positioned high above the basket. It
falls straight down due to gravity.
- **Basket (Gray):** A dynamic container sitting near the purple ground.
 Its opening faces UPWARD. Without intervention, the green ball falls
directly into it and gets trapped.
```

```
- **Purple Ground:** A static bar at the very bottom of the scene (y ~
-5). This is the target surface.

**The Goal:**
Place ONE Red Ball so that the Green Ball touches the Purple Ground for
at least 3 seconds. The green ball starts directly above the basket and
will fall into it unless deflected sideways.

**Placement Constraints:**
- The red ball must be completely inside the box: -5 + radius <= x <= 5
- radius, -5 + radius <= y <= 5 - radius.
- The red ball must NOT overlap with the green ball or basket at t=0.
- 0.1 <= radius <= 2.0

You have access to the following tools to interact with the physics
simulation:

1. get_level_state ... 2. simulate_action ... 3. get_contact_log ... 4.
simulate_partial ...

5. compute_basket_analysis
   Description: Analyze the basket case setup. ...
   Arguments: None
   Usage: Action: compute_basket_analysis

6. finish ...

To solve this puzzle, you will reason step-by-step and use tools to test
 your ideas. ...
```

I.1.7 CLIFFHANGER

```
You are an expert physics reasoning agent solving a 2D physics puzzle.
You have access to a physics simulator and can test your ideas before
submitting a final answer.

**Puzzle: Cliffhanger**

The environment is a 2D box with coordinates ranging from -5 to 5 on
both axes. Gravity pulls objects downward.

**Key Elements (factual --- no implied approach):**
- **Green Bar:** A dynamic, vertical bar (length 2.0--3.0) standing
upright on the black platform near one of the platform's edges.
- **Black Platform:** A static horizontal bar (length 4.0--6.0) at
variable height y in [-3, 0]; the green bar stands on top of it.
- **Ceiling:** A static horizontal bar spanning the box, positioned
above the platform (y above the green bar's top).
- **Purple Ground:** The static floor at the bottom of the box (y ~= -5)
.

**The Goal:**
Place ONE Red Ball somewhere in the box so that, once the simulation
runs, the green bar contacts the purple ground for at least 3 seconds.
The success condition is ONLY the green-bar / purple-ground contact ---
how you achieve it is your choice.

**Placement Constraints:**
- The red ball must be completely inside the box: -5 + radius <= x <= 5
- radius, -5 + radius <= y <= 5 - radius.
- The red ball must NOT overlap with any existing object at t=0.
- 0.1 <= radius <= 2.0
```

```
You have access to the following tools to interact with the physics
simulation:

1. get_level_state ... 2. simulate_action ... 3. get_contact_log ... 4.
simulate_partial ...

5. compute_cliffhanger_analysis
   Description: Analyse the cliffhanger geometry. ...
   Arguments: None
   Usage: Action: compute_cliffhanger_analysis

6. finish ...

To solve this puzzle, you will reason step-by-step and use tools to test
 your ideas. ...
```

I.1.8 TIPPING POINT

```
You are an expert physics reasoning agent solving a 2D physics puzzle.

Puzzle: Tipping Point

The environment is a 2D box with coordinates ranging from -5 to 5 on
both axes. Gravity pulls objects downward.

Key Elements (factual):
- Green Bar: dynamic vertical bar (length 2.0--5.0) resting upright in a
 small gray basket on the ground.
- Gray Basket: small dynamic basket holding the green bar near the
bottom of the box.
- Purple Wall: static vertical bar flush against the LEFT (x ~= -4.9) or
 RIGHT (x ~= 4.9) side of the box; spans top to bottom.
- Box bounds: x in [-5, 5], y in [-5, 5].

Goal: Place ONE Red Ball so that green_bar contacts purple_wall and
maintains contact for the success-time duration. The path to that
outcome is yours to design.

Placement Constraints:
- Red ball must be inside the box: -5 + radius <= x <= 5 - radius, -5 +
radius <= y <= 5 - radius.
- No overlap with existing objects at t=0.
- 0.1 <= radius <= 2.0

When your simulation shows green_bar in continuous contact with
purple_wall for the required duration, call finish.

You have access to the following tools to interact with the physics
simulation:

1. get_level_state ... 2. simulate_action ... 3. get_contact_log ... 4.
simulate_partial ...

5. compute_tipping_point_analysis
   Description: Analyse tipping_point geometry. ...
   Arguments: None
   Usage: Action: compute_tipping_point_analysis

6. finish ...

To solve this puzzle, you will reason step-by-step and use tools to test
 your ideas. ...
```

I.2 HExA Evolver Prompts

These are the prompts presented to the evolver LLM $\mathcal{M}_\theta$ that drive every skill-distillation result in this paper. Their mapping to the experiments in §H and §3.4 is summarised in Table 20. Placeholder fields shown in curly braces (`{level_name}`, `{success_block}`, `{n_sources}`, etc.) are filled in at run time from the level identifier, the rendered trajectory blocks, and other context. Double-braces `{{...}}` appear as single braces at run time and are used to denote literal JSON braces inside the prompt body.

I.2.1 Pass 1 — Contrastive Skill Distillation

Every HExA round invokes this prompt to extract *strategy skills* by contrasting high-reward against low-reward trajectories within the same level batch. The reward function from Eq. 15 is communicated to the evolver as a per-trajectory `Reward` annotation embedded in the rendered success and failure blocks.

```
You are an expert physics analyst distilling agent behavior into concise
, actionable skills.

**Environment**: 2D physics simulation (Box2D). Gravity = -9.8 m/s².
World bounds [-5, 5] on both axes. The agent places a red ball at (x, y)
 with a given radius, then the simulation runs to completion.

**Level: {level_name}**
{level_description}

Below are SUCCESSFUL and FAILED trajectories from the same level. Each
shows the agent's reasoning (Thought), actions taken, and simulation
observations. Each trajectory has a **Reward** score:
- Successes: +1.0 (solved fast, 1-3 iters) to +0.25 (solved slowly,
16-25 iters)
- Failures: -0.5 (tried all iterations) to -0.75 (gave up early)

**Weight your analysis by reward** -- skills from high-reward
trajectories (fast solves) are more reliable than skills from low-reward
 ones (barely solved).

=== SUCCESSFUL TRAJECTORIES ({n_successes}) ===
{success_block}

=== FAILED TRAJECTORIES ({n_failures}) ===
{failure_block}

---

Your task: By CONTRASTING the successes and failures, extract the KEY
PHYSICS SKILLS that distinguish solving from failing. Focus especially
on high-reward successes -- what insight let the agent solve it quickly?
 What did the failed agents miss?

For each skill, provide:
- **title**: Short name (3-7 words)
- **principle**: The physics insight (2-3 sentences). What mechanism was
 exploited? Why does it work?
- **when_to_apply**: Specific trigger condition (1 sentence)
- **source_seeds**: List of seed numbers from the trajectories above
that this skill was primarily derived from. Include only the seeds whose
 behavior directly demonstrates or motivates this skill.

Output a JSON array:
```json
[
  {{
```
```

```
    "title": "...",
    "principle": "...",
    "when_to_apply": "...",
    "source_seeds": [1, 5, 16]
  }}
]
```

Extract 4-6 skills. Each should capture a DISTINCT insight from the success/failure contrast. Avoid redundancy.
```

#### I.2.2 PASS 2 — MISTAKE & PARTIAL-SKILL EXTRACTION

Run on the failure subset $\mathcal{T}_\ell^{(n),-}$ alone. Produces (i) structured *mistake* records $\mu = (\delta_\mu, \rho_\mu, \alpha_\mu)$ describing recurring failure modes, and (ii) *partial skills* extracted from individual correct steps inside otherwise-failed trajectories.

```
You are an expert at analyzing agent failures and distilling them into
avoidable mistake patterns.

Environment: 2D physics simulation (Box2D). Gravity = -9.8 m/s^2.
World bounds [-5, 5].

Level: {level_name}
{level_description}

Below are FAILED trajectories. Each shows the agent's reasoning, actions
, and simulation results.

{failure_block}

Your task has TWO parts:

Part 1 -- Mistakes: Identify the COMMON MISTAKE PATTERNS across
these failures. For each mistake, analyze:
1. What exactly the agent did wrong
2. WHY the agent made this error (what broken causal belief led to it)
3. A concrete actionable fix

Part 2 -- Partial insights: Even in failed trajectories, some
individual steps show CORRECT physics reasoning or useful discoveries (e
.g., the agent found a valid placement region but then abandoned it, or
correctly identified a mechanism but applied it with wrong parameters).
Extract 1-2 skills from these "good steps within bad trajectories".
These should be genuine physics insights, not just restating what went
wrong.

Format as JSON object with two arrays:
```json
{{
  "mistakes": [
    {{
      "description": "What the mistake is (1 sentence)",
      "why_it_happens": "The broken belief or reasoning error that
causes this (1 sentence)",
      "how_to_avoid": "Concrete actionable fix -- what to do instead
(1-2 sentences)"
    }}
  ],
  "partial_skills": [
    {{
```

```
      "title": "Short name (3-7 words)",
      "principle": "The physics insight from the failed trajectory (2-3
sentences)",
      "when_to_apply": "Specific trigger condition (1 sentence)",
      "source_seeds": [5, 11]
    }}
  ]
}}
```

```
Extract 3-5 mistakes and 1-2 partial skills. For mistakes, group similar
 failures into one and focus on ROOT CAUSES. For partial skills, only
extract genuinely useful insights -- do not force it if no good steps
exist.
```

I.2.3 SKILL-BANK EVOLUTION

Used by the evolving variants (Off2On Evol., Online Evol.) at every round $n \geq 1$. Receives both the current bank $\mathcal{K}_\ell^{(n)}$ and the new trajectory batch $\mathcal{T}_\ell^{(n)}$, and emits the evolved bank $\mathcal{K}_\ell^{(n+1)}$ (Eq. 5) subject to the per-level skill budget B and the mistake cap. The evolver decides which existing skills to retain (with their reward preserved), which to remove, and which to add (with skill reward assigned via Equation 16).

```
You are a physics evolver evolving a skill bank for puzzle-solving
agents.

LEVEL: {level_name}
LEVEL DESCRIPTION: {level_description}

CURRENT SKILL BANK (from previous rounds):
{existing_skills_block}

NEW TRAJECTORIES (from the latest round):
Successes: {n_successes}
Failures: {n_failures}

{new_trajectories_block}

YOUR TASK: Evolve the skill bank by merging the existing skills with
insights from the new trajectories.

RULES:
1. Output the COMPLETE FINAL skill bank (not a diff) -- include both
retained existing skills and any new ones.
2. Hard constraints:
   - Maximum {max_skills} total skills for this level
   - Maximum {max_mistakes} total mistakes for this level
3. For each skill you include:
   - If it's a RETAINED skill from the existing bank: set "is_new":
false
   - If it's a NEW skill extracted from the new trajectories: set "
is_new": true
   - Include "source_seeds" listing seed numbers where this skill was
observed (required for confidence calibration)
   - Include "confidence": a float in [0.1, 1.0] representing your
confidence in this skill
4. For retained skills, preserve their existing confidence values (they'
ve been validated).
5. For new skills, estimate confidence based on:
   - Success rate among source trajectories (high success = high
confidence)
   - Universality (applies to multiple seed conditions = higher
confidence)
```

```
   - Clarity and actionability of the principle
6. Do not include duplicate skills. If a new trajectory confirms an
existing skill, keep the existing one (possibly with slightly higher
confidence).
7. Remove skills that are:
   - Redundant or subsumed by other skills.
   - Contradicted by the new trajectories
   - Too specific or rarely applicable
   - Low confidence (< 0.3) and not directly observed in new
trajectories
8. Do not remove mistakes unless the new trajectories show they're no
longer common.

OUTPUT JSON OBJECT:
{{
  "skills": [
    {{
      "title": "<short name of skill>",
      "principle": "<2-3 sentence physics insight>",
      "when_to_apply": "<condition for applicability>",
      "example": "<optional concrete coordinate example>",
      "source_seeds": [<seed numbers>],
      "confidence": <float in [0.1, 1.0]>,
      "is_new": <true|false>
    }}
    ...
  ],
  "mistakes": [
    {{
      "description": "<what the mistake is>",
      "why_it_happens": "<why agents make this mistake>",
      "how_to_avoid": "<actionable fix>",
      "is_new": <true|false>
    }}
    ...
  ],
  "removed_skill_titles": ["<title of removed skill 1>", ...],
  "reasoning": "<brief explanation of key changes: what was removed,
what was added, why>"
}}

Be concise but precise. Focus on physics insights that directly help
puzzle-solving.
```

I.2.4 CATAPULT-SPECIFIC VARIANTS (CLAUDE)

The within-level HExA catapult result in Table 8 was produced with catapult-specific variants of all three evolver prompts above. The variants differ from the generic prompts in two ways: (i) a *factual* catapult scene block (objects, success condition, placement constraints) is injected after the **Level:** header so the evolver knows the scene without seeing strategy hints, and (ii) minor task-framing wording is tightened (e.g. "Let the trajectories — not your prior expectations — determine the mechanisms that work").

Shared catapult scene description.

```
**Scene (factual -- no approach implied):**
- Green Ball: small dynamic ball on the LEFT end of a gray bar.
- Gray Bar (Catapult Arm): dynamic lever resting on a gray ball (pivot);
 the green ball sits on its left end.
- Gray Ball (Pivot): dynamic ball acting as the fulcrum; sits on the
left black platform.
- Black Ball (Ceiling Blocker): static ball near the top of the scene.
- Black Platform (Left): static horizontal platform on the left side.
```

```
- Black Ledge (Right): static (possibly angled) platform on the right
side.
- Basket (Gray): dynamic basket sitting on the right ledge.
- Blue Ball (Target): dynamic ball inside the basket.

**Success condition:** The green ball must contact the blue ball for at
least 3 seconds after the red ball is placed and the simulation runs.

**Placement constraints:**
- The red ball must be completely inside the box: -5 + r <= x <= 5 - r,
-5 + r <= y <= 5 - r.
- The red ball must NOT overlap with any existing object at t=0.
- 0.1 <= radius <= 2.0.
```

Pass 1 — contrastive (catapult).

```
You are an expert physics analyst distilling agent behavior into concise
, actionable skills.

**Environment**: 2D physics simulation (Box2D). Gravity = -9.8 m/s^2.
...

**Level: {level_name}**
{catapult scene description}

Below are SUCCESSFUL and FAILED trajectories from this level. ... Each
trajectory has a **Reward** score:
- Successes: +1.0 (solved fast, 1-3 iters), +0.75 (4-7 iters), +0.5
(8-15 iters), +0.25 (solved slowly, 16-25 iters)
- Failures: -0.5 (tried all 25 iters), -0.75 (gave up early, <10 iters)
...

Your task: By CONTRASTING the successes and failures, extract the KEY
PHYSICS SKILLS that distinguish solving from failing on this catapult
level. Focus on what high-reward successes DID that low-reward failures
missed. Let the trajectories -- not your prior expectations -- determine
 the mechanisms that work.

For each skill, provide:
- **title**: ... - **principle**: ... - **when_to_apply**: ... - **
source_seeds**: ...
...
Extract 4-6 skills. Each should capture a DISTINCT insight from the
success/failure contrast. Avoid redundancy.
```

Pass 2 — mistakes & partial skills (catapult).

```
You are an expert at analyzing agent failures and distilling them into
avoidable mistake patterns.

**Environment**: 2D physics simulation (Box2D). Gravity = -9.8 m/s^2.
World bounds [-5, 5].

**Level: {level_name}**
{catapult scene description}

Below are FAILED trajectories. ...
{failure_block}
...
Extract 3-5 mistakes and 1-2 partial skills. ...
```

Skill-bank evolution (catapult).

```
You are a physics evolver evolving a skill bank for a catapult-puzzle-
solving agent.

**Environment**: 2D physics simulation (Box2D). Gravity = -9.8 m/s^2.
World bounds [-5, 5].

**Level: {level_name}**
{catapult scene description}

CURRENT SKILL BANK (from previous rounds):
{existing_skills_block}

NEW TRAJECTORIES (from the latest round):
...
RULES:
1. Output the COMPLETE FINAL skill bank (not a diff) ...
...
8. Do not remove mistakes unless the new trajectories show they're no
longer common.

OUTPUT JSON OBJECT:
{{ "skills": [...], "mistakes": [...], "removed_skill_titles": [...], "
reasoning": "..." }}

Be concise but precise. Focus on physics insights grounded in the
trajectories.
```

I.2.5 CROSS-LEVEL SYNTHESIS — FULL (CLAUDE)

Used for the catapult cross-level result in Table 19 where the evolver is Claude Sonnet 4.6 and three source banks are consumed (`down_to_earth`, `two_body_problem`, `pass_the_parcel`). The eight hard constraints — citation requirement, no invented coordinates, target-side entities only, no platitudes, mandatory transfer rationale, confidence calibration, no redundancy, mistake constraints — are enumerated explicitly so that every synthesised skill remains traceable to the source skill IDs that motivated it.

```
You are an expert physics analyst performing CROSS-LEVEL OFFLINE SKILL
TRANSFER.

You will be given expert skill banks from {n_sources} SOURCE physics
levels.
Each source bank was distilled and evolved over many rounds of an agent
solving
that level. You will also be given a factual description of a TARGET
level
that the agent has NEVER attempted yet -- no target-level trajectories,
no target-level skills, no target-level success/failure data exist.

Your task: synthesise a TARGET skill bank by extracting transferable
physics
PRINCIPLES from the source banks, re-grounding them in the target scene,
 and
predicting which principles the target level will reward.

================================================================================

**Environment (shared across all levels)**: 2D physics simulation (Box2D
).
Gravity = -9.8 m/s^2. World bounds [-5, 5] on both axes. The agent
places ONE
```

```
red ball at (x, y) with a chosen radius (0.1 <= r <= 2.0) at t=0; the
simulation
then runs to completion. Mass scales as r^3; momentum and impact force
scale
with both mass and the ball's velocity at first contact.
================================================================================


=== SOURCE LEVELS ===

{source_blocks}

=== TARGET LEVEL: {target_level} ===

{target_block}

=== TASK ===

Produce a SKILL BANK for the target level. The bank must contain:
- 6 to 10 SKILLS (physics principles you predict will help on the target
).
- 2 to 4 MISTAKES (anti-patterns transferable from source failures).

=== HARD CONSTRAINTS ===

1. **Every skill MUST cite source skills.** For each output skill,
include
   `source_skills`: a non-empty list of objects of the form
   `{{"source_level": "<level>", "skill_id": "<id>"}}`, listing the
source-bank
   entries that motivated this skill. If you cannot cite at least one
source
   skill, do NOT emit the skill -- synthesise something else instead.

2. **No invented coordinates.** You have NEVER seen the target level
solved.
   Do NOT predict specific (x, y, r) values that "work" for the target.
The
   `example` field, if used, must describe a *qualitative* placement
   (e.g. "place near the end of the lever opposite the green ball, with
   r large enough to dominate the lever's mass") -- never specific
numbers.

3. **Use target-level entities.** Phrase each skill in terms of the
entities
   that exist in the TARGET scene description above. Do NOT mention
source-
   level-only entities (e.g. "ramp", "inverted basket") in the principle
 or
   when_to_apply, even if those entities motivated the transfer. The
   `transfer_rationale` field is where you may name source entities to
   explain the analogy.

4. **No platitudes.** Skills like "use physics intuition", "consider
gravity",
   "be careful" are forbidden. Each skill must name a SPECIFIC mechanism
   (e.g. "torque about a pivot scales with moment arm x force"; "the
line of
   centres at first contact determines the post-collision velocity
direction").

5. **Each skill must include a `transfer_rationale`** (1-2 sentences)
explaining
   what physics primitive bridges the source observation to the target
   prediction. This is the audit trail for your synthesis.
```

```
6. **Confidence calibration.** Weight a skill's confidence by:
   - How many source banks corroborate the underlying primitive (more =
higher).
   - How directly the primitive applies to the target scene (direct =
higher).
   - How load-bearing the source skill was (high source confidence =
higher).
   Confidence is a float in [0.1, 1.0]. Assign 0.7+ only when the
primitive
   appears in >=2 source banks AND the target scene clearly invokes it.

7. **Avoid redundancy.** Each skill must capture a DISTINCT mechanism.
Do not
   emit two skills that are paraphrases of the same idea.

8. **Mistakes**: derive each from source-bank mistakes or from the
failure
   modes those mistakes imply. Apply the same hard constraints (cite
sources,
   no invented coordinates, target-side entities, specific mechanism).

=== OUTPUT FORMAT (single JSON object) ===

```json
{{
  "skills": [
    {{
      "title": "<3-7 word name>",
      "principle": "<2-3 sentence physics insight, in target-level terms
>",
      "when_to_apply": "<specific condition, in target-level terms>",
      "example": "<qualitative placement description, NO numbers -- or
omit>",
      "confidence": <float in [0.1, 1.0]>,
      "source_skills": [
        {{"source_level": "<level>", "skill_id": "<id>"}}
      ],
      "transfer_rationale": "<1-2 sentences naming the bridging physics
primitive>"
    }}
  ],
  "mistakes": [
    {{
      "description": "<1 sentence in target-level terms>",
      "why_it_happens": "<1 sentence root cause>",
      "how_to_avoid": "<1-2 sentence concrete fix>",
      "source_skills": [
        {{"source_level": "<level>", "skill_id": "<id>"}}
      ],
      "transfer_rationale": "<1-2 sentences>"
    }}
  ],
  "synthesis_reasoning": "<2-4 sentences: which source levels
contributed most heavily, which primitives you treated as load-bearing,
which source skills you deliberately did NOT transfer and why>"
}}
```

Be precise and specific. Do not pad. Do not produce extra fields. Output
 ONLY
the JSON object inside a single fenced ```json block.
```

I.2.6 CROSS-LEVEL SYNTHESIS — SLIM (QWEN-7B)

Used for the Falling Into Place and Two Body Problem cross-level rows of Table 19, where the evolver is Qwen-2.5-7B. The slim variant drops audit fields (`source_skills`, `transfer_rationale`), reduces to three rules, and tightens form constraints on `principle` / `when_to_apply` / `example` to prevent the platitude-and-hallucination failure mode the long prompt produces on smaller models (e.g. inventing a lever on a level with no pivot). Output budget is reduced to 4–6 skills and 1–3 mistakes.

```
You are extracting transferable physics PRINCIPLES from one or more
source level skill banks for a NEW target level.

ENVIRONMENT: 2D physics (Box2D), gravity = -9.8 m/s^2, bounds [-5, 5].
Agent places ONE red ball (x, y, radius 0.1-2.0) at t=0; simulation runs
 to completion.

=== SOURCE LEVELS ===

{source_blocks}

=== TARGET LEVEL: {target_level} ===

{target_block}

=== TASK ===

Output a skill bank for the target level. Produce 4-6 skills and 1-3
mistakes.

RULES (FOLLOW STRICTLY):

1. SKILLS ARE PRINCIPLES, NOT COMMANDS. Each skill states a physics
observation, mechanism, or relationship the agent can REASON FROM.
Skills MUST NOT tell the agent what to do. Each field has a strict FORM:
   - `principle`: starts with a condition or relationship ("When X holds
, Y occurs", "X scales with Y", "A and B differ in Z"). NEVER an
imperative verb. FORBIDDEN openings: place / position / put / drop /
move / set / use / adjust.
   - `when_to_apply`: describes a REASONING context the agent enters ("
When evaluating whether...", "When comparing alternatives that...").
NEVER a target outcome to achieve ("when you need to tip...", "when
knocking the bar off...").
   - `example`: contrasts two physical scenarios or illustrates the
principle abstractly. NEVER a placement, never a specific scene action.

2. Each skill names a SPECIFIC mechanism. FORBIDDEN platitudes: "use
physics", "place strategically", "consider gravity", "adjust as needed",
 "be careful".

3. Use ONLY entities present in the TARGET scene above. Do not mention
source-only entities anywhere. If the target scene has no pivot, do NOT
write skills about levers or torque-around-a-pivot.

4. NO coordinates and NO placements anywhere in any field.

5. Mistakes describe MISCONCEPTIONS the agent might form (wrong mental
models), NOT actions to avoid. `how_to_avoid` is a CORRECTED
UNDERSTANDING, not a placement instruction.

OUTPUT FORMAT (single JSON object, fenced ```json block, no extra fields
):
```json
{{
  "skills": [
```
```

```
    {{"title": "...", "principle": "...", "when_to_apply": "...", "
example": "...", "confidence": 0.X}}
  ],
  "mistakes": [
    {{"description": "<misconception, NOT an action>", "why_it_happens":
 "...", "how_to_avoid": "<corrected understanding, NOT a placement
command>"}}
  ],
  "synthesis_reasoning": "<1-2 sentences: which source primitive
transferred>"
}}
```

Be concise. Output ONLY the JSON object.

I.3 REFLEXION

After a failed trial, the reflection model receives the following prompt, built from the failed REACT trajectory (observations truncated to 600 characters by head and tail) and any earlier reflections on the same seed.

```
SYSTEM:
You are analyzing a failed attempt at a 2D physics puzzle.

You will receive the full Thought/Action/Observation trajectory from a
single failed attempt by a ReAct-style agent, followed by the final
outcome and any prior reflections accumulated on this same task.

Your job: produce a short reflection (≤5 sentences) that the agent will
 read before its next attempt on the SAME task. Cover:
(a) which strategy/approach the agent pursued in this attempt;
(b) the specific kinematic, geometric, or procedural reason it failed (
cite concrete coordinates, distances, or contact events from the
observations);
(c) one concrete different action or strategy to try next -- be specific
 (object names, approximate (x, y, radius), expected mechanism).

Hard rules:
- Do NOT repeat lessons that already appear in the prior reflections
list.
- Do NOT re-state the goal or the puzzle rules. ...

USER:
Outcome: FAILURE.
Final observation:
{final_observation}

Trajectory:
--- Step 1 ---
Thought: {thought}
Action: {action}
Observation: {observation}
...

## Prior reflections (do not repeat these)
1. {prior_reflection_1}
...

Write the reflection now. Be specific and concise (≤5 sentences).
```

The returned reflection is appended to the level's REACT system prompt for the next trial:

```
{ReAct system prompt of the level}
```

```
## Past Lessons (from prior failed attempts on this seed)
1. {reflection_1}
...
Use these lessons to choose a different approach this trial. Do not
repeat the same failed strategy.
```

I.4 EPISODIC RAG

Episodic RAG makes no LLM call to build its memory. The retrieved successful and unsuccessful rollouts are rendered deterministically and appended to the level's REACT system prompt; the `Past rollout` section repeats once per retrieved record.

```
{ReAct system prompt of the level}

## Retrieved Past Rollouts (from your own past attempts on this level)

### Past rollout {i} -- {SUCCESS|FAILURE} (level {level}, seed {seed})
Level: {level}  |  Seed: {seed}
Verifier: {SUCCESS|FAILURE}
Iterations used: {iterations}
Simulator calls: {simulator_calls}
Shaped reward: {reward}
Final placement: x={x}, y={y}, radius={radius}
Final observation: {final_observation}

Rollout:
(showing last 12 of {total_steps} steps)
--- Step 1 ---
Thought: {thought}
Action: {action}
Observation: {observation}
...

These are complete past attempts with the environment's verdict. Use the
 successful rollout as a template where the geometry matches, and avoid
repeating what the unsuccessful rollout did.
```

I.5 MEMRL

Retrieval key. Both Episodic RAG and MemRL embed a geometry intent: one line per scene object (position, extents, angle, dynamic flag) followed by the offsets between the objects named by the goal. For Catapult, seed 1:

```
basket +3.50 -3.33 a+8.97 dyn
black_ball -2.46 +4.60 r+0.40 static
black_platform -1.52 -3.71 l-3.02 r-0.02 t-3.71 b-3.71 a+0.00 static
blue_ball +3.50 -2.75 r+0.38 dyn
...
green_ball->blue_ball dx+6.35 dy-0.94 d+6.42
green_ball->gray_ball dx+1.92 dy-1.10 d+2.22
...
```

Memory-building prompts. After each episode MemRL makes one LLM call to turn the trajectory into experience text: a script for a successful episode and a reflection for a failed one.

```
SYSTEM (successful episode):
You are analyzing a SUCCESSFUL attempt at a 2D physics placement puzzle.

You will receive the task (level description and scene geometry) and the
 agent's Thought/Action/Observation trajectory.
```

```
Produce a concise, high-level script that captures the essential steps
and decision points. The script must be:
1. Generic enough to apply to similar puzzle instances (different seeds,
 shifted geometry).
2. Specific enough to provide useful guidance (name objects, give
approximate coordinates or offsets, state the mechanism).
3. At most 5 high-level steps.
4. Focused on strategy and key decisions, not on individual tool calls.

Output ONLY the numbered script. ...

SYSTEM (failed episode):
You are analyzing a FAILED attempt at a 2D physics placement puzzle.

You will receive the task (level description and scene geometry) and the
 agent's Thought/Action/Observation trajectory.

This attempt failed. Analyze what went wrong and suggest improvements
for future similar tasks. Cover:
1. Incorrect assumptions the agent made about the physics or geometry.
2. Steps to improve -- cite concrete coordinates, distances, or contact
events from the observations.
3. What to avoid next time.

Keep it under 8 sentences. Output ONLY the reflection text. ...

USER (both):
Task:
{intent}

Trajectory:
{rendered_last_12_steps}
```

Actor prompt. The selected memories are ordered by their Phase-B score and appended to the level's REACT system prompt. A successful record's experience is `Task: {level} (seed {seed})`, then `SCRIPT: {script}`, then `TRAJECTORY: {rendered_last_12_steps}`; a failed record's is shown below.

```
{ReAct system prompt of the level}

## Archived Memories (from your own past experiences)

### Memory {i} -- {SUCCESS|FAILURE} (seed {seed}, utility Q={Q}, updates
={q_visits})
TASK REFLECTION:
Task: {level} (seed {seed})

What went wrong:
{reflection}

Failed approach:
{rendered_last_12_steps}
...

These experiences were retrieved because they are similar to the current
 scene and have been useful in the past. Adapt them to this seed's
geometry rather than copying coordinates blindly.
```

J PROMPTS FOR THE OTHER BENCHMARKS

This appendix reproduces the actor, reflector, summariser and evolver prompts of every arm — ZERO-SHOT, Reflexion, Episodic RAG, MemRL and HEXA — on the interactive games (§J.1), the reasoning and planning tasks (§J.2) and DISCOVERPHYSICS (§J.3); the environments themselves are described in Appendix E. Long passages are abbreviated with `...`. Fields in braces, such as `{problem}`, are filled at run time; a listing with several parts separates them with banners, which are not part of the prompt.

J.1 INTERACTIVE GAMES: MASTERMIND AND MAZE

All five arms share one actor. It receives the multi-episode system prompt of (Lin et al., 2026) and, as user messages, the raw text the environment returns; every per-turn user message is identical across arms. Each memory arm differs from ZERO-SHOT only in one block appended to the system message as `SYSTEM + "\n\n" + BLOCK`, so the system prompt below is the ZERO-SHOT prompt when `{BLOCK}` is empty. The chat is cumulative over the three episodes of an instance, except that Reflexion clears it back to the system message at each episode boundary. The reflector, the MemRL summariser and the HEXA evolver are the same model as the actor in each block (`gpt-oss-120b` or Qwen3-8B); the HEXA evolver is sent a single user message with no system prompt.

J.1.1 ACTOR SYSTEM PROMPT (ALL ARMS)

`{BLOCK}` is empty for ZERO-SHOT and is the injected block of §J.1.5 (Reflexion), §J.1.6 (Episodic RAG), §J.1.8 (MemRL) or §J.1.16 (HEXA) otherwise.

```
============================================================
  SYSTEM PROMPT
============================================================
You are solving the same task across multiple episodes with a fixed ...

{BLOCK}
```

J.1.2 MASTERMIND ENVIRONMENT MESSAGES (USER TURNS, ALL ARMS)

Configuration: code length 3, digits 1–6, no duplicates, 3 guesses per episode. `{EXAMPLE}` is a random distinct-digit guess such as `\boxed{4 3 2}`.

```
============================================================
  USER MESSAGE: OPENING OF EPISODE {k}
============================================================
[Episode {k}] New episode begins.
You are playing Mastermind.
You have to guess the secret 3 digit code within 3 turns.
The code consists of digits from 1 to 6 (inclusive). Duplicate numbers are not allowed.
After you enter your guess, I will say mark your guess with black and white pegs, where
  a black peg indicates a correct digit in the correct position, while a white peg
  indicates a correct digit in the wrong position.
After thinking, format your final answer inside \boxed{...}, for example, {EXAMPLE}.
As you play, the history of your guesses will be appended below. Use the information to
  complete the game before you run out of guesses.
Enter your first guess to start the game.

============================================================
  USER MESSAGE: FEEDBACK AFTER EACH GUESS
============================================================
At turn {t}, you guessed [{d1}, {d2}, {d3}]. This guess receives {B} black peg(s) and
  {W} white peg(s).Enter your guess.

============================================================
  USER MESSAGE: OTHER PER-TURN REPLIES
============================================================
At turn {t}, you did not provide a valid guess.Enter your guess.
At turn {t}, you guessed [{..}] which has {n} entries but the code has length 3.Enter
  your guess.
At turn {t}, you guessed [{..}], which has numbers outside the range 1 to 6.Enter your
  guess.
At turn {t}, you guessed [{..}], which has been already guessed before.Enter your guess.

============================================================
```

```
  USER MESSAGE: END OF EPISODE
============================================================
Congratulations! You guessed the correct code [{d1}, {d2}, {d3}] in {t} turns.Enter
  your guess.
You have reached the maximum number of turns.Enter your guess.

============================================================
  USER MESSAGE: EPISODE BOUNDARY (ONE MESSAGE)
============================================================
{TERMINAL_REPLY_OF_EPISODE_k}

[Episode {k+1}] New episode begins.
You are playing Mastermind.
... (opening text as above) ...
```

J.1.3 MAZE ENVIRONMENT MESSAGES (USER TURNS, ALL ARMS)

Configuration: 6×6 to 7×7 mazes, 9 moves per episode; `{up}`, `{down}`, `{left}` and `{right}` are each one of `wall`, `path` or `goal`.

```
============================================================
  USER MESSAGE: OPENING OF EPISODE {k}
============================================================
[Episode {k}] New episode begins.
You are a maze-solving agent. Your goal is to navigate from the START position to the
  GOAL position in the fewest turns possible. You are at the START position. Now you
  are at position ({x}, {y}) in the maze. Around you, up leads to {up}, down leads to
  {down}, left leads to {left}, and right leads to {right}.
Output your next move from up/down/left/right within \boxed{}.

============================================================
  USER MESSAGE: AFTER A LEGAL MOVE
============================================================
You move to {action}. Now you are at position ({x}, {y}) in the maze. Around you, up
  leads to {up}, down leads to {down}, left leads to {left}, and right leads to
  {right}.
Output your next move from up/down/left/right within \boxed{}.

============================================================
  USER MESSAGE: OTHER PER-TURN REPLIES
============================================================
You hit the wall. Please choose another direction.
Invalid action. Please try again.

============================================================
  USER MESSAGE: END OF EPISODE
============================================================
Congratulations! You arrived at the goal! Let's play again.

You have reached the maximum number of turns (9). Let's play again.

============================================================
  USER MESSAGE: EPISODE BOUNDARY (ONE MESSAGE)
============================================================
{TERMINAL_REPLY_OF_EPISODE_k}

[Episode {k+1}] New episode begins.
You are a maze-solving agent. ... (opening text as above) ...
```

J.1.4 REFLEXION: REFLECTOR PROMPTS

The reflector runs only after a failed episode. `{PAST_BLOCK}` is empty when there are no earlier reflections; `{TRIAL}` renders the finished episode, and the `(episode ended)` marker appears only on its last turn. Observations longer than 400 characters are cut to their first and last 200.

```
============================================================
  SYSTEM PROMPT (MASTERMIND)
============================================================
You are analysing a failed attempt at Mastermind.

The secret is a 3-digit code using digits 1-6 with no repeats. ...

Write a short reflection (at most 5 sentences) that the agent will ...
(a) which guesses it made and what the feedback ruled in or out;
```

```
(b) where its deduction went wrong -- a guess inconsistent with earlier ...
(c) a concrete opening guess and follow-up policy for the next attempt.

Hard rules:
- The agent will NOT have the failed transcript. ...
- Do NOT state or guess the secret code as though you ...
- Do NOT repeat lessons already listed as prior reflections.
- Output ONLY the reflection text. No preamble, no headings, no markdown.

============================================================
  SYSTEM PROMPT (MAZE)
============================================================
You are analysing a failed attempt at a grid maze navigation task.

The agent sees only its own (x, y) position and whether each ...

Write a short reflection (at most 5 sentences) that the agent will ...
(a) what search or movement strategy it actually followed;
(b) the specific reason that failed -- cite concrete coordinates, the cells ...
(c) one concrete different strategy for the next attempt, naming coordinates or ...

Hard rules:
- The agent will NOT have the failed transcript. ...
- Do NOT invent map knowledge the observations did not show.
- Do NOT repeat lessons already listed as prior reflections.
- Output ONLY the reflection text. No preamble, no headings, no markdown.

============================================================
  USER PROMPT (BOTH GAMES)
============================================================
Outcome: FAILURE.

What happened:
{TRIAL}{PAST_BLOCK}

Write the reflection now. At most 5 sentences.

============================================================
  {PAST_BLOCK} (WHEN PRIOR REFLECTIONS EXIST)
============================================================

Prior reflections (do not repeat these):
1. {REFLECTION_1}
2. {REFLECTION_2}

============================================================
  {TRIAL}
============================================================

=== Episode {k} ===
turn {t}: {OBSERVATION} ...

Verdict: episode {k} FAILED after {n} turns.
```

J.1.5 REFLEXION: LESSONS BLOCK

Appended to the actor system prompt from episode 2 on; the list holds one lesson per failed episode so far.

```
============================================================
  INJECTED BLOCK
============================================================
## Lessons from your previous attempts at THIS task You have attempted ...

1. {REFLECTION_1}
2. {REFLECTION_2}

Use these lessons to try a different approach. ...
```

J.1.6 EPISODIC RAG: INJECTED BLOCK AND STORED RECORD

Episodic RAG makes no LLM call besides the actor. It retrieves one success and one failure (the top two by similarity if either class is empty), and each stored record is the verbatim rollout, at most 12 turns per episode with observations cut to 600 characters. The `Best guess placed` line appears

on Mastermind only, the position suffix on `Result` on Maze only, and the `NOTE` line only when every episode repeated the same action sequence. `{outcome label}` is `GOAL`, `wall`, `invalid`, `budget exhausted` or `moved` on Maze, and `SOLVED`, `rejected`, `out of guesses` or the black/white peg counts on Mastermind.

```
============================================================
  INJECTED BLOCK
============================================================
## Retrieved Past Attempts (your own transcripts on other instances of this task)

### Past attempt 1 --- {SUCCESS|FAILURE} (task {task}, instance {instance_id})
{RECORD_CONTENT_1}

### Past attempt 2 --- {SUCCESS|FAILURE} (task {task}, instance {instance_id})
{RECORD_CONTENT_2}

Each block above is a complete past attempt at a different instance ...

============================================================
  STORED RECORD ({RECORD_CONTENT_i})
============================================================
Task: {task}  |  Instance: {instance_id}
Verifier: {solved first at episode N | never solved} (R={success_count}/3) ...

Rollout:
== Episode 1 --- {SOLVED|FAILED} ({n} turns) == ...
```

J.1.7 MEMRL: MEMORY-BUILDING PROMPTS

After each instance the summariser writes a *script* (success, temperature 0.0) or a *reflection* (failure, temperature 0.3). `{INTENT}` is the opening observation of episode 1 without the episode header (with the example guess masked on Mastermind, and only the position clause kept on Maze).

```
============================================================
  SYSTEM PROMPT, SUCCESS (MASTERMIND)
============================================================
You are analyzing a SUCCESSFUL attempt at Mastermind (a hidden 3-digit code, ...

The agent played ONE instance of this task for THREE episodes. ...

The hidden part of the task is different on every instance. ...

After each guess the agent is told how many digits are correct ...

Produce a concise, high-level script capturing the guessing strategy that worked. It
  must be:
1. Generic across codes: about how to choose an opening guess and ...
2. Specific enough to act on: state the inference rule, not a vague intention.
3. At most 5 high-level steps.
4. Explicit about what the agent carried from earlier episodes, if anything.

Do NOT state the secret code or the winning guess.

Output ONLY the numbered script. No preamble, no headers, no markdown fences.

============================================================
  SYSTEM PROMPT, FAILURE (MASTERMIND)
============================================================
You are analyzing a FAILED attempt at Mastermind (a hidden 3-digit code, ...

The agent played ONE instance of this task for THREE episodes. ...

The hidden part of the task is different on every instance. ...

After each guess the agent is told how many digits are correct ...

This attempt failed. Analyze what went wrong and what a future attempt ...
1. Whether any guess was inconsistent with the peg feedback the agent ...
2. Whether episodes 2 and 3 repeated guesses from episode 1 instead ...
3. What to avoid next time, as a rule that applies to any code.

Do NOT state this instance's secret code.

Keep it under 8 sentences. ...

============================================================
```

```
  SYSTEM PROMPT, SUCCESS (MAZE)
============================================================
You are analyzing a SUCCESSFUL attempt at a hidden-maze navigation task.

The agent played ONE instance of this task for THREE episodes. ...

The hidden part of the task is different on every instance. ...

The agent sees only its own coordinates and whether each of the ...

Produce a concise, high-level script capturing the search behaviour that worked. It
  must be:
1. Generic across mazes: about how to explore, backtrack and use what ...
2. Specific enough to act on: name the decision rule and the ...
3. At most 5 high-level steps.
4. Explicit about what changed between episodes, if anything did.

Do NOT state the route, the goal coordinates, or any specific sequence of moves.

Output ONLY the numbered script. No preamble, no headers, no markdown fences.

============================================================
  SYSTEM PROMPT, FAILURE (MAZE)
============================================================
You are analyzing a FAILED attempt at a hidden-maze navigation task.

The agent played ONE instance of this task for THREE episodes. ...

The hidden part of the task is different on every instance. ...

The agent sees only its own coordinates and whether each of the ...

This attempt failed. Analyze what went wrong and what a future attempt ...
1. The search behaviour that wasted turns --- revisiting cells, walking into ...
2. Whether episodes 2 and 3 changed anything after episode 1 failed, ...
3. What to avoid next time, as a rule that applies to any maze.

Do NOT state this maze's route or goal position.

Keep it under 8 sentences. ...

============================================================
  USER PROMPT (BOTH GAMES)
============================================================
Task:
{INTENT}

Trajectory:
{TRAJECTORY_TEXT}
```

J.1.8 MEMRL: INJECTED BLOCK AND STORED RECORDS

The top three memories are injected, labelled with their learned utility Q. `{TRAJECTORY}` uses the Episodic RAG rollout renderer of §J.1.6, from `== Episode 1` down.

```
============================================================
  INJECTED BLOCK
============================================================
## Archived Memories (from your own past experiences on this task)

### Memory 1 --- {SUCCESS|FAILURE} (instance {instance_id}, utility Q={q:+.3f},
  updates={q_visits})
{RECORD_CONTENT_1}

### Memory 2 --- ...
{RECORD_CONTENT_2}

### Memory 3 --- ...
{RECORD_CONTENT_3}

These were retrieved because they have been useful before on instances like this one.
  ...

============================================================
  STORED RECORD (SUCCESS)
============================================================
Task: {task}  |  Instance: {instance_id}
```

```
SCRIPT:
{SCRIPT}

TRAJECTORY:
{TRAJECTORY}

============================================================
  STORED RECORD (FAILURE)
============================================================
TASK REFLECTION:
Task: {task}  |  Instance: {instance_id}

What went wrong:
{REFLECTION}

Failed approach:
{TRAJECTORY}
```

J.1.9 HExA EVOLVER: CONTRASTIVE SKILL DISTILLATION (MASTERMIND)

Round 0, pass 1, on up to five successful and five failed instances.

```
============================================================
  USER PROMPT
============================================================
You are analysing an agent playing Mastermind and distilling what separates its
successes from its failures into reusable skills.

**Task: mastermind**
The agent must find a hidden code of distinct digits. ...

**Instance reward** grades *when* the code was first cracked, not how many
episodes succeeded:
- near +1.00 -- solved on episode 1, in few guesses. The cold-start case.
- +0.60 -- first solved on episode 2. Cross-episode deduction worked.
- +0.30 -- first solved on episode 3. It worked, slowly.
- negative -- never solved. ...
- -1.00 -- never solved, and every episode repeated the same guesses. ...

=== SOLVED ({n_successes} instances) ===
{success_block}

=== NOT SOLVED ({n_failures} instances) ===
{failure_block}

---

Compare them. Each block shows every guess and the pegs it drew, ...

Write the most general skill the comparison supports. ...

**What the agent is already told.** Every agent receives this system prompt ...

    "You are solving the same task across multiple episodes with a fixed ...

and the environment restates the rules of the game -- code length, ...

**The test.** Ask of each candidate skill: ...

Bank only method the prompt does NOT supply: ...

Write every skill as guidance for a DIFFERENT hidden code. ...

Concretely, NEVER write a skill that contains:
- a specific code or digit sequence to try ("start with 1 ...
- a claim about which digits are in the code,
- an opening guess justified by what worked on these instances rather ...

A skill that would stop working if the hidden code changed is ...

OUTPUT JSON OBJECT:
{
  "skills": [
    ...
  ]
}
```

J.1.10 HEXA EVOLVER: MISTAKE EXTRACTION (MASTERMIND)

Round 0, pass 2, on up to eight failed instances. Mastermind has no partial-skill field.

```
============================================================
  USER PROMPT
============================================================
You are analysing failed Mastermind attempts to name the mistakes behind them.

**Task: mastermind**
The agent must find a hidden code of distinct digits. ...

=== FAILED ATTEMPTS ===
{failure_block}

---

Name what these agents did wrong, in terms of the peg feedback ...

**What the agent is already told.** Every agent receives this system prompt ...

    "You are solving the same task across multiple episodes with a fixed ...

and the environment restates the rules of the game -- code length, ...

**The test.** Ask of each candidate skill: ...

Bank only method the prompt does NOT supply: ...

OUTPUT JSON OBJECT:
{
  "mistakes": [
    ...
  ]
}
```

J.1.11 HEXA EVOLVER: BANK EVOLUTION (MASTERMIND)

Rounds 1 onward, once per round, on the round's selected instances (12 on `gpt-oss-120b`, 6 on Maze with Qwen3-8B).

```
============================================================
  USER PROMPT
============================================================
You are curating a skill bank for an agent playing Mastermind.

**Task: mastermind**
The agent must find a hidden code of distinct digits. ...

**Instance reward** grades *when* the code was first cracked, not how many
episodes succeeded:
- near +1.00 -- solved on episode 1, in few guesses. The cold-start case.
- +0.60 -- first solved on episode 2. Cross-episode deduction worked.
- +0.30 -- first solved on episode 3. It worked, slowly.
- negative -- never solved. ...
- -1.00 -- never solved, and every episode repeated the same guesses. ...

=== CURRENT SKILL BANK ===
{existing_skills_block}

=== NEW EVIDENCE: {n_successes} solved, {n_failures} unsolved instances ===
{new_trajectories_block}

---

**First: what is invariant here?** Read the block above for regularities that ...

**Then generalise before you write anything down.** Look at the invariants you just
  listed. ...

Concretely, before adding an entry:
  - If it fires only on one specific feedback value, one count, ...
  - If two or more entries you are considering differ only in ...
  - Prefer an entry that applies at every turn to several that ...

Then revise the bank against this new evidence. Rules:
```

```
1. **Keep** skills the new instances still support, preserving their confidence.
2. **Update** a skill whose wording is right but whose trigger the evidence sharpens.
3. **Add** a skill for a pattern the new instances support and ...
   - **Below the cap, add it.** A free slot is not a ...
   - **At the cap**, an addition must displace something: ...
4. **Remove** skills that are redundant, contradicted by the new instances, tied ...

**What the agent is already told.** Every agent receives this system prompt ...

    "You are solving the same task across multiple episodes with a fixed ...

and the environment restates the rules of the game -- code length, ...

**The test.** Ask of each candidate skill: ...

Bank only method the prompt does NOT supply: ...

Write every skill as guidance for a DIFFERENT hidden code. ...

Concretely, NEVER write a skill that contains:
- a specific code or digit sequence to try ("start with 1 ...
- a claim about which digits are in the code,
- an opening guess justified by what worked on these instances rather ...

A skill that would stop working if the hidden code changed is ...

OUTPUT JSON OBJECT:
{
  "invariants": ["<a regularity that held, with the count supporting it>"],
  "skills": [
    ...
  ],
  "mistakes": [
    ...
  ],
  "removed_skill_titles": ["<title>"],
  "reasoning": "<what was removed, what was added, why>"
}
```

J.1.12 HExA EVOLVER: CONTRASTIVE SKILL DISTILLATION (MAZE)

Round 0, pass 1, on up to five successful and five failed instances.

```
============================================================
  USER PROMPT
============================================================
You are an expert analyst of agent behaviour in interactive environments, distilling ...

**Setting**: an interactive, partially observed navigation task played over repeated
  attempts. Each ...

This means there are two distinct things a trajectory can teach:
- how to explore an unknown maze efficiently on a cold start, and
- how to convert what episode 1 revealed into a better episode ...

The second is only visible because you are shown all three episodes at once. ...

**Reward** is printed in each instance header and grades *when* the maze ...
- near +1.00 -- solved on episode 1, at or near the ...
- +0.70 to +1.00 -- solved on episode 1 with some wasted moves.
- +0.60 -- first solved on episode 2; +0.30 -- first solved ...
- -0.50 -- never solved, but later episodes tried something genuinely different.
- -1.00 -- never solved, and every episode repeated the same move sequence. ...

So instances are **not** equally informative. ...

**Task: maze**
Navigate a grid maze from a fixed start cell to a fixed goal cell. ...

Below are SOLVED and UNSOLVED instances. ...

=== SOLVED INSTANCES ({n_successes}) ===
{success_block}

=== UNSOLVED INSTANCES ({n_failures}) ===
{failure_block}

---
```

```
Your task: by CONTRASTING the solved and unsolved instances, extract the KEY ...

**What the agent is already told.** Every agent in this task receives ...

    "You are solving the same task across multiple episodes with a fixed ...

**The test, which is not a list of banned phrases.** Ask of ...

Applying the test honestly matters more than avoiding particular wording. ...

So a skill about reusing earlier episodes is admissible, but ONLY if ...

Bank only what the prompt does NOT say -- concrete method the ...

Write every skill as guidance for a FRESH, unseen maze: ...

Concretely, NEVER write a skill that contains:
- a sequence of specific moves ("go left, left, then up"),
- specific coordinates ("the goal is near (5, 2)", "avoid (3, 3)"),
- any claim about where the goal is in a particular maze.

A skill that would stop working if the walls were rearranged is ...

For each skill, provide:
- **title**: Short name (3-7 words)
- **principle**: The insight (2-3 sentences). What was done, and why does it work?
- **when_to_apply**: Specific trigger condition (1 sentence)
- **source_seeds**: the instance identifiers this skill was primarily derived from.
  Copy ...

Output a JSON array:
[
  {
    ...
  }
]
```

J.1.13 HEXA EVOLVER: MISTAKE EXTRACTION (MAZE)

Round 0, pass 2, on up to eight failed instances.

```
============================================================
  USER PROMPT
============================================================
You are an expert analyst of agent failures in interactive environments.

**Setting**: an interactive, partially observed navigation task played over repeated
  attempts. Each ...

This means there are two distinct things a trajectory can teach:
- how to explore an unknown maze efficiently on a cold start, and
- how to convert what episode 1 revealed into a better episode ...

The second is only visible because you are shown all three episodes at once. ...

**Reward** is printed in each instance header and grades *when* the maze ...
- near +1.00 -- solved on episode 1, at or near the ...
- +0.70 to +1.00 -- solved on episode 1 with some wasted moves.
- +0.60 -- first solved on episode 2; +0.30 -- first solved ...
- -0.50 -- never solved, but later episodes tried something genuinely different.
- -1.00 -- never solved, and every episode repeated the same move sequence. ...

So instances are **not** equally informative. ...

**Task: maze**
Navigate a grid maze from a fixed start cell to a fixed goal cell. ...

Below are UNSOLVED instances -- three episodes each, none of which reached the goal.

{failure_block}

---

Your task, in two parts.

1. **Mistakes.** Identify the recurring errors that cost these agents the maze. ...
```

```
2. **Partial skills.** Some unsolved instances still contain good behaviour -- a ...

**What the agent is already told.** Every agent in this task receives ...

    "You are solving the same task across multiple episodes with a fixed ...

**The test, which is not a list of banned phrases.** Ask of ...

Applying the test honestly matters more than avoiding particular wording. ...

So a skill about reusing earlier episodes is admissible, but ONLY if ...

Bank only what the prompt does NOT say -- concrete method the ...

Write every skill as guidance for a FRESH, unseen maze: ...

Concretely, NEVER write a skill that contains:
- a sequence of specific moves ("go left, left, then up"),
- specific coordinates ("the goal is near (5, 2)", "avoid (3, 3)"),
- any claim about where the goal is in a particular maze.

A skill that would stop working if the walls were rearranged is ...

Output a JSON object:
{
  "mistakes": [
    ...
  ],
  "partial_skills": [
    ...
  ]
}
```

J.1.14 HEXA EVOLVER: BANK EVOLUTION (MAZE)

Rounds 1 onward, once per round, on the round's selected instances (12 on `gpt-oss-120b`, 6 on Maze with Qwen3-8B).

```
============================================================
  USER PROMPT
============================================================
You are curating a skill bank for an agent solving interactive maze navigation.

**Setting**: an interactive, partially observed navigation task played over repeated
  attempts. Each ...

This means there are two distinct things a trajectory can teach:
- how to explore an unknown maze efficiently on a cold start, and
- how to convert what episode 1 revealed into a better episode ...

The second is only visible because you are shown all three episodes at once. ...

**Reward** is printed in each instance header and grades *when* the maze ...
- near +1.00 -- solved on episode 1, at or near the ...
- +0.70 to +1.00 -- solved on episode 1 with some wasted moves.
- +0.60 -- first solved on episode 2; +0.30 -- first solved ...
- -0.50 -- never solved, but later episodes tried something genuinely different.
- -1.00 -- never solved, and every episode repeated the same move sequence. ...

So instances are **not** equally informative. ...

**Task: maze**
Navigate a grid maze from a fixed start cell to a fixed goal cell. ...

=== CURRENT SKILL BANK ===
{existing_skills_block}

=== NEW EVIDENCE: {n_successes} solved, {n_failures} unsolved instances ===
{new_trajectories_block}

---

**First, state what is invariant.** From the pooled counts above, list regularities ...

Then revise the bank against the new evidence:

1. **Keep** skills the new instances still support.
```

```
2. **Update** a skill whose trigger condition or reasoning the evidence sharpens.
3. **Add** a skill for a supported, transferable pattern no entry covers ...
4. **Remove** skills that are redundant, contradicted, route-specific, or low
  confidence (< ...
5. Keep mistakes unless the new instances show they are no longer common.
6. **Cite a turn**: for every skill, name an instance and the ...

**Bank only what the actor's system prompt does not already say.** Every ...

OUTPUT JSON OBJECT:
{
  "invariants": [
    ...
  ],
  "skills": [
    ...
  ],
  "mistakes": [
    ...
  ],
  "removed_skill_titles": ["<title of removed skill 1>", ...],
  "confidence_changes": [
    ...
  ],
  "reasoning": "<what was removed, what was added, why -- and for any invariant you
  stated but banked no skill from, name it and say why it is unactionable. If a
  quantity inside a retained skill changed, name the instance that justified it>"
}

Be concise but precise. Focus on transferable exploration strategy grounded in the
  trajectories.
```

J.1.15 HExA EVOLVER INPUTS: INSTANCE AND BANK RENDERING

How each instance fills `{success_block}`, `{failure_block}` and `{new_trajectories_block}`, and how the current bank fills `{existing_skills_block}`. The `note:` lines appear only when they apply. On Maze the evolution prompt (not the round-0 passes) prefixes the individual transcripts with a cross-instance aggregate of what the agent observed. The bank has one entry per skill and per mistake, and reads `(No existing skills or mistakes yet)` when empty.

```
============================================================
  INSTANCE RENDER (MASTERMIND)
============================================================
=== Instance {instance_id} | reward {reward:+.2f} ({reward_bin}) | first solved on
  episode ...
  note: every episode repeated the same guesses -- this agent did not ...

============================================================
  INSTANCE RENDER (MAZE)
============================================================
=== Instance {instance_id} | reward {reward:+.2f} ({reward_bin}) | solved first on
  episode ...
  note: {n} turn(s) produced no parseable action and were spent doing nothing ...

Episodes solved: {solved}/{total}
Turns used, all episodes:        {histogram} ...

=== INDIVIDUAL TRANSCRIPTS ===

{per-instance renders as above}

============================================================
  EXISTING BANK
============================================================
EXISTING LEVEL-SPECIFIC SKILLS:
  - [{CONFIDENCE_PCT}%] {TITLE} ...

EXISTING MISTAKES:
  - {MISTAKE_DESCRIPTION}
```

J.1.16 HEXA: INJECTED SKILL BLOCK

Skills are ordered by confidence and capped at 10 skills and 5 mistakes, then truncated to 1,000 tokens; entries distilled from failures are tagged `Lesson` instead of `Strategy`. The Maze block is identical with `(maze)` in the section header.

```
============================================================
  INJECTED BLOCK
============================================================
=== LEARNED PROBLEM-SOLVING STRATEGIES ===

## Task-Specific Strategies (mastermind)
1. [Strategy: {TITLE_1}] {PRINCIPLE_1} ...

2. [Strategy: {TITLE_2}] {PRINCIPLE_2}
   Apply when: {WHEN_TO_APPLY_2} ...

3. [Lesson: {TITLE_3}] {PRINCIPLE_3}
   Apply when: {WHEN_TO_APPLY_3} ...

## Common Mistakes to Avoid
1. **{DESCRIPTION_1}** ...

2. **{DESCRIPTION_2}**
   Why it happens: {WHY_IT_HAPPENS_2} ...

=== END SKILLS ===
```

J.2 REASONING AND PLANNING TASKS

All arms build the actor prompt with the same task factory's prompt builder (`build_solve_prompt`; Appendix E.2), sent as a single user message. ZERO-SHOT passes no guidelines. Reflexion, Episodic RAG, MemRL and HEXA place their block in the `{guidelines}` slot, so each task's prompt is shown once with that slot; the ZERO-SHOT prompt is the same text without the three lines that start at `Before you begin` (for MathGAP, it is `{problem}` alone). Sampling for every actor call is temperature 0.6, top-p 0.95, reasoning effort high and a 60,000-token budget.

J.2.1 ACTOR PROMPT: MATHGAP

```
============================================================
  USER PROMPT (ZERO-SHOT)
============================================================
{problem}

============================================================
  USER PROMPT (WITH MEMORY BLOCK)
============================================================
Before you begin, here are your past reflections on the most helpful ...
{guidelines}

Now here is the task:
{problem}
```

J.2.2 ACTOR PROMPT: ZEBRALOGIC

`{output_schema_json}` is the puzzle's JSON answer schema.

```
============================================================
  USER PROMPT
============================================================
Solve the ZebraLogic puzzle below.

# Example Puzzle
There are 3 houses, numbered 1 to 3 from left to right, ...
- Each person has a unique name: Peter, Eric, Arnold
- Each person has a unique favorite drink: tea, water, milk

## Clues for the Example Puzzle
1. Peter is in the second house.
```

```
2. Arnold is directly left of the one who only drinks water.
3. The one who only drinks water is directly left of the ...

## Answer to the Example Puzzle
{
  "reasoning": "Given Clue 1, Peter is in House 2. According to Clue 2, Arnold is
  directly left of the one who only drinks water, so Arnold must be in House 1 and
  Peter must drink water. Then Clue 3 implies the person in House 3 drinks milk, so
  Eric is in House 3 and Arnold drinks tea.",
  "solution": {
    ...
  }
}

Before you begin, here are your past reflections on the most helpful ...
{guidelines}

# Puzzle to Solve
{problem}

# Instruction
Now please solve the above puzzle. ...
```

J.2.3 ACTOR PROMPT: TOWER OF HANOI

`{problem}` is filled from the template in the second block.

```
============================================================
  USER PROMPT
============================================================
You are a helpful assistant. ...

1. Only one disk can be moved at a time.

2. Each move consists of taking the upper disk from one stack ...

3. A larger disk may not be placed on top of a smaller disk.

The goal is to transform the initial configuration into the goal configuration. ...

moves = [[1, 0, 2], [2, 0, 1], [1, 2, 1], [3, 0, 2],
         [1, 1, 0], [2, 1, 2], [1, 0, 2]]

Requirements:

- When exploring potential solutions in your thinking process, always include the ...

- The positions are 0-indexed (the leftmost peg is 0).

- Ensure your final answer includes the complete list of moves in the format:
  moves = [[disk_id, from_peg, to_peg], ...]

Before you begin, here are your past reflections on the most helpful ...
{guidelines}

{problem}

Ensure your final answer includes the complete list of moves in the format: ...

============================================================
  {problem} TEMPLATE
============================================================
I have a puzzle with {num_disks} disks of different sizes with
Initial configuration:

- Peg 0: {initial_peg_0}

- Peg 1: {initial_peg_1}

- Peg 2: {initial_peg_2}

Goal configuration:

- Peg 0: {goal_peg_0}

- Peg 1: {goal_peg_1}

- Peg 2: {goal_peg_2}
```

```
Rules:

- Only one disk can be moved at a time.

- Only the top disk from any stack can be moved.

- A larger disk may not be placed on top of a smaller disk.

Remember that the peg contents are listed from bottom to top.

Find the sequence of moves to transform the initial configuration into the goal
  configuration.
```

J.2.4 ACTOR PROMPT: MATCHSTICK

```
============================================================
  USER PROMPT
============================================================
You are an expert problem solver. ...

Imagine that each number and operator in the equation that you will ...
- 'I' is composed of one vertical matchstick.
- 'II' is composed of two vertical matchsticks, stacked side by side.
- 'X' is composed of two matchsticks, which are crossed diagonally.
- '=' is composed of two horizontal matchsticks stacked on top of each other.

The equation you will see below will be incorrect. ...

Important constraints: You cannot simply remove a matchstick; you must relocate it ...

Here is an example problem: I = VI - III
The solution is: I = IV - III

Before you begin, here are your past reflections on the most helpful ...
{guidelines}

Now here is the new equation that you have to correct, following ...
{problem}

Please think step by step. ...
```

J.2.5 REFLEXION: REFLECTOR PROMPTS AND LESSONS BLOCK

Three trials per item. After a failed trial the reflector, the only call in any arm that sends a system message, writes one lesson; the list of prior reflections grows by one per failed trial. The lessons block fills `{guidelines}` on the next trial.

```
============================================================
  SYSTEM PROMPT
============================================================
You are analysing a FAILED attempt at a reasoning problem with a ...

You will be shown the problem and the solver's own reasoning-and-answer text. ...

Write a short reflection (at most 5 sentences) that a fresh attempt ...
(a) the specific step where the reasoning went wrong -- an arithmetic ...
(b) what to verify before committing to a final answer next time;
(c) one concrete different approach to try.

Hard rules:
- Do NOT state or imply what the correct answer is; you ...
- Do NOT repeat lessons already listed as prior reflections.
- Output ONLY the reflection text. No preamble, no headings, no markdown.

============================================================
  USER PROMPT (AFTER TRIAL 1)
============================================================
Problem:
{problem_text}

Solver's reasoning and answer:
{raw_text}
```

```
Parsed final answer: {parsed_answer}
Verdict: INCORRECT

Write the reflection now. At most 5 sentences.

============================================================
  USER PROMPT (AFTER TRIAL 2; PRIOR LIST GROWS)
============================================================
Problem:
{problem_text}

Solver's reasoning and answer:
{raw_text}

Parsed final answer: {parsed_answer}
Verdict: INCORRECT

Prior reflections (do not repeat these):
1. {reflection_1}

Write the reflection now. At most 5 sentences.

============================================================
  INJECTED BLOCK (TRIAL 2)
============================================================
## Lessons from your previous attempts at THIS problem
You have attempted this exact problem before and failed. You cannot see
those attempts, but you wrote down what went wrong:

1. {reflection_1}

Use these lessons to try a different approach. ...

============================================================
  INJECTED BLOCK (TRIAL 3)
============================================================
## Lessons from your previous attempts at THIS problem
You have attempted this exact problem before and failed. You cannot see
those attempts, but you wrote down what went wrong:

1. {reflection_1}
2. {reflection_2}

Use these lessons to try a different approach. ...
```

J.2.6 EPISODIC RAG: INJECTED BLOCK

Stored records are the raw rollouts, cut to 3,000 characters (head and tail); no LLM call builds them. One success and one failure are retrieved.

```
============================================================
  INJECTED BLOCK
============================================================
## Retrieved Past Rollouts (from your own past attempts on this level)

### Past rollout 1 --- SUCCESS (level {task_name}, seed {problem_id})
Task: {task_name}  |  Instance: {task_name}:{problem_id} ...

Rollout:
--- Step 1 --- ...

### Past rollout 2 --- FAILURE (level {task_name}, seed {problem_id})
Task: {task_name}  |  Instance: {task_name}:{problem_id} ...

Rollout:
--- Step 1 --- ...

These are complete past attempts with the environment's verdict. ...
```

J.2.7 MEMRL: MEMORY-BUILDING PROMPTS AND INJECTED BLOCK

Each build call is one user message. Up to three memories are injected.

```
============================================================
  USER PROMPT, SUCCESS (SCRIPT)
```

```
============================================================
You are analyzing a SUCCESSFUL solution to a reasoning problem with a ...

You will receive the problem statement and the solver's full reasoning-plus-answer text.

Produce a concise, high-level script that captures the essential solution strategy. ...
1. Generic enough to apply to similar problem instances (different numbers, different
  ...
2. Specific enough to provide useful guidance (name the technique, the key ...
3. At most 5 high-level steps.
4. Focused on strategy and the decisive insight, not a transcription of the arithmetic.

Output ONLY the numbered script. No preamble, no headers, no markdown fences.

Problem:
{problem_text}

Solver's reasoning and answer:
--- Step 1 --- ...

============================================================
  USER PROMPT, FAILURE (REFLECTION)
============================================================
You are analyzing a FAILED attempt at a reasoning problem with a ...

You will receive the problem statement and the solver's full reasoning-plus-answer
  text, ...

This attempt failed. Analyze what went wrong and suggest improvements for future ...
1. The incorrect assumption or arithmetic/logic slip that produced the wrong answer.
2. What to check or verify before committing to a final answer next time.
3. What to avoid repeating.

Keep it under 8 sentences. ...

Problem:
{problem_text}

Solver's reasoning and answer:
--- Step 1 --- ...

============================================================
  INJECTED BLOCK
============================================================
## Archived Memories (from your own past experiences)

### Memory 1 --- SUCCESS (seed {problem_id}, utility Q={q:+.3f}, updates={q_visits})
Task: {task_name}  |  Instance: {task_name}:{problem_id}

SCRIPT:
{script}

TRAJECTORY:
--- Step 1 --- ...

### Memory 2 --- FAILURE (seed {problem_id}, utility Q={q:+.3f}, updates={q_visits})
TASK REFLECTION: ...

What went wrong:
{reflection}

Failed approach:
--- Step 1 --- ...

### Memory 3 --- SUCCESS (seed {problem_id}, utility Q={q:+.3f}, updates={q_visits})
{memory_3_content}

These experiences were retrieved because they are similar to the current scene ...
```

J.2.8 HExA EVOLVER: BANK EVOLUTION

The only evolver prompt called in the reported runs, once per round, as a single user message. `{level_name}` is the task and `{level_description}` its description from §J.2.9.

```
============================================================
  USER PROMPT
============================================================
```

```
You are a teacher curating a strategy bank for a model solving reasoning problems.

**Setting**: a single-turn reasoning benchmark. For each problem the model receives one
  ...

**Reward** is printed in each attempt's header and is **baseline-relative**: ...

- A success on a problem the model rarely solves unaided scores near +1.00. ...
- A success on a problem it almost always solves unaided scores near 0.00. ...
- A failure scores negatively, most negatively when the problem was easy.
- A reward of exactly +1.00 or -1.00 means no baseline was ...

So attempts are **not** equally informative, and a skill drawn from high-reward ...

**Task: {level_name}**
{level_description}

CURRENT SKILL BANK (from previous rounds):
{existing_skills_block}

NEW ATTEMPTS (from the latest round):
Accepted: {n_successes} ...

{new_trajectories_block}

YOUR TASK: evolve the bank using the evidence in the new attempts.

FIRST, before writing any skill, work out the CONTRAST:
- What did the higher-reward attempts do that the lower-reward ones did not?
- Look at the reasoning itself, not just the verdicts: ...
- **If a behaviour appears in both accepted and rejected attempts, it ...

A skill earns its place only if it names something the attempts revealed. ...

RULES:
1. Output the COMPLETE FINAL bank (not a diff) --- include both ...
2. Hard constraints:
   - Maximum 10 total skills for this task.
   - Maximum 5 total mistakes for this task.
3. For each skill you include:
   - If it is a RETAINED skill from the existing bank: ...

Write every skill as guidance for a FRESH, unseen problem of the same task: ...

OUTPUT JSON OBJECT:
{
  "skills": [
    ...
  ],
  "mistakes": [
    ...
  ],
  "removed_skill_titles": ["<title of removed skill 1>", ...],
  "reasoning": "<brief explanation of key changes: what was removed, what was added,
  why>"
}

Be concise but precise. Focus on transferable reasoning strategies grounded in the
  attempts.
```

J.2.9 HEXA EVOLVER INPUTS: TASK DESCRIPTIONS AND RENDERING

The task description filling `{level_description}`, how each attempt fills `{new_trajectories_block}`, and how the bank fills `{existing_skills_block}`.

```
============================================================
  TASK DESCRIPTION (MATHGAP)
============================================================
Each problem is an arithmetic word problem about how many of one ...

============================================================
  TASK DESCRIPTION (ZEBRALOGIC)
============================================================
Each problem is a constraint-satisfaction grid puzzle: ...

============================================================
  TASK DESCRIPTION (TOWER OF HANOI)
```

```
============================================================
Each problem gives a three-peg Tower of Hanoi position: ...

============================================================
  TASK DESCRIPTION (MATCHSTICK)
============================================================
Each problem shows one arithmetic equation written in Roman numerals with the ...

============================================================
  ATTEMPT RENDER (ACCEPTED)
============================================================
--- Problem {problem_id} | Task: {task_name} | Verdict: CORRECT | Reward: {reward:+.2f}
  ---
  Step 1:
    Thought: {raw_text: reasoning+final, clipped to 6000 chars + '... [reasoning
  truncated]', then middle-elided to 2400 chars}
    Action: {parsed_answer}
    Observation: CORRECT --- the verifier accepted the answer

  Final answer: {parsed_answer}

============================================================
  ATTEMPT RENDER (REJECTED, TRUNCATED)
============================================================
--- Problem {problem_id} | Task: {task_name} | Verdict: INCORRECT | Reward:
  {reward:+.2f} | truncated at the token limit ---
  Step 1:
    Thought: {raw_text: reasoning+final, clipped to 6000 chars + '... [reasoning
  truncated]', then middle-elided to 2400 chars}
    Action: {parsed_answer}
    Observation: INCORRECT --- the verifier rejected the answer (generation hit the
  token limit; the answer may be truncated)

  Final answer: {parsed_answer}

============================================================
  EXISTING BANK (ROUND 0)
============================================================
(No existing skills or mistakes yet)

============================================================
  EXISTING BANK (LATER ROUNDS)
============================================================
EXISTING LEVEL-SPECIFIC SKILLS:
  - [{confidence_pct}%] {title}
      principle: {principle}
      apply when: {when_to_apply}
      example: {example}

EXISTING MISTAKES:
  - {description}
```

J.2.10 HExA: INJECTED SKILL BLOCK

Fills `{guidelines}`, capped at 1,000 tokens; the section header names the task.

```
============================================================
  INJECTED BLOCK
============================================================
=== LEARNED PROBLEM-SOLVING STRATEGIES ===

## Task-Specific Strategies (mathgap)
1. [Strategy: {title_1}] {principle_1} ...

2. [Strategy: {title_2}] {principle_2}
   Apply when: {when_to_apply_2} ...

## Common Mistakes to Avoid
1. **{mistake_description_1}** ...

=== END SKILLS ===
```

J.3 DISCOVERPHYSICS

All arms share one actor (Claude Sonnet, at most 10 rounds, at least 2 before a final law, no critic). The arms differ only in a block appended to the end of the system prompt, and ZERO-SHOT is the system prompt alone. The system prompt is identical across worlds except for its world-specific block. We compare ZERO-SHOT, Episodic RAG, MemRL and HEXA on this benchmark.

J.3.1 ACTOR SYSTEM PROMPT (OSCILLATOR, ALL ARMS)

For circle and hubble, the block under `World-Specific Instructions` is replaced by the one in §J.3.2.

```
============================================================
  SYSTEM PROMPT
============================================================
You are an expert physics and AI research scientist tasked with discovering ...

**Workflow:**
1. Analyse the mission description provided.
2. Design a set of experiments to test your hypotheses.
3. Use the `<run_experiment>` tag to submit your experimental inputs.
4. The system will return the results in an `<experiment_output>` tag.
   - If a returned value is `nan`, ignore it --- it indicates ...
5. Only one action is allowed per round: either `<run_experiment>` or `<final_law>`.
...

(Round budget --- total rounds available and minimum rounds before final submission ...

## World-Specific Instructions

The block below describes this world's topology: ...

### Topology: 2-particle

Two particles in a 2D universe.

Your task: discover the law of motion governing particle 2.

### Control Parameters

| Field | Meaning | Typical range |
|---|---|---| ...

(Some worlds in this topology accept additional optional fields --- e.g. ...

### Input Format (interactive mode only)

```
<run_experiment>
...
```

You may submit several experiments per round in the JSON array. ...

Output Format

Each experiment returns:

```
<experiment_output>
...
```

`pos1` and `velocity1` are reported for completeness even though particle 1 is held
  fixed.

`discovered_law` Signature

```python
def discovered_law(pos1, pos2, p1, p2, velocity2, duration, **params):
...
```

The positional arguments must appear in exactly this order. ...

Noise: Reported particle **positions may contain Gaussian observation noise of
  unknown scale** ...
```

```
## Final Submission

When confident, submit your findings as a single Python function enclosed in
  `<final_law>` tags.

**Submission Requirements:**
1. The function must be named `discovered_law`.
2. The positional signature must match the stub given in the world-specific ...
3. Return what the world-specific stub says to return (typically the final ...
4. If you conclude that one of the input arguments does not ...
5. Constants you are CERTAIN about (e.g. a dimensional prefactor you have ...
6. Constants that remain UNCERTAIN --- exponents, screening lengths, diffusion
  coefficients, wave ...

**Implementation note --- time integration:**
The evaluator calls your `discovered_law` many times --- once per (case x ...

Skeleton:

```python
from scipy.integrate import solve_ivp
...
```

If your law genuinely cannot be expressed as an ODE, vectorise with ...

Fittable Parameters (optional, recommended when you have uncertain constants):
Alongside `discovered_law`, you may define a second function `fit_parameters()` that
  returns a ...

```python
def fit_parameters():
...
```

Inside `discovered_law`, read fitted values from `**params` (e.g. ...

Critical Boundaries:
- Do NOT include any explanation or commentary inside the `<final_law>` block ...
- In your final-submission round, output ONLY the `<final_law>` block followed by ...

Explanation Tag (required in the final submission round):
Alongside your `<final_law>`, you MUST also include a separate `<explanation>` tag
  containing ...

Reminder:
The laws of physics in this universe may differ from those in ...

RUN-SPECIFIC CONSTRAINTS (these override any conflicting numbers above)
- For this session you have EXACTLY 10 round(s) of experiments. ...
- You must run at least 2 round(s) of experiments before submitting a <final_law>.
- If your `discovered_law` integrates the trajectory with a for-loop over some ...
- ALWAYS set `dt > 0.005` in EVERY test simulation AND in ...
```

J.3.2 WORLD-SPECIFIC INSTRUCTIONS (CIRCLE, HUBBLE)

```
============================================================
  CIRCLE
============================================================
### Topology: Centre + ring (constrained input)

A 2D universe containing 11 particles:
- **Particle 0**: starts at the centre `[0, 0]`.
- **Particles 1--10**: arranged equally spaced on a ring of radius `ring_radius`, ...

All 11 particles interact through an unknown field. ...

### Control Parameters

| Field | Meaning | Typical range |
|---|---|---| ...

### Input Format (interactive mode only)

```
<run_experiment>
...
```
```

````
```

You may submit several experiments per round in the JSON array. ...

### Output Format

```
<experiment_output>
...
```

Particle ordering is fixed: ...

### `discovered_law` Signature

```python
def discovered_law(positions, velocities, duration, **params):
...
```

The signature takes raw per-particle initial conditions (not `ring_radius` /
  `initial_tangential_velocity`) --- ...

============================================================
  HUBBLE
============================================================
### Topology: Anchor + ring orbiters + 5 probes (with masses)

A 2D universe containing 26 particles:
- **Particle 0**: a single "anchor" near the centre with very large ...
- **Particles 1--20**: 20 background "orbiter" particles arranged on a ring around ...
- **Particles 21--25**: 5 probe particles you fully control --- position, velocity, AND
  mass.

You are scored on how accurately your law predicts the trajectories of ...

### Control Parameters

| Field | Meaning | Typical range |
|---|---|---| ...

Avoid placing probes exactly on the anchor (origin) or on top of ...

### Input Format (interactive mode only)

```
<run_experiment>
...
```

You may submit several experiments per round in the JSON array. ...

### Output Format

```
<experiment_output>
...
```

Particle ordering: index 0 = anchor, indices 1--20 = ring orbiters, indices ...

### `discovered_law` Signature

```python
def discovered_law(positions, velocities, masses, duration):
...
```

Your function must simulate **all 26** particles forward (the orbiters and anchor ...
````

J.3.3 ACTOR USER MESSAGES (ALL ARMS)

The first user message, identical for all three worlds, followed by every message the harness can send on later turns. `{law_stub}` is the world's `discovered_law` signature and `{experiment_format}` a one-line `&lt;run_experiment&gt;` example.

```
============================================================
```
```

```
  USER PROMPT (FIRST TURN)
============================================================
You are an expert physics and AI research scientist tasked with discovering ...

============================================================
  USER MESSAGE: ROUND 9 OF 10 WARNING
============================================================
Warning: this is round {round_num} of {max_rounds}. You have one round remaining. Your
  next response MUST be a <final_law> submission --- you will not be able to run
  further experiments after this.

============================================================
  USER MESSAGE: FINAL ROUND (ROUND 10)
============================================================
This is your final round. You MUST now submit your best guess as a <final_law>
  regardless of confidence. Do not run more experiments.

<final_law>
{law_stub}</final_law>

============================================================
  USER MESSAGE: MISSING EXPLANATION
============================================================
Your <final_law> submission is accepted, but you did not include the required
  <explanation> tag. Reply NOW with ONLY a single <explanation>...</explanation> block
  containing a 2--3 sentence plain-English description of the physical system you
  discovered. Do not include any other text or tags.

============================================================
  USER MESSAGE: FINAL LAW BEFORE ROUND 2
============================================================
You have only run {round_num} round(s) of experiments. You must run at least
  {min_rounds} rounds before submitting a final law. Please design and run at least one
  more experiment.

============================================================
  USER MESSAGE: NO RECOGNISED TAG
============================================================
ERROR: No <run_experiment>, <run_mse_fit>, or <final_law> tag found in your response.
  You must respond with one of these XML tags --- no code fences, no markdown, just the
  raw tag.

Option 1 --- run an experiment:
{experiment_format}

Option 2 --- submit your final law:
<final_law> ...

Option 3 --- fit your candidate law's free parameters against the data ...
<run_mse_fit> ...

Respond with the XML tag NOW. No explanation before or after.

============================================================
  USER MESSAGE: EXPERIMENT RESULT
============================================================
<experiment_output>
{results_json}
</experiment_output>

============================================================
  USER MESSAGE: EXPERIMENT ERROR
============================================================
<experiment_output>
Error running experiment: {error}
</experiment_output>

============================================================
  USER MESSAGE: REPLY TO <run_mse_fit>
============================================================
<mse_fit_output>
{"error":"trajectory CSV logging is disabled, so no data is available for MSE fitting
  in this run.","loss_before":null,"loss_after":null,"fitted_params":{},
  "declared_params":{},"n_training":0}
</mse_fit_output>
```

J.3.4 HEXA: INJECTED SKILL BLOCK

Skills are sorted by confidence (12 skills and 6 mistakes within a world, 8 and 4 for the cross-world hubble bank).

```
============================================================
  INJECTED BLOCK
============================================================
The following experimentation and reasoning skills were learned from discovery episodes
  on ...

=== LEARNED PHYSICS SKILLS ===

## Cross-World Experimentation Skills (learned on other worlds --- verify here)
{i}. [{Strategy|Lesson}: {skill.title}] {skill.principle} ...

... (repeated per skill)
## Common Mistakes to Avoid ...

... (repeated per mistake)
=== END SKILLS ===
```

J.3.5 HEXA EVOLVER: CONTRASTIVE SKILL DISTILLATION

Round 0, pass 1. Sent as a single user message to `claude-sonnet-5`.

```
============================================================
  USER PROMPT
============================================================
You are an expert experimental physicist distilling agent behavior into concise,
transferable discovery skills.

**Environment**: an LLM agent does open-ended physics discovery. Each episode it is ...

**Reward** per trajectory is a blend of explanation quality and normalized trajectory
  ...

**These skills are CROSS-WORLD**: ...

**Current world (setup only --- the governing law is hidden even from you):**
{level_description}

Below are SUCCESSFUL and FAILED discovery episodes on this world. ...

=== SUCCESSFUL EPISODES ({n_successes}) ===
{success_block}

=== FAILED EPISODES ({n_failures}) ===
{failure_block}

---

By CONTRASTING successes and failures, extract the KEY EXPERIMENTATION SKILLS that
  distinguished ...

For each skill provide:
- **title**: short name (3-7 words)
- **principle**: the transferable method (2-3 sentences). What to do and why ...
- **when_to_apply**: the trigger condition (1 sentence)
- **source_seeds**: seed numbers from the episodes above this skill came from

Output a JSON array:

```json
[
...
```

Extract 4-6 DISTINCT skills. Avoid redundancy. No fitted constant values.
```

J.3.6 HEXA EVOLVER: MISTAKE AND PARTIAL-SKILL EXTRACTION

Round 0, pass 2. Sent as a single user message to `claude-sonnet-5`.

```
============================================================
  USER PROMPT
============================================================
You are an expert at analyzing physics-discovery failures and distilling avoidable
mistake patterns.

**Environment**: an LLM agent does open-ended physics discovery. Each episode it is ...

**Reward** per trajectory is a blend of explanation quality and normalized trajectory
  ...

**These skills are CROSS-WORLD**: ...

**Current world (setup only):**
{level_description}

Below are FAILED discovery episodes.

{failure_block}

---

**Part 1 --- Mistakes**: identify COMMON MISTAKE PATTERNS across these failures. For
  each:
1. what the agent did wrong (experiment design, data interpretation, fitting, or ...
2. WHY (the broken belief or shortcut behind it),
3. a concrete, transferable fix.

**Part 2 --- Partial insights**: ...

Output a JSON object:

```json
{
...
```

Extract 3-5 mistakes and 1-2 partial skills. Group similar failures; focus on root
causes. World-agnostic methods only --- no fitted constant values.
```

J.3.7 HExA EVOLVER: BANK EVOLUTION

Rounds 1 onward, once per round. Sent as a single user message to `claude-sonnet-5`.

```
============================================================
  USER PROMPT
============================================================
You are a physics-discovery mentor evolving a CROSS-WORLD skill bank for a discovery
  agent. ...

**Environment**: an LLM agent does open-ended physics discovery. Each episode it is ...

**Reward** per trajectory is a blend of explanation quality and normalized trajectory
  ...

**These skills are CROSS-WORLD**: ...

**World that produced the NEW episodes below (setup only):**
{level_description}

CURRENT SKILL BANK (from previous worlds/rounds):
{existing_skills_block}

EPISODES:
Successes: {n_successes}   Failures: {n_failures}

Episodes are marked by provenance. ...

{new_trajectories_block}

YOUR TASK: merge the existing bank with insights from the new episodes into the
COMPLETE FINAL bank (not a diff).

RULES:
1. Output the full final bank --- retained skills AND new ones.
2. Hard caps: at most {max_skills} skills and {max_mistakes} mistakes total.
3. Per skill: set "is_new" (true = extracted from the new episodes, ...
```

```
4. Retained skills keep their existing confidence unless the new episodes clearly ...
5. New-skill confidence reflects: reward of source episodes, and how broadly the ...
6. REMOVE skills that are: redundant/subsumed, contradicted by the new episodes,
  world-specific ...

OUTPUT JSON OBJECT:
{
  "skills": [
    ...
  ],
  "mistakes": [
    ...
  ],
  "removed_skill_titles": ["..."],
  "reasoning": "<brief: what changed and why, esp. anything demoted/removed>"
}

Be concise and precise. Ground every skill in the episodes.
```

J.3.8 HExA EVOLVER INPUT: WORLD DESCRIPTION (OSCILLATOR)

`{level_description}` for the oscillator evolver prompts: the world's facts and controls, then the first 6,000 characters of the actor's own instructions, so that the evolver does not distil skills the actor is already told.

```
A central two-/N-body configuration. ...

Controls you can set in each experiment (and no others): ...
**WHAT THE AGENT IS ALREADY TOLD.** Every episode the agent receives the ...
```

J.3.9 EPISODIC RAG: INJECTED BLOCK

Numeric literals and law forms in the stored steps are masked, and each rollout keeps its last 12 steps with observations cut to 600 characters.

```
============================================================
  INJECTED BLOCK
============================================================
## Retrieved Past Attempts (your own transcripts from earlier discovery episodes)

### Past attempt {i} --- {SUCCESS|FAILURE} (world {W#}, seed {seed})
Level: {scope_level}  |  Seed: {seed} ...

Rollout:
--- Step {k} --- ...

Each block above is one complete past attempt at discovering a hidden ...
```

J.3.10 MEMRL: MEMORY-BUILDING PROMPTS

Each build call is one prompt, with the role labels written as plain text.

```
============================================================
  PROMPT, SUCCESS (SCRIPT)
============================================================
SYSTEM:
You are analyzing a SUCCESSFUL attempt to discover the hidden law of ...

You will receive the world's setup (what is observable and which controls ...

Produce a concise, high-level script capturing the experimental strategy. It must be:
1. Generic enough to apply to a different world with a different hidden law.
2. Specific about EXPERIMENT DESIGN: which quantity was varied, what was held ...
3. At most 5 high-level steps.
4. Restricted to controls named in the world setup you are given. ...
5. Free of any claim about what the law turned out to ...

Output ONLY the numbered script. No preamble, no headers, no markdown fences.

USER:
World setup: ...
```

```
Trajectory:
{trajectory_text}

ASSISTANT:

============================================================
  PROMPT, FAILURE (REFLECTION)
============================================================
SYSTEM:
You are analyzing a FAILED attempt to discover the hidden law of ...

You will receive the world's setup (what is observable and which controls ...

This attempt failed. Analyze what went wrong at the level of EXPERIMENT ...
1. What the agent failed to measure, or measured in a way ...
2. Concrete design fixes -- which variable to sweep, over what range, ...
3. What to avoid next time.

Constraints:
- Restrict every suggestion to controls named in the world setup you are given. ...
- Do not state or guess what the hidden law is. ...

Keep it under 8 sentences. ...

USER:
World setup: ...

Trajectory:
{trajectory_text}

ASSISTANT:
```

J.3.11 MEMRL: INJECTED BLOCK

```
============================================================
  INJECTED BLOCK
============================================================
## Archived Memories (from your own past discovery episodes)

### Memory {i} --- SUCCESS (world {W#}, seed {seed}, utility Q={q:+.3f},
  updates={q_visits})
Level: {scope_level}  |  Seed: {seed}

SCRIPT:
{script}

TRAJECTORY:
--- Step {k} --- ...

### Memory {i} --- FAILURE (world {W#}, seed {seed}, utility Q={q:+.3f},
  updates={q_visits})
TASK REFLECTION: ...

What went wrong:
{reflection}

Failed approach:
--- Step {k} --- ...

These were retrieved because they have been useful before on tasks like this one. ...
```

K WORKING EXAMPLE: REACT VS. HEXA ON LEVEL CATAPULT OF INTERPHYRE

This appendix walks through a single Interphyre seed — `catapult` seed 64 — on which the REACT baseline fails and HEXA succeeds, with both agents using Claude Sonnet 4.6 as the actor. Seed 64 is a held-out evaluation seed: the bank was frozen after round 17 and was never distilled on this seed. We reproduce, in order: the system prompt and full Thought/Action/Observation transcript for REACT (failure, 25 iterations); the round-17 evolving skill bank that the HEXA agent has access to; the system prompt and full transcript for HEXA (success in 6 iterations, final placement $(x{=}0.3, y{=}-0.25, r{=}2.0)$). The two prompts differ only in the LEARNED PHYSICS SKILLS block injected by the retriever ahead of the user prompt; the tool list, format instructions, and kickoff message are identical, isolating the contribution of the skill bank.

K.1 SPARSIFIED SIMULATION FRAMES (CATAPULT SEED 64)

Figure 21 shows seven sparsely-sampled frames from each agent's final simulation, in left-to-right time order. The top row is REACT: after 25 iterations the agent never produces a placement that brings the green ball into contact with the blue ball. The bottom row is HEXA's 6-iteration solve at $(x{=}0.3,\ y{=}{-}0.25,\ r{=}2.0)$: the large red ball strikes the catapult arm just right of the pivot, the lever launches the green ball in a high arc across the scene, and the green ball comes to rest in the basket against the blue ball.

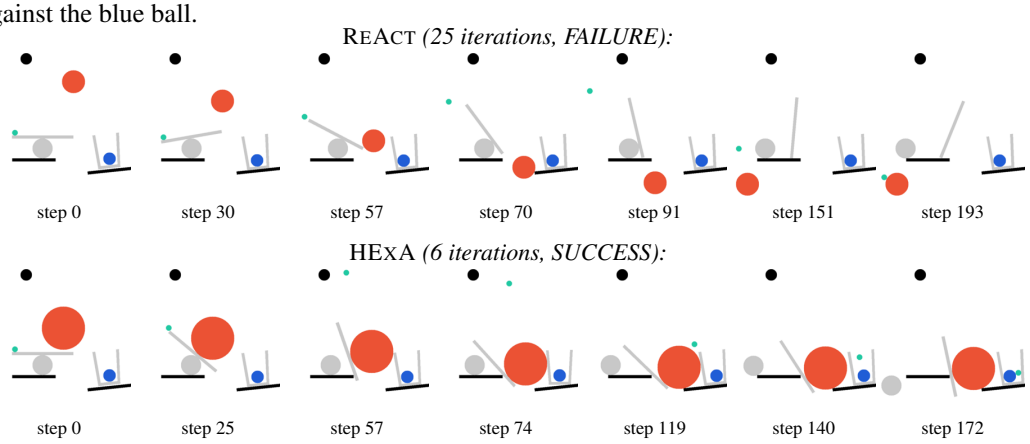


Figure 21: Sparsified simulation frames for catapult seed 64, time increasing left to right. Top row: REACT exhausts its 25-iteration budget without a green–blue contact. Bottom row: HEXA's successful $(x{=}0.3,\ y{=}{-}0.25,\ r{=}2.0)$ placement; the lever launches the green ball into the basket, where it makes the required 3-second contact with the blue ball.

K.2 REACT SYSTEM PROMPT (CATAPULT SEED 64)

The full REACT catapult system prompt is given in Appendix I.1.4 and the tool descriptions in Appendix F; we omit them here and show only the opening lines and the user prompt used for this run.

```
============================================================
  SYSTEM PROMPT
============================================================
You are an expert physics reasoning agent solving a 2D physics puzzle. You have access
  to a physics simulator and can test your ideas before submitting a final answer.

**Puzzle: Catapult**

[... puzzle description, placement constraints, tool list, Thought/Action format and
  rules omitted; see text ...]

============================================================
  USER PROMPT
============================================================
Solve the Catapult puzzle. You have 25 iterations to solve it. The success condition is:
   green_ball must contact blue_ball. Use tools effectively and think about alternate
  approaches if one does not work.
- **ALWAYS call predict_first_contact first** --- it is cheap and tells you if your red
```

```
  ball hits the intended object.
- Only if predict confirms the right contact: call trace_green_ball to track green ball.
```

K.3 REACT TRAJECTORY (CATAPULT SEED 64, 25 ITERATIONS, FAILURE)

```
--- Step 1 ---
Thought:
Let me start by understanding the scene geometry to plan my approach.

Action:
describe_scene_geometry({})

Observation:
=== Scene Geometry ===
World bounds: x in [-5, 5], y in [-5, 5]

Balls:
  green_ball: pos=(-3.06, -1.97), r=0.20, dynamic=True
  blue_ball: pos=(3.50, -1.99), r=0.32, dynamic=True
  black_ball: pos=(-1.17, 4.60), r=0.40, dynamic=False
  gray_ball: pos=(-1.14, -3.07), r=0.70, dynamic=True

Bars:
  ledge: pos=(3.50, -2.71), angle=-3.6 deg, len=3.01, dynamic=False
  black_platform: pos=(-1.74, -3.87), angle=0.0 deg, len=3.00, dynamic=False
  gray_platform: pos=(-1.14, -2.27), angle=0.0 deg, len=4.25, dynamic=True

Baskets:
  basket: pos=(3.50, -2.51), dynamic=True

Key distances:
  green<->blue: 6.56

Success condition: green_ball <-> blue_ball for 3.0s
==================================================

--- Step 2 ---
Thought:
The green ball is at (-3.06, -1.97) sitting on the left end of the gray bar (catapult
  arm). The gray ball pivot is at (-1.14, -3.07). The blue ball is at (3.50, -1.99)
  inside the basket. The catapult mechanism: if I drop something heavy on the right end
  of the gray bar, it will launch the green ball upward and to the right. The gray bar
  is at (-1.14, -2.27) with length 4.25. The right end of the bar would be at
  approximately (-1.14 + 4.25/2, -2.27) = (0.99, -2.27). Let me drop the red ball on the
  right side of the catapult arm to launch the green ball toward the blue ball.

Action:
predict_first_contact({'x': 1.0, 'y': 0.0, 'radius': 0.5})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=1.000, y=0.000, radius=0.500
first_contact_object: gray_platform
contact_step: 31
approach_speed: 2.945
contact_point (approx midpoint): (-0.079, -1.792)
contact_normal (red->other inverted): (0.918, 0.396)

--- Step 3 ---
Thought:
The red ball hits the gray platform (catapult arm) at step 31. Now let me trace the
  green ball to see if it gets launched toward the blue ball.

Action:
trace_green_ball({'x': 1.0, 'y': 0.0, 'radius': 0.5})

Observation:
=== Green Ball Trajectory (steps run: 330) ===
red_ball drop: x=1.000, y=0.000, radius=0.500
Success: False
green_ball: start=(-3.06, -1.97) end=(-4.78, -4.78) Delta=(-1.72, -2.81) peak_y=-0.54 v_max
↪ =7.32

Waypoints (step: x, y):
  step    1: (-3.06, -1.97)
  step   30: (-3.09, -2.26)
  step   60: (-3.24, -0.54)
```

```
  step   90: (-3.24, -0.57)
  step  120: (-3.25, -3.04)
  step  150: (-3.58, -3.76)
  step  180: (-3.96, -4.78)
  step  210: (-4.33, -4.78)
  step  240: (-4.70, -4.78)
  step  270: (-4.78, -4.78)
  step  300: (-4.78, -4.78)
  step  330: (-4.78, -4.78)

--- Step 4 ---
Thought:
The green ball went left and down, not toward the blue ball. I need to hit the right end
  of the catapult arm more precisely to launch the green ball to the right. The gray bar
  extends from about (-1.14 - 4.25/2) = (-3.26) to (-1.14 + 4.25/2) = (0.99) in x. I
  need to hit further right on the bar. Let me try dropping further to the right end.

Action:
predict_first_contact({'x': 2.0, 'y': -1.0, 'radius': 0.5})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=2.000, y=-1.000, radius=0.500
first_contact_object: ledge
contact_step: 27
approach_speed: 0.883
contact_point (approx midpoint): (2.750, -2.362)
contact_normal (red->other inverted): (-0.906, 0.423)

--- Step 5 ---
Thought:
The red ball hits the ledge, not the catapult arm. I need to hit the right end of the
  gray bar more precisely. The bar extends to about x=0.99 on the right. Let me try
  dropping closer to the right end of the bar.

Action:
predict_first_contact({'x': 0.5, 'y': -1.0, 'radius': 0.5})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=0.500, y=-1.000, radius=0.500
first_contact_object: gray_platform
contact_step: 20
approach_speed: 1.569
contact_point (approx midpoint): (-0.322, -1.905)
contact_normal (red->other inverted): (0.916, 0.401)

--- Step 6 ---
Thought:
The red ball hits the gray platform (catapult arm). Let me trace the green ball with
  this placement.

Action:
trace_green_ball({'x': 0.5, 'y': -1.0, 'radius': 0.5})

Observation:
=== Green Ball Trajectory (steps run: 300) ===
red_ball drop: x=0.500, y=-1.000, radius=0.500
Success: False
green_ball: start=(-3.06, -1.97) end=(-4.66, -4.78) Delta=(-1.60, -2.81) peak_y=-0.21 v_max
↪ =4.62

Waypoints (step: x, y):
  step    1: (-3.06, -1.97)
  step   30: (-3.09, -1.70)
  step   60: (-2.65, -0.21)
  step   90: (-2.11, -0.92)
  step  120: (-2.38, -3.03)
  step  150: (-3.86, -3.88)
  step  180: (-4.67, -4.67)
  step  210: (-4.62, -4.78)
  step  240: (-4.63, -4.78)
  step  270: (-4.65, -4.78)
  step  300: (-4.66, -4.78)

--- Step 7 ---
Thought:
The green ball is going up but then falling to the left. I need it to go right toward
  the blue ball at (3.50, -1.99). The catapult needs to be hit on the far right end so
```

```
  the green ball launches to the right. Let me try hitting closer to x=0.9 which should
  be near the right end of the bar.

Action:
predict_first_contact({'x': 0.8, 'y': 0.5, 'radius': 0.3})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=0.800, y=0.500, radius=0.300
first_contact_object: gray_platform
contact_step: 37
approach_speed: 1.724
contact_point (approx midpoint): (-0.189, -1.812)
contact_normal (red->other inverted): (0.914, 0.406)

--- Step 8 ---
Thought:
The red ball is hitting the bar but not at the right end. The contact point is at
  x=-0.189, which means the ball is landing near the center-left of the bar. I need to
  position the ball so it hits the far right end of the bar (near x=0.99). Let me try a
  smaller radius and position more to the right.

Action:
trace_green_ball({'x': 0.8, 'y': 0.5, 'radius': 0.3})

Observation:
=== Green Ball Trajectory (steps run: 240) ===
red_ball drop: x=0.800, y=0.500, radius=0.300
Success: False
green_ball: start=(-3.06, -1.97) end=(-4.71, -4.78) Delta=(-1.65, -2.81) peak_y=-1.58 v_max
↪ =8.08

Waypoints (step: x, y):
  step    1: (-3.06, -1.97)
  step   30: (-3.09, -2.26)
  step   60: (-3.47, -1.58)
  step   90: (-3.91, -1.96)
  step  120: (-4.36, -4.79)
  step  150: (-4.77, -4.78)
  step  180: (-4.75, -4.78)
  step  210: (-4.73, -4.78)
  step  240: (-4.71, -4.78)

--- Step 9 ---
Thought:
The green ball keeps going left. The catapult lever needs to be hit on the right side of
  the pivot to launch the green ball (on left end) upward and to the right. The pivot is
  at x=-1.14, so the right end of the bar is around x=0.99. The ball hitting the bar
  seems to be tilting it the wrong way. Let me try hitting very close to the right tip
  of the bar.

Action:
predict_first_contact({'x': 0.9, 'y': -1.5, 'radius': 0.2})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=0.900, y=-1.500, radius=0.200
first_contact_object: gray_platform
contact_step: 16
approach_speed: 0.270
contact_point (approx midpoint): (-0.121, -2.054)
contact_normal (red->other inverted): (0.980, 0.198)

--- Step 10 ---
Thought:
The contact point is still at x=-0.121, not at the right end. The red ball seems to be
  rolling/sliding along the bar to where contact is detected. Let me try placing the red
  ball directly above the far right end of the bar at x=0.99.

Action:
trace_green_ball({'x': 0.9, 'y': -1.5, 'radius': 0.2})

Observation:
=== Green Ball Trajectory (steps run: 360) ===
red_ball drop: x=0.900, y=-1.500, radius=0.200
Success: False
green_ball: start=(-3.06, -1.97) end=(-4.78, -4.78) Delta=(-1.72, -2.81) peak_y=-1.90 v_max
↪ =2.74
```

```
Waypoints (step: x, y):
  step    1: (-3.06, -1.97)
  step   30: (-3.09, -2.02)
  step   60: (-3.17, -1.93)
  step   90: (-3.24, -1.90)
  step  120: (-3.27, -2.01)
  step  150: (-3.36, -2.22)
  step  180: (-3.70, -2.72)
  step  210: (-4.16, -4.71)
  step  240: (-4.63, -4.78)
  step  270: (-4.78, -4.78)
  step  300: (-4.78, -4.78)
  step  330: (-4.78, -4.78)
  step  360: (-4.78, -4.78)

--- Step 11 ---
Thought:
The green ball keeps going left after launch. Let me try a large, heavy drop from high
  up on the far right end of the catapult arm to create more rightward force. I'll use a
  bigger radius ball dropped from high above the right end of the bar.

Action:
predict_first_contact({'x': 0.9, 'y': 3.0, 'radius': 1.0})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=0.900, y=3.000, radius=1.000
first_contact_object: gray_platform
contact_step: 50
approach_speed: 6.945
contact_point (approx midpoint): (-0.176, -1.352)
contact_normal (red->other inverted): (0.769, 0.639)

--- Step 12 ---
Thought:
The red ball hits the gray platform with high speed (6.945). Let me trace the green ball
  to see if this launches it toward the blue ball.

Action:
trace_green_ball({'x': 0.9, 'y': 3.0, 'radius': 1.0})

Observation:
=== Green Ball Trajectory (steps run: 420) ===
red_ball drop: x=0.900, y=3.000, radius=1.000
Success: False
green_ball: start=(-3.06, -1.97) end=(-4.78, -4.78) Delta=(-1.71, -2.81) peak_y=+0.89 v_max
↪ =8.35

Waypoints (step: x, y):
  step    1: (-3.06, -1.97)
  step   30: (-3.09, -2.26)
  step   60: (-3.83, -1.46)
  step   90: (-4.63, +0.89)
  step  120: (-4.30, +0.52)
  step  150: (-3.96, -2.31)
  step  180: (-4.45, -3.26)
  step  210: (-4.78, -3.54)
  step  240: (-4.78, -3.61)
  step  270: (-4.78, -3.75)
  step  300: (-4.78, -4.16)
  step  330: (-4.78, -4.78)
  step  360: (-4.78, -4.78)
  step  390: (-4.78, -4.78)
  step  420: (-4.78, -4.78)

--- Step 13 ---
Thought:
The green ball keeps going left instead of right. Let me get the full level state to
  better understand the bar geometry and try a different approach.

Action:
get_level_state({})

Observation:
=== Level State: catapult ===

* green_ball (green Ball): position=(-3.0633, -1.9686), radius=0.2000, dynamic=True
* blue_ball (blue Ball): position=(3.5000, -1.9920), radius=0.3200, dynamic=True
* ledge (black Bar): position=(3.5000, -2.7124), left=2.0000, right=5.0000, top=-2.6172,
↪ bottom=-2.8075, thickness=0.2000, dynamic=False
```

```
* basket: {'x': 3.5, 'y': np.float64(-2.5119720543860606), 'color': 'gray', 'dynamic': True,
↪  'type': 'Basket', 'angle': -3.6288244368770233, 'scale': 0.7880514159250508}
* black_ball (black Ball): position=(-1.1695, 4.6000), radius=0.4000, dynamic=False
* black_platform (black Bar): position=(-1.7383, -3.8686), left=-3.2383, right=-0.2383, top
↪ =-3.8686, bottom=-3.8686, thickness=0.2000, dynamic=False
* gray_ball (gray Ball): position=(-1.1383, -3.0686), radius=0.7000, dynamic=True
* gray_platform (gray Bar): position=(-1.1383, -2.2686), left=-3.2633, right=0.9867, top
↪ =-2.2686, bottom=-2.2686, thickness=0.2000, dynamic=True

World bounds: x in [-5, 5], y in [-5, 5]
Action objects: ['red_ball'] -- you control the red ball placement
Success condition: green_ball must touch blue_ball for 3 seconds
============================================================

--- Step 14 ---
Thought:
The green ball keeps launching left because the catapult rotation sends the left end
  upward-left tangentially. I need a different approach - let me try placing the red
  ball to the LEFT of the green ball to directly push it rightward off the arm.

Action:
predict_first_contact({'x': -3.6, 'y': 0.0, 'radius': 0.5})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=-3.600, y=0.000, radius=0.500
first_contact_object: green_ball
contact_step: 38
approach_speed: 4.580
contact_point (approx midpoint): (-3.359, -2.227)
contact_normal (red->other inverted): (-0.733, 0.680)

--- Step 15 ---
Thought:
The red ball hits the green ball! Let me trace the green ball to see if this pushes it
  toward the blue ball.

Action:
trace_green_ball({'x': -3.6, 'y': 0.0, 'radius': 0.5})

Observation:
=== Green Ball Trajectory (steps run: 240) ===
red_ball drop: x=-3.600, y=0.000, radius=0.500
Success: False
green_ball: start=(-3.06, -1.97) end=(-3.80, -4.78) Delta=(-0.74, -2.81) peak_y=-1.97 v_max
↪ =3.94

Waypoints (step: x, y):
  step    1: (-3.06, -1.97)
  step   30: (-3.09, -2.26)
  step   60: (-3.02, -3.52)
  step   90: (-3.27, -3.56)
  step  120: (-3.63, -4.39)
  step  150: (-3.80, -4.78)
  step  180: (-3.80, -4.78)
  step  210: (-3.80, -4.78)
  step  240: (-3.80, -4.78)

--- Step 16 ---
Thought:
The green ball goes down instead of right. Let me try dropping from directly above the
  green ball to push it rightward, or try hitting it from the left side more precisely.
  Let me try placing the red ball to the left of the green ball at the same height to
  push it horizontally.

Action:
predict_first_contact({'x': -4.0, 'y': -1.97, 'radius': 0.5})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=-4.000, y=-1.970, radius=0.500
first_contact_object: bottom_wall
contact_step: 43
approach_speed: 1.405
contact_point (approx midpoint): (-2.000, -4.732)
contact_normal (red->other inverted): (-0.991, 0.130)

--- Step 17 ---
Thought:
```

```
The red ball falls through to the bottom wall without hitting the green ball. Let me try
  a different approach - place the red ball just to the left of the green ball at the
  same height so it rolls into it.

Action:
predict_first_contact({'x': -4.5, 'y': -1.97, 'radius': 0.3})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=-4.500, y=-1.970, radius=0.300
first_contact_object: bottom_wall
contact_step: 45
approach_speed: 1.470
contact_point (approx midpoint): (-2.250, -4.828)
contact_normal (red->other inverted): (-0.997, 0.074)

--- Step 18 ---
Thought:
The red ball isn't hitting the green ball when placed at the same height. Let me try a
  different approach - place the red ball directly above the blue ball/basket area to
  push the blue ball down, or better, try to have the green ball land in the basket. Let
  me try placing the red ball on the right side of the catapult arm near the pivot to
  tilt the arm differently.

Action:
predict_first_contact({'x': -0.5, 'y': 2.0, 'radius': 0.8})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=-0.500, y=2.000, radius=0.800
first_contact_object: gray_platform
contact_step: 47
approach_speed: 5.033
contact_point (approx midpoint): (-0.860, -1.645)
contact_normal (red->other inverted): (0.510, 0.860)

--- Step 19 ---
Thought:
The red ball hits the gray platform near the center. Let me trace the green ball
  trajectory.

Action:
trace_green_ball({'x': -0.5, 'y': 2.0, 'radius': 0.8})

Observation:
=== Green Ball Trajectory (steps run: 300) ===
red_ball drop: x=-0.500, y=2.000, radius=0.800
Success: False
green_ball: start=(-3.06, -1.97) end=(-3.44, -3.94) Delta=(-0.38, -1.97) peak_y=+2.86 v_max
↪ =9.56

Waypoints (step: x, y):
  step    1: (-3.06, -1.97)
  step   30: (-3.09, -2.26)
  step   60: (-3.84, -0.61)
  step   90: (-4.68, +2.50)
  step  120: (-4.38, +2.86)
  step  150: (-4.09, +0.77)
  step  180: (-3.74, -3.53)
  step  210: (-3.44, -3.83)
  step  240: (-3.44, -3.92)
  step  270: (-3.44, -3.94)
  step  300: (-3.44, -3.94)

--- Step 20 ---
Thought:
The green ball keeps going left. The catapult mechanism launches it to the left due to
  the bar angle. Let me try a completely different approach - place the red ball to
  knock the blue ball out of the basket toward the green ball, or drop it to push the
  blue ball left toward the green ball.

Action:
predict_first_contact({'x': 3.5, 'y': 2.0, 'radius': 0.5})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=3.500, y=2.000, radius=0.500
first_contact_object: basket
```

```
contact_step: 43
approach_speed: 4.784
contact_point (approx midpoint): (3.504, -1.580)
contact_normal (red->other inverted): (0.003, 1.000)

--- Step 21 ---
Thought:
The red ball hits the basket. Let me simulate this to see if the blue ball gets knocked
  toward the green ball.

Action:
simulate_action({'x': 3.5, 'y': 2.0, 'radius': 0.5})

Observation:
FAILURE. The green ball did NOT contact the blue ball in the basket.
Total simulation steps: 2000
Reward: -0.1

Final object positions:
  green_ball: pos=(-4.777, -4.780), vel=(0.000, 0.000)
  red_ball: pos=(3.552, -1.247), vel=(0.000, 0.000)
  blue_ball: pos=(3.621, -2.103), vel=(0.000, 0.000)
  ledge: pos=(3.500, -2.712), vel=(0.000, 0.000)
  basket: pos=(3.470, -2.593), vel=(0.000, 0.000)
  black_ball: pos=(-1.170, 4.600), vel=(0.000, 0.000)
  black_platform: pos=(-1.738, -3.869), vel=(0.000, 0.000)
  gray_ball: pos=(-2.160, -3.064), vel=(0.000, 0.000)
  gray_platform: pos=(-3.222, -3.040), vel=(0.000, 0.000)

--- Step 22 ---
Thought:
The green ball keeps launching left. Let me try a completely different approach - drop
  the red ball to push the blue ball out of the basket to the left, so the blue ball
  travels toward the green ball's final position at the bottom left.

Action:
predict_first_contact({'x': 4.5, 'y': 0.0, 'radius': 0.5})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=4.500, y=0.000, radius=0.500
first_contact_object: right_wall
contact_step: 1
approach_speed: 0.163
contact_point (approx midpoint): (4.738, -0.001)
contact_normal (red->other inverted): (-1.000, -0.005)

--- Step 23 ---
Thought:
The green ball keeps launching to the upper-left. Let me try hitting the blue
  ball/basket from the right side to knock the blue ball leftward toward where the green
  ball ends up (around -4.78, -4.78). I'll drop the red ball to the right of the basket.

Action:
predict_first_contact({'x': 4.8, 'y': -1.5, 'radius': 0.2})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=4.800, y=-1.500, radius=0.200
first_contact_object: right_wall
contact_step: 1
approach_speed: 0.163
contact_point (approx midpoint): (4.888, -0.751)
contact_normal (red->other inverted): (-0.141, -0.990)

--- Step 24 ---
Thought:
Let me try to knock the blue ball out of the basket by dropping from above and to the
  right of it, so it moves left toward where the green ball will be.

Action:
predict_first_contact({'x': 3.8, 'y': -1.5, 'radius': 0.3})

Observation:
INVALID placement:
  - Overlaps with blue_ball: dist=0.5762, sum_of_radii=0.6200. Move at least 0.0938 units
↪ further away.
placement_valid=False; no contact check performed.
```

```
--- Step 25 ---
Thought:
Let me try to drop the red ball to hit the blue ball from above to knock it out of the
  basket toward the left.

Action:
predict_first_contact({'x': 3.8, 'y': -1.0, 'radius': 0.3})

Observation:
INVALID ACTION: Action object 0 at (3.80, -1.00) with radius 0.30 is invalid
```

K.4 ROUND-17 EVOLVING SKILL BANK USED BY HEXA

```
# CATAPULT SKILLS

[cat_ev_9_000]
  Title:
    Right-of-Pivot Arm Strike Zone (x~=0.2--0.5 Default)
  generation: 11    confidence: 0.93
  source seeds: [43, 48, 49, 51, 52, 53]
  Principle:
    Placing the red ball at x~=0.2--0.5 on the right half of the catapult arm creates a
      downward force that lifts the green ball in a rightward arc toward the basket.
      x=0.3 is the default starting point; x=0.2 at standard y reduces range ~1.3 units.
      Shifting x rightward (>=0.5) at standard-y positions risks diminishing returns.
  When to apply:
    Default for all catapult strikes; start with x=0.3. At moderate elevated y (~=0.5),
      decreasing x below 0.3 increases range (see leftward-x skill). Do not exceed x~=0.5
      without checking for diminishing returns.
  Example:
    Seed 43: x=0.3 y=-0.5 -> green at x=3.10, success. Seed 48: x=0.3 y=0.3 -> success.
      Seed 51: x=0.3 y=0.0 -> success.

[cat_ev_6_001]
  Title:
    Low Drop (y~=-0.5 to 0.0) Maximizes Range; Binary-Search y First
  generation: 11    confidence: 0.93
  source seeds: [48, 49, 51, 53]
  Principle:
    Drop height y is the primary range control: y~=-0.5 produces maximum range, y~=0.0
      produces moderate range, y~=0.3--0.5 produces reduced range. A +/-0.25 change in y at
      standard heights shifts landing x by 1.5--2.0 units. When standard y (<=0.0) is
      blocked, the operative range shifts to y=0.25--0.5, where lower y still produces
      more range.
  When to apply:
    Default for all seeds; start at y=-0.5, then binary-search toward y=0.0 if
      overshooting. If gray_ball blocks all y<=0.0, shift to y=0.5 as first valid
      placement, then binary-search DOWN to y=0.3 or y=0.25 to increase range. Use
      y=-0.25 as first midpoint in standard regime.
  Example:
    Seed 49: y=-0.5 blocked, y=0.0->x=4.26 (overshoot), y=0.25->x=3.71, success. Seed 53:
      y=0.0->x=4.45, y=0.25->x=4.19, y=0.5->x=4.17 (y-dominant regime detected).

[cat_ev_11_002]
  Title:
    Overlap Fallback Sequence: y=-0.5 -> y=0.0 -> y=0.5 -> Binary-Search y=0.3--0.5
  generation: 11    confidence: 0.91
  source seeds: [48, 50, 51, 52, 53]
  Principle:
    When y=-0.5 is blocked, try y=0.0, then y=0.5 with r=2.0. Once a valid placement at
      y=0.5 is confirmed, binary-search DOWNWARD within y=0.3--0.5 to fine-tune range,
      since lower y in this zone still increases range. Only escalate to y=1.0 if y=0.3
      is also blocked or all y=0.3--0.5 attempts miss by more than 1 unit.
  When to apply:
    When predict_first_contact returns invalid for y=-0.5; follow sequence y=0.0 -> y=0.5
      r=2.0 -> binary-search y=0.3--0.5. Skip x=1.0 at y=1.0 always. For elevated green
      ball scenes (y>=-0.5), standard y slots may all be blocked and sequence starts at
      y=1.0.
  Example:
    Seed 48: y=-0.5/0.0 blocked -> y=0.5 (x=2.37 undershoot) -> y=0.3 (x=3.14, success).
      Seed 51: y=-0.5 blocked -> y=0.0 valid (gap 0.21, success). Seed 52: y=-0.5/0.0
      blocked -> y=0.5 undershoot -> bisect down.

[cat_ev_11_003]
  Title:
    Trace-then-Simulate: Gap Threshold, Bisect Chaotic Cliffs, Simulate Narrow Boundary
  generation: 11    confidence: 0.87
  source seeds: [49, 50, 51, 52]
```

```
  Principle:
    At standard drop heights (y<=0.0), commit to simulate when gap is under 0.8 units. At
      elevated heights (y=0.25--0.5), commit at gap <0.5. When a chaotic cliff is
      detected in y=0.3--0.5, bisect to the midpoint rather than simulating immediately.
      After bisection has narrowed the cliff to Deltay<=0.01 (minimum resolution),
↪ simulating
      the overshoot boundary IS valid---the green ball can contact the blue ball during
      its arc even if the final resting position overshoots.
  When to apply:
    At y<=0.0: gap <0.8 -> simulate. At y=0.25--0.5: gap <0.5 -> simulate. If chaotic cliff:
      bisect first. Once Deltay<=0.01 and no gap reduction is possible, simulate the
      overshoot side---arc contact is achievable at that boundary.
  Example:
    Seed 51: gap ~0.21 at y=0.0 -> success. Seed 52: cliff at y=0.34/0.35 after full
      bisection -> simulated y=0.34 (overshoot) -> success via arc contact.

[cat_ev_1_005]
  Title:
    Seed-Dependent Scene Geometry Requires Per-Seed Calibration
  generation: 11    confidence: 0.85
  source seeds: [42, 43, 51, 52, 53]
  Principle:
    Arm height, pivot position, blue ball position, green ball starting position, and
      arm tilt vary meaningfully across seeds. If green ball starts at y>=-0.5
      (elevated), standard low-y drops will be blocked, requiring elevated-y placements
      starting at y~=1.0--1.1. If blue ball is at y<-3.0, standard r=2.0 placements may
      produce too flat a trajectory.
  When to apply:
    Always call describe_scene_geometry first; if green ball y>=-0.5, treat as
      elevated-arm seed requiring y>=1.0 placements. If first 2--3 standard placements
      show green going left, treat as non-standard geometry seed requiring extra
      calibration.
  Example:
    Seed 43: standard geometry -> x=0.3 y=-0.5 -> success in 4 steps. Seed 42: green at
      y=-0.38 (elevated), standard y blocked, needed y=1.0 x-sweep. Seed 53: blue ball
      at y=-3.02, needed x=0.1 at y=0.5.

[cat_ev_9_006]
  Title:
    Chaotic Trajectory Zones: Avoid Fine-Tuning in Sensitive Ranges
  generation: 11    confidence: 0.91
  source seeds: [44, 45, 50, 52]
  Principle:
    Chaotic zones exist in both y and x dimensions where tiny parameter changes cause
      large trajectory swings: (1) y~=-0.15 to -0.25 at x=0.3; (2) y~=0.3--0.4 at r=2.0 in
      some seeds (Deltay=0.1 -> >1.9 range jump); (3) y~=1.05--1.1 at elevated arms; (4)
      x~=0.35--0.40 at y=0.75. When a chaotic cliff is detected, bisect methodically; the
      cliff boundary can be as narrow as Deltay=0.01.
  When to apply:
    When two consecutive traces with small Delta in x or y produce landing positions >1.5
      units apart: for y-chaotic cliffs bisect (e.g., try y=0.35 between y=0.3 and
      y=0.4); for x-chaotic cliffs simulate overshoot side. Do not iterate further
      within the chaotic range without a clear bisection strategy.
  Example:
    Seed 44: y=0.5->x=4.50, y=0.7->x=-0.33 (y-cliff). Seed 52: y=0.3->x=4.23, y=0.4->x=2.71
      (cliff); bisected to y=0.34/y=0.35 boundary; simulated y=0.34 -> success.

[cat_ev_6_009]
  Title:
    Identical Range at Different x/r Signals y-Dominant Regime
  generation: 11    confidence: 0.93
  source seeds: [44, 47, 53]
  Principle:
    When trace_green_ball returns nearly identical green ball landing positions despite
      changing x or radius, the scene is in a y-dominant regime where drop height
      controls range. Stop iterating x or r and switch to y-binary-search or x-sweep at
      a different y level. If lower y is blocked, try x<0.3 at moderate elevated y
      (~=0.5) to escape the regime.
  When to apply:
    When two consecutive traces with different x or r values produce landing positions
      within 0.2 units of each other; immediately switch to adjusting y. If y is blocked
      and regime persists at x=0.3, try x=0.1--0.2 at y=0.5 to break out.
  Example:
    Seed 44: x=0.3 and x=0.2 at r=2.0 both ->x=4.50. Seed 47: x=0.3 y=0.5 and various r
      all gave x~=2.69. Seed 53: y=0.0->x=4.45, y=0.25->x=4.19, y=0.5->x=4.17 (plateau) ->
      switched to x=0.1 -> success.

[cat_ev_10_007]
  Title:
    x=0.2 Escape for Standard-y Chaotic Cliff (y<=0.0 Only)
  generation: 11    confidence: 0.85
```

```
  source seeds: [41]
  Principle:
    When y binary-search at standard heights (y~=-0.5 to 0.0) reveals that y=-0.5 and
      y=-0.25 produce identical overshooting landings but y=0.0 flips the green ball far
      left, shifting x from 0.3 to 0.2 at y=-0.5 reduces range by ~1.3 units. This
      escape is strictly for standard y (<=0.0): at elevated y (~=0.5), x=0.2 increases
      range (opposite effect).
  When to apply:
    Only when standard-y (y~=-0.5 to 0.0) binary search hits chaotic cliff with both
      y=-0.5 and y=-0.25 overshooting identically but y=0.0 going far left. Do NOT apply
      at elevated y (>=0.5)---use leftward x-sweep there for the opposite
      (range-increasing) effect.
  Example:
    Seed 41: x=0.3 y=-0.5->x=4.33, y=-0.25->x=4.33 (identical), y=0.0->x=-4.78; x=0.2
      y=-0.5->x=3.05, success.

[cat_ev_9_009]
  Title:
    Elevated-y Fallback: x=2.0, r=1.5 at y=1.0--1.5 as Last Resort
  generation: 11    confidence: 0.8
  source seeds: [36, 40]
  Principle:
    When the overlap fallback sequence reaches y>=1.0 and x-sweep at y=1.0 (0.3->0.5)
      still fails, escalate to x=2.0, r=1.5---not x=1.0 at y=1.0 (INVALID ACTION). At
      x=2.0, y=1.0, r=1.5, the red ball contacts the arm at a rightward strike point,
      sending green ball with full range. An alternative for elevated scenes is x=0.3,
      y=1.1, r=2.0 when x=0.3 y=1.0 is blocked by gray_ball.
  When to apply:
    After exhausting x-sweep (0.3--0.5) at y=1.0 in elevated-arm seeds; use x=2.0 r=1.5
      at y=1.0, then y=1.5 if still blocked. Skip x=1.0 at y=1.0 always.
  Example:
    Seed 36: x=2.0, y=1.0, r=1.5 -> success. Seed 40: x=0.3, y=1.1, r=2.0 -> green at
      x=4.32, success.

[cat_ev_11_009]
  Title:
    Leftward x Shift (<0.3) at Moderate Elevated y (~=0.5) Dramatically Increases Range
  generation: 11    confidence: 0.72
  source seeds: [47, 53]
  Principle:
    At moderate elevated drop heights (y~=0.5), decreasing x below 0.3 dramatically
      increases green ball range---opposite to standard-y behavior. This occurs because at
      y=0.5 the arm contact geometry changes such that a more central strike delivers
      more angular momentum to the arm. When stuck in a y-dominant regime with
      insufficient or excessive range at x=0.3, try x=0.1--0.2 at y=0.5 before escalating
      to y=1.0.
  When to apply:
    When y=0.5 with x=0.3 produces insufficient range AND lower y is blocked or chaotic;
      try x=0.2 (binary mid) then x=0.1 at y=0.5. Binary-search x between 0.1 and 0.3 to
      target blue ball position. Do NOT apply at y>=1.0.
  Example:
    Seed 47: x=0.3 y=0.5->x=2.72, x=0.1 y=0.5->x=4.78; x=0.2 y=0.5 simulated -> success.
      Seed 53: x=0.3 y=0.5->x=4.17 (regime), x=0.1 y=0.5->x=3.63 (gap 0.13) -> success.

# CATAPULT COMMON MISTAKES

[cat_err_000]
  Description:
    Fine-tuning x or y in a chaotic zone when trajectory response is nonlinear.
  generation: 11
  Why it happens:
    Agents assume monotonic response and keep incrementing when the system is actually
      sensitive/chaotic.
  How to avoid:
    When two consecutive traces produce landing positions >1.5 units apart with small Delta,
      switch to bisection rather than linear search.

[cat_err_001]
  Description:
    Using x=1.0 at y=1.0 in elevated-arm scenes.
  generation: 11
  Why it happens:
    Agents choose an intermediate x value between 0.3 and 2.0 without checking validity.
  How to avoid:
    Skip x=1.0 at y=1.0 always; go directly to x=2.0 r=1.5 as the elevated fallback.

[cat_err_002]
  Description:
    Simulating the overshoot side of a y-chaotic cliff (y=0.3--0.5) without bisecting
      first.
  generation: 11
```

```
  Why it happens:
    Agents see a near-target landing on overshoot trace and try to simulate without
      confirming the cliff boundary.
  How to avoid:
    Always bisect the chaotic cliff first. Only simulate the overshoot boundary after
      bisection has narrowed to Deltay<=0.01 and no further resolution is possible.

[cat_err_003]
  Description:
    Placing the red ball at x>3 (far right of scene).
  generation: 11
  Why it happens:
    Agents try to increase rightward strike force by moving the contact point further
      right.
  How to avoid:
    Keep x within 0.1--0.5 for standard strikes; x=2.0 is maximum for elevated fallback.

[cat_err_004]
  Description:
    Treating x=0.2 as a universal range-reducer regardless of drop height.
  generation: 11
  Why it happens:
    Agents apply the x=0.2 escape learned at standard y to elevated y situations where x
      behavior is reversed.
  How to avoid:
    x=0.2 reduces range only at y<=0.0. At elevated y (~=0.5), x=0.2 and x=0.1 INCREASE
      range---use this deliberately when in a low-range y-dominant regime.
```

K.5 HExA system prompt (catapult seed 64)

The HExA prompt is the ReAct prompt of §K.2 plus the retrieved skills and mistakes, and a four-line `IMPORTANT` reminder to consult them after a failed simulation. The reminder is emitted only when the retrieved block is non-empty, since it refers to a section that does not otherwise exist; on an empty bank the prompt is byte-for-byte ReAct. It carries no physics, geometry or placement guidance of its own.

```
============================================================
  SYSTEM PROMPT
============================================================
[... ReAct system prompt (puzzle description, placement constraints, tool list,
  Thought/Action format and rules) omitted; identical to the ReAct prompt above ...]

=== LEARNED PHYSICS SKILLS ===

## Catapult-Specific Skills
1. [Strategy: Right-of-Pivot Arm Strike Zone (x~=0.2--0.5 Default)] Placing the red ball
  at x~=0.2--0.5 on the right half of the catapult arm creates a downward force that lifts
  the green ball in a rightward arc toward the basket. x=0.3 is the default starting
  point; x=0.2 at standard y reduces range ~1.3 units. Shifting x rightward (>=0.5) at
  standard-y positions risks diminishing returns.
   Apply when: Default for all catapult strikes; start with x=0.3. At moderate elevated
   y (~=0.5), decreasing x below 0.3 increases range (see leftward-x skill). Do not
   exceed x~=0.5 without checking for diminishing returns.
   Example: Seed 43: x=0.3 y=-0.5 -> green at x=3.10, success. Seed 48: x=0.3 y=0.3 ->
   success. Seed 51: x=0.3 y=0.0 -> success.

2. [Strategy: Low Drop (y~=-0.5 to 0.0) Maximizes Range; Binary-Search y First] Drop
  height y is the primary range control: y~=-0.5 produces maximum range, y~=0.0 produces
  moderate range, y~=0.3--0.5 produces reduced range. A +/-0.25 change in y at standard
  heights shifts landing x by 1.5--2.0 units. When standard y (<=0.0) is blocked, the
  operative range shifts to y=0.25--0.5, where lower y still produces more range.
   Apply when: Default for all seeds; start at y=-0.5, then binary-search toward y=0.0
   if overshooting. If gray_ball blocks all y<=0.0, shift to y=0.5 as first valid
   placement, then binary-search DOWN to y=0.3 or y=0.25 to increase range. Use y=-0.25
   as first midpoint in standard regime.
   Example: Seed 49: y=-0.5 blocked, y=0.0->x=4.26 (overshoot), y=0.25->x=3.71, success.
   Seed 53: y=0.0->x=4.45, y=0.25->x=4.19, y=0.5->x=4.17 (y-dominant regime detected).

3. [Strategy: Identical Range at Different x/r Signals y-Dominant Regime] When
  trace_green_ball returns nearly identical green ball landing positions despite
  changing x or radius, the scene is in a y-dominant regime where drop height controls
  range. Stop iterating x or r and switch to y-binary-search or x-sweep at a different y
  level. If lower y is blocked, try x<0.3 at moderate elevated y (~=0.5) to escape the
  regime.
   Apply when: When two consecutive traces with different x or r values produce landing
   positions within 0.2 units of each other; immediately switch to adjusting y. If y is
```

```
   blocked and regime persists at x=0.3, try x=0.1--0.2 at y=0.5 to break out.
   Example: Seed 44: x=0.3 and x=0.2 at r=2.0 both ->x=4.50. Seed 47: x=0.3 y=0.5 and
   various r all gave x~=2.69. Seed 53: y=0.0->x=4.45, y=0.25->x=4.19, y=0.5->x=4.17
   (plateau) -> switched to x=0.1 -> success.

4. [Strategy: Overlap Fallback Sequence: y=-0.5 -> y=0.0 -> y=0.5 -> Binary-Search
  y=0.3--0.5] When y=-0.5 is blocked, try y=0.0, then y=0.5 with r=2.0. Once a valid
  placement at y=0.5 is confirmed, binary-search DOWNWARD within y=0.3--0.5 to fine-tune
  range, since lower y in this zone still increases range. Only escalate to y=1.0 if
  y=0.3 is also blocked or all y=0.3--0.5 attempts miss by more than 1 unit.
   Apply when: When predict_first_contact returns invalid for y=-0.5; follow sequence
   y=0.0 -> y=0.5 r=2.0 -> binary-search y=0.3--0.5. Skip x=1.0 at y=1.0 always. For
   elevated green ball scenes (y>=-0.5), standard y slots may all be blocked and sequence
   starts at y=1.0.
   Example: Seed 48: y=-0.5/0.0 blocked -> y=0.5 (x=2.37 undershoot) -> y=0.3 (x=3.14,
   success). Seed 51: y=-0.5 blocked -> y=0.0 valid (gap 0.21, success). Seed 52:
   y=-0.5/0.0 blocked -> y=0.5 undershoot -> bisect down.

5. [Strategy: Chaotic Trajectory Zones: Avoid Fine-Tuning in Sensitive Ranges] Chaotic
  zones exist in both y and x dimensions where tiny parameter changes cause large
  trajectory swings: (1) y~=-0.15 to -0.25 at x=0.3; (2) y~=0.3--0.4 at r=2.0 in some seeds
  (Deltay=0.1 -> >1.9 range jump); (3) y~=1.05--1.1 at elevated arms; (4) x~=0.35--0.40 at
  y=0.75. When a chaotic cliff is detected, bisect methodically; the cliff boundary can
  be as narrow as Deltay=0.01.
   Apply when: When two consecutive traces with small Delta in x or y produce landing
   positions >1.5 units apart: for y-chaotic cliffs bisect (e.g., try y=0.35 between
   y=0.3 and y=0.4); for x-chaotic cliffs simulate overshoot side. Do not iterate
   further within the chaotic range without a clear bisection strategy.
   Example: Seed 44: y=0.5->x=4.50, y=0.7->x=-0.33 (y-cliff). Seed 52: y=0.3->x=4.23,
   y=0.4->x=2.71 (cliff); bisected to y=0.34/y=0.35 boundary; simulated y=0.34 -> success.

6. [Strategy: Trace-then-Simulate: Gap Threshold, Bisect Chaotic Cliffs, Simulate Narrow
  Boundary] At standard drop heights (y<=0.0), commit to simulate when gap is under 0.8
  units. At elevated heights (y=0.25--0.5), commit at gap <0.5. When a chaotic cliff is
  detected in y=0.3--0.5, bisect to the midpoint rather than simulating immediately.
  After bisection has narrowed the cliff to Deltay<=0.01 (minimum resolution), simulating
↪ the
  overshoot boundary IS valid---the green ball can contact the blue ball during its arc
  even if the final resting position overshoots.
   Apply when: At y<=0.0: gap <0.8 -> simulate. At y=0.25--0.5: gap <0.5 -> simulate. If
   chaotic cliff: bisect first. Once Deltay<=0.01 and no gap reduction is possible, simulate
   the overshoot side---arc contact is achievable at that boundary.
   Example: Seed 51: gap ~0.21 at y=0.0 -> success. Seed 52: cliff at y=0.34/0.35 after
   full bisection -> simulated y=0.34 (overshoot) -> success via arc contact.

## Common Mistakes to Avoid
1. **Fine-tuning x or y in a chaotic zone when trajectory response is nonlinear.**
   Why it happens: Agents assume monotonic response and keep incrementing when the
   system is actually sensitive/chaotic.
   How to avoid: When two consecutive traces produce landing positions >1.5 units apart
   with small Delta, switch to bisection rather than linear search.

2. **Using x=1.0 at y=1.0 in elevated-arm scenes.**
   Why it happens: Agents choose an intermediate x value between 0.3 and 2.0 without
   checking validity.
   How to avoid: Skip x=1.0 at y=1.0 always; go directly to x=2.0 r=1.5 as the elevated
   fallback.

3. **Simulating the overshoot side of a y-chaotic cliff (y=0.3--0.5) without bisecting
  first.**
   Why it happens: Agents see a near-target landing on overshoot trace and try to
   simulate without confirming the cliff boundary.
   How to avoid: Always bisect the chaotic cliff first. Only simulate the overshoot
   boundary after bisection has narrowed to Deltay<=0.01 and no further resolution is
   possible.

4. **Placing the red ball at x>3 (far right of scene).**
   Why it happens: Agents try to increase rightward strike force by moving the contact
   point further right.
   How to avoid: Keep x within 0.1--0.5 for standard strikes; x=2.0 is maximum for
   elevated fallback.

=== END SKILLS ===

============================================================
  USER PROMPT
============================================================
Solve the Catapult puzzle. You have 25 iterations to solve it. The success condition is:
   green_ball must contact blue_ball. Use tools effectively and think about alternate
  approaches if one does not work.
- **ALWAYS call predict_first_contact first** --- it is cheap and tells you if your red
```

```
  ball hits the intended object.
- Only if predict confirms the right contact: call trace_green_ball to track green ball.


IMPORTANT --- After each FAILED simulation, before trying a new placement, you MUST:
1. Review the LEARNED PHYSICS SKILLS listed in the system prompt above.
2. Identify which skill applies to the failure you just observed.
3. Explicitly state which skill you are applying and why in your next Thought.
```

K.6 HExA TRAJECTORY (CATAPULT SEED 64)

Highlighting is ours and is not part of the model's output: blue marks reasoning that invokes the injected skill bank, red marks the observations that drive each decision, and *grey* annotations give the bank entry being applied. Skill numbers (#1–#6) refer to the retrieved block in §K.5.

```
--- Step 1 ---
Thought:
Let me start by describing the scene geometry to understand the layout.

Action:
describe_scene_geometry({})

Observation:
=== Scene Geometry ===
World bounds: x in [-5, 5], y in [-5, 5]

Balls:
  green_ball: pos=(-3.06, -1.97), r=0.20, dynamic=True
  blue_ball: pos=(3.50, -1.99), r=0.32, dynamic=True
  black_ball: pos=(-1.17, 4.60), r=0.40, dynamic=False
  gray_ball: pos=(-1.14, -3.07), r=0.70, dynamic=True

Bars:
  ledge: pos=(3.50, -2.71), angle=-3.6 deg, len=3.01, dynamic=False
  black_platform: pos=(-1.74, -3.87), angle=0.0 deg, len=3.00, dynamic=False
  gray_platform: pos=(-1.14, -2.27), angle=0.0 deg, len=4.25, dynamic=True

Baskets:
  basket: pos=(3.50, -2.51), dynamic=True

Key distances:
  green<->blue: 6.56

Success condition: green_ball <-> blue_ball for 3.0s
==================================================

--- Step 2 ---
Thought:
The scene shows a catapult setup. The green ball is at (-3.06, -1.97) on the left end of
  the gray platform (catapult arm). The gray ball pivot is at (-1.14, -3.07). The blue
  ball is at (3.50, -1.99) in a basket. I need to drop a red ball on the right side of
  the catapult arm to launch the green ball toward the blue ball. Following skills, I'll
  start with x=0.3, y=-0.5, r=2.0. First, let me predict first contact.
  <- Skill #1 cat_ev_9_000 (x~=0.3 strike zone) + Skill #2 cat_ev_6_001 (low drop y=-0.5,
↪ search y first)

Action:
predict_first_contact({'x': 0.3, 'y': -0.5, 'radius': 2.0})

Observation:
=== Predicted First Contact ===
placement_valid: True
red_ball drop: x=0.300, y=-0.500, radius=2.000
first_contact_object: gray_platform
contact_step: 10
approach_speed: 1.464
contact_point (approx midpoint): (-0.420, -1.454)
contact_normal (red->other inverted): (0.664, 0.748)

--- Step 3 ---
Thought:
The red ball contacts the gray_platform (catapult arm) at step 10. This is correct - it
  hits the right side of the arm which should launch the green ball. Let me trace the
  green ball trajectory.

Action:
trace_green_ball({'x': 0.3, 'y': -0.5, 'radius': 2.0})
```

```
Observation:
=== Green Ball Trajectory (steps run: 360) ===
red_ball drop: x=0.300, y=-0.500, radius=2.000
Success: False
green_ball: start=(-3.06, -1.97) end=(+4.78, -2.49) Delta=(+7.84, -0.52) peak_y=+4.74 v_max
↪ =10.31

Waypoints (step: x, y):
  step    1: (-3.06, -1.97)
  step   30: (-2.47, +0.18)
  step   60: (-0.59, +3.72)
  step   90: (+1.27, +4.74)
  step  120: (+3.05, +3.07)
  step  150: (+4.77, -1.04)
  step  180: (+4.39, -2.25)
  step  210: (+4.46, -2.47)
  step  240: (+4.58, -2.47)
  step  270: (+4.78, -2.49)
  step  300: (+4.78, -2.49)
  step  330: (+4.78, -2.49)
  step  360: (+4.78, -2.49)

--- Step 4 ---
Thought:
The green ball overshoots, landing at x=4.78 while the blue ball is at x=3.50. I need to
  reduce range. Following skills, I should increase y (reduce drop height). Try y=0.0.
  <- Skill #2 cat_ev_6_001 / Skill #4 cat_ev_11_002 (step y from -0.5 toward 0.0)

Action:
trace_green_ball({'x': 0.3, 'y': 0.0, 'radius': 2.0})

Observation:
=== Green Ball Trajectory (steps run: 240) ===
red_ball drop: x=0.300, y=0.000, radius=2.000
Success: False
green_ball: start=(-3.06, -1.97) end=(-0.92, -3.09) Delta=(+2.14, -1.12) peak_y=+3.94 v_max
↪ =11.93

Waypoints (step: x, y):
  step    1: (-3.06, -1.97)
  step   30: (-2.97, -0.95)
  step   60: (-1.48, +3.94)
  step   90: (-1.36, +2.11)
  step  120: (-1.30, -2.32)
  step  150: (-0.92, -3.09)
  step  180: (-0.92, -3.09)
  step  210: (-0.92, -3.09)
  step  240: (-0.92, -3.09)

--- Step 5 ---
Thought:
Big trajectory swing between y=-0.5 (lands x=4.78, overshoot) and y=0.0 (lands x=-0.92,
  massive undershoot). This is a chaotic cliff per Skill #5. I should bisect at y=-0.25.
  <- Skill #5 cat_ev_9_006 (chaotic zones) + Mistake #1 (bisect, don't fine-tune)

Action:
trace_green_ball({'x': 0.3, 'y': -0.25, 'radius': 2.0})

Observation:
=== Green Ball Trajectory (steps run: 330) ===
red_ball drop: x=0.300, y=-0.250, radius=2.000
Success: False
green_ball: start=(-3.06, -1.97) end=(+3.29, -1.74) Delta=(+6.35, +0.23) peak_y=+3.78 v_max
↪ =10.75

Waypoints (step: x, y):
  step    1: (-3.06, -1.97)
  step   30: (-2.76, -0.35)
  step   60: (-1.25, +3.54)
  step   90: (+0.52, +3.78)
  step  120: (+2.36, +1.23)
  step  150: (+3.97, -0.87)
  step  180: (+3.70, -1.59)
  step  210: (+3.60, -1.59)
  step  240: (+3.31, -1.71)
  step  270: (+3.29, -1.74)
  step  300: (+3.29, -1.74)
  step  330: (+3.29, -1.74)

--- Step 6 ---
```

```
Thought:
At y=-0.25, the green ball lands at x=3.29, which is close to the blue ball at x=3.50
  (gap ~0.21). This is within the commit threshold of 0.8 per Skill #6. Let me simulate
  this.
  <- Skill #6 cat_ev_11_003 (trace-then-simulate: commit when gap < 0.8)

Action:
simulate_action({'x': 0.3, 'y': -0.25, 'radius': 2.0})

Observation:
SUCCESS! The green ball contacted the blue ball in the basket.
Total simulation steps: 423
Reward: 1.0

Final object positions:
  green_ball: pos=(3.289, -1.741), vel=(0.000, 0.000)
  red_ball: pos=(0.527, -2.229), vel=(0.000, 0.000)
  blue_ball: pos=(3.653, -2.106), vel=(0.000, 0.000)
  ledge: pos=(3.500, -2.712), vel=(0.000, 0.000)
  basket: pos=(3.501, -2.596), vel=(0.000, 0.000)
  black_ball: pos=(-1.170, 4.600), vel=(0.000, 0.000)
  black_platform: pos=(-1.738, -3.869), vel=(0.000, 0.000)
  gray_ball: pos=(-2.912, -3.064), vel=(0.000, 0.000)
  gray_platform: pos=(-0.730, -3.363), vel=(0.000, 0.000)

--- Step 7 ---
Thought:
(auto-finish after Success:True)

Action:
finish({'x': 0.3, 'y': -0.25, 'radius': 2.0})

Observation:
SUCCESS! The green ball contacted the blue ball in the basket.
Total simulation steps: 423
Reward: 1.0

Final object positions:
  green_ball: pos=(3.289, -1.741), vel=(0.000, 0.000)
  red_ball: pos=(0.527, -2.229), vel=(0.000, 0.000)
  blue_ball: pos=(3.653, -2.106), vel=(0.000, 0.000)
  ledge: pos=(3.500, -2.712), vel=(0.000, 0.000)
  basket: pos=(3.501, -2.596), vel=(0.000, 0.000)
  black_ball: pos=(-1.170, 4.600), vel=(0.000, 0.000)
  black_platform: pos=(-1.738, -3.869), vel=(0.000, 0.000)
  gray_ball: pos=(-2.912, -3.064), vel=(0.000, 0.000)
  gray_platform: pos=(-0.730, -3.363), vel=(0.000, 0.000)
```

How the bank drove the solve. After reading the scene, the agent starts from the bank's default: the right-of-pivot strike zone $x{\approx}0.3$ (`cat_ev_9_000`) with a low drop $y{=}-0.5$ and large radius $r{=}2.0$, searching over y first (`cat_ev_6_001`). One `predict_first_contact` call confirms that the red ball strikes the arm, and a trace shows the green ball overshooting to $x{=}4.78$ (blue ball at $x{=}3.50$). Raising the drop to $y{=}0.0$ swings the landing point to $x{=}-0.92$, a large undershoot. Rather than micro-tuning, the agent recognises this jump as a chaotic cliff (`cat_ev_9_006`) and bisects to $y{=}-0.25$, which lands the green ball at $x{=}3.29$. This gap of ≈ 0.21 is below the commit threshold of `cat_ev_11_003` (Trace-then-Simulate), so the agent spends its only `simulate_action` on this placement, which succeeds. The whole solve takes three traces and one simulation.

L Working Example: Episodic RAG vs. HExA on Level Pass the Parcel of InterPhyre

Our second example compares HExA with the strongest baseline, Episodic RAG, on `pass_the_parcel` seed 57. The goal is to knock an inverted top basket off its platform so that the green ball it covers falls onto the blue ball in the bottom basket. Both agents use Claude Sonnet 4.6 as the actor, the same tools and the same 25-iteration budget, and both run with frozen memory on held-out seeds 57–106. HExA uses its round-17 bank; Episodic RAG uses its store of 51 raw rollouts (25 successes, 26 failures) collected on the training seeds. The only difference is what the memory contains. RAG retrieves two complete past rollouts (6,862 added tokens). HExA retrieves six distilled skills and three mistakes. RAG uses all 25 iterations and fails; HExA succeeds in 5 iterations with final placement ($x{=}-0.072$, $y{=}1.3$, $r{=}0.7$).

The system prompts follow the templates of Appendix I.4 (Episodic RAG) and Appendix K.4 (HExA), on top of the `pass_the_parcel` REACT prompt (Appendix I.1.3), so we omit them. RAG's retrieved memory is two complete rollouts: a seed-33 success, final placement $(0.934, 0.916, 0.5)$, and a seed-28 failure; we do not reproduce them. Highlighting follows §K.6: blue marks reasoning that draws on the retrieved memory, red marks decisive observations, and *grey* lines are our annotations.

L.1 HExA RETRIEVED SKILLS (PASS_THE_PARCEL SEED 57)

```
=== LEARNED PHYSICS SKILLS === (titles only; principle, "Apply when" and
  "Example" fields omitted)

## Pass The Parcel-Specific Skills
1. [pas_ev_7_001] Radius r=0.7 Default; Escalate to r=1.0 Before r=1.2 Only When Basket
  Actually Moved
2. [pas_ev_6_002] Switch to Opposite Side When Hit Direction Misaligns with Blue
3. [pas_ev_7_000] Direct Lateral Sidewall Hit: Side Selection and y Fallback Sequence
4. [pas_ev_2_002] Basket Must Clear Green's Fall Path by >0.7 Units Horizontally
5. [pas_ev_7_004] Lower y (0.5-1.7) for Direct Sidewall Contact When Basket Barely Moves
6. [pas_ev_7_007] Ramp Approach When All Direct Sidewall Heights Are Blocked

## Common Mistakes to Avoid
1. [pas_err_000] Agent escalates directly to r=1.2, r=1.5, or r=2.0 when r=0.7 strands
  green, causing green to overshoot far past blue.
2. [pas_err_001] Agent places the red ball directly above the inverted basket expecting
  downward force to dislodge it, but the ball lands on the basket's solid top surface
  and sits there.
3. [pas_evo_002] Agent exhausts radius escalations (r=0.7, r=1.0, r=1.2) at the same
  high y when the basket barely moves, failing to recognize that the contact angle ---
  not the force --- is the problem.
=== END SKILLS ===
```

L.2 EPISODIC RAG TRAJECTORY (PASS_THE_PARCEL SEED 57)

```
.... Step 1 condensed (thought omitted):
  [1] get_level_state({}) -> scene description

--- Step 2 ---
Thought: Based on the past successful rollout (seed 33), the winning strategy was
  placing the red ball just above and slightly to the right of the top basket. The
  basket in this seed is at (0.428, 0.051). Following the same pattern (offset ~0.15
  right, ~0.87 above the basket), I'll try x=0.578, y=0.921, radius=0.5. But let me
  first try the exact same coordinates that worked in seed 33: x=0.934, y=0.916,
  radius=0.5.
  <- reuses seed 33's absolute placement; here the basket sits 0.36 units further left
Action: simulate_action({'x': 0.934, 'y': 0.916, 'radius': 0.5})
Observation: FAILURE. The green ball did NOT contact the blue ball.
  green_ball: pos=(-0.085, 0.206), vel=(0.000, 0.000)
  blue_ball: pos=(-1.329, -4.455), vel=(0.000, 0.000)
  top_basket: pos=(0.292, 0.126), vel=(0.000, 0.000)
  ....

--- Step 3 ---
Thought: The basket moved slightly but green didn't fall to blue. Let me try placing the
  ball closer to the basket center, directly above it to push it down and off the
  platform.
  <- the drop-on-top error HExA's Mistake #2 (pas_err_001) warns against
Action: simulate_action({'x': 0.428, 'y': 1.2, 'radius': 0.5})
Observation: FAILURE. The green ball did NOT contact the blue ball.
  green_ball: pos=(0.428, 0.210), vel=(0.000, 0.000)
  blue_ball: pos=(-1.329, -4.455), vel=(0.000, 0.000)
  top_basket: pos=(0.429, 0.116), vel=(0.000, 0.000)
  ....

--- Step 4 ---
Thought: The basket barely moved. I need to push it harder to the left so green falls
  into the bottom basket. Let me try placing the ball to the right of the basket to push
  it left with more force.
  <- wrong side: blue is LEFT of the basket (cf. HExA Skill #2, pas_ev_6_002)
Action: simulate_action({'x': 1.2, 'y': 0.1, 'radius': 0.7})
Observation: INVALID placement:
  - Overlaps with green_ball: dist=0.7977, sum_of_radii=0.9500. Move at least 0.2023
    units further away.
  - Overlaps with black_platform: dist_to_platform=0.2492 < radius=0.7000. Move at least
    0.5008 units further away from the platform.
  ....
```

```
.... Steps 5--20 condensed (thoughts omitted):
  [5] simulate_action({'x': 1.5, 'y': 0.5, 'radius': 0.6}) -> FAILURE; green at (0.428,
    0.301)
  [6] simulate_action({'x': 3.5, 'y': 4.0, 'radius': 0.5}) -> FAILURE; green at (-1.314,
    -2.999)
  [7] simulate_action({'x': 3.7, 'y': 4.2, 'radius': 0.6}) -> FAILURE; green at (-1.390,
    -2.995)
  [8] simulate_action({'x': 0.6, 'y': 0.95, 'radius': 0.5}) -> INVALID placement
  [9] simulate_action({'x': 0.8, 'y': 1.1, 'radius': 0.5}) -> FAILURE; green at (0.931,
    0.206)
  [10] simulate_action({'x': 1.6, 'y': 0.3, 'radius': 0.8}) -> INVALID placement
  [11] simulate_action({'x': 1.8, 'y': 0.5, 'radius': 0.7}) -> INVALID placement
  [12] simulate_action({'x': 1.8, 'y': 0.8, 'radius': 0.7}) -> FAILURE; green at (0.428,
    0.210)
  [13] simulate_action({'x': 4.0, 'y': 4.4, 'radius': 0.5}) -> FAILURE; green at
    (-1.299, -3.001)
  [14] simulate_action({'x': 4.2, 'y': 4.4, 'radius': 0.7}) -> INVALID placement
  [15] simulate_action({'x': 4.2, 'y': 4.2, 'radius': 0.7}) -> FAILURE; green at
    (-4.725, -4.560)
  [16] simulate_action({'x': 0.65, 'y': 1.3, 'radius': 0.4}) -> FAILURE; green at
    (0.428, 0.210)
  [17] simulate_action({'x': 3.5, 'y': 4.0, 'radius': 0.6}) -> FAILURE; green at
    (-1.360, -2.994)
  [18] simulate_action({'x': 3.6, 'y': 4.3, 'radius': 0.55}) -> FAILURE; green at
    (-1.393, -3.684)
  [19] simulate_action({'x': 3.65, 'y': 4.3, 'radius': 0.6}) -> FAILURE; green at
    (-1.387, -2.990)
  [20] simulate_action({'x': 3.6, 'y': 4.3, 'radius': 0.58}) -> FAILURE; green at
    (-1.349, -3.001)

--- Step 21 ---
Thought: With r=0.55 green was at y=-3.684 (very close to blue), but r=0.58 sent green
  back to y=-3.0. Let me try r=0.56 to find the sweet spot.
  <- steps 17-25: r tuned within 0.55--0.60 at almost fixed (x, y) on the ramp; green
     keeps stopping on the basket rim and the budget runs out
Action: simulate_action({'x': 3.6, 'y': 4.3, 'radius': 0.56})
Observation: FAILURE. The green ball did NOT contact the blue ball.
  green_ball: pos=(-1.429, -3.689), vel=(0.000, 0.000)
  blue_ball: pos=(-1.290, -4.451), vel=(0.000, 0.000)
  top_basket: pos=(-1.416, -2.891), vel=(0.000, 0.000)
  ....

.... Steps 22--25 condensed (thoughts omitted):
  [22] simulate_action({'x': 3.6, 'y': 4.3, 'radius': 0.57}) -> FAILURE; green at
    (-1.432, -3.689)
  [23] simulate_action({'x': 3.4, 'y': 4.3, 'radius': 0.56}) -> FAILURE; green at
    (-1.447, -3.000)
  [24] simulate_action({'x': 3.6, 'y': 4.3, 'radius': 0.555}) -> FAILURE; green at
    (-1.434, -3.690)
  [25] simulate_action({'x': 3.8, 'y': 4.0, 'radius': 0.55}) -> FAILURE; green at
    (-1.407, -3.700)
```

L.3 HEXA TRAJECTORY (PASS_THE_PARCEL SEED 57)

```
--- Step 1 ---
Thought: Let me start by getting the current level state to understand the positions of
  all objects.
Action: get_level_state({})
Observation: === Level State: pass_the_parcel ===
* green_ball (green Ball): position=(0.4279, 0.3008), radius=0.2500, dynamic=True
* blue_ball (blue Ball): position=(-1.3292, -4.0400), radius=0.3600, dynamic=True
* top_basket: {'x': 0.42794570936818155, 'y': 0.050799519646729074, 'color': 'gray',
  'dynamic': True, 'type': 'Basket', 'angle': 180, 'scale': 0.6}
  ....

--- Step 2 ---
Thought: Blue ball is at x=-1.3292 (LEFT of basket at x=0.4279). Per Skill 2, I should
  use a left-side hit. Per Skill 3, basket_y=0.0508 > 0, so start at y=2.0. Placement:
  x=basket_x-0.5=0.4279-0.5=-0.0721, y=2.0, r=0.7.
  <- Skill #2 pas_ev_6_002 (blue left of basket -> x=basket_x-0.5), Skill #3 pas_ev_7_000
     (basket_y>0 -> y=2.0), Skill #1 pas_ev_7_001 (r=0.7 default)
Action: simulate_action({'x': -0.0721, 'y': 2.0, 'radius': 0.7})
Observation: FAILURE. The green ball did NOT contact the blue ball.
  green_ball: pos=(0.428, 0.210), vel=(0.000, 0.000)
  blue_ball: pos=(-1.691, -4.447), vel=(0.000, 0.000)
  top_basket: pos=(0.431, 0.116), vel=(0.000, 0.000)
  ....
```

```
--- Step 3 ---
Thought: Basket barely moved (from 0.0508 to 0.116 = 0.065 units). Per Skill 5 and
  Mistake 3, basket moved <0.12 units at y=2.0, so I should NOT escalate radius.
  Instead, lower y to 0.5. Try x=-0.0721, y=0.5, r=0.7.
  <- Skill #5 pas_ev_7_004 + Mistake #3 pas_evo_002: wrong contact angle, not too little
↪ force
Action: simulate_action({'x': -0.0721, 'y': 0.5, 'radius': 0.7})
Observation: INVALID placement:
  - Overlaps with green_ball: dist=0.5383, sum_of_radii=0.9500. Move at least 0.4617
    units further away.
  - Overlaps with black_platform: dist_to_platform=0.6492 < radius=0.7000. Move at least
    0.1008 units further away from the platform.
  ....

--- Step 4 ---
Thought: y=0.5 is blocked by green ball and platform. Per Skill 5, try y=1.0 next.
Action: simulate_action({'x': -0.0721, 'y': 1.0, 'radius': 0.7})
Observation: INVALID placement:
  - Overlaps with green_ball: dist=0.8596, sum_of_radii=0.9500. Move at least 0.1404
    units further away.
  ....

--- Step 5 ---
Thought: y=1.0 blocked by green ball. Per Skill 5, try y=1.3 next.
  <- Skill #5 fallback ladder y=0.5 -> 1.0 -> 1.3 at fixed x, r=0.7
Action: simulate_action({'x': -0.0721, 'y': 1.3, 'radius': 0.7})
Observation: SUCCESS! The green ball contacted the blue ball.
  green_ball: pos=(-1.606, -4.032), vel=(0.000, 0.000)
  blue_ball: pos=(-1.171, -4.452), vel=(0.000, 0.000)
  top_basket: pos=(1.251, 0.119), vel=(0.000, 0.000)
  ....

.... Step 6 condensed (thought omitted):
  [6] finish({'x': -0.0721, 'y': 1.3, 'radius': 0.7}) -> SUCCESS! The green ball
    contacted the blue ball; green at (-1.606, -4.032)
```

Why raw episodes fail where distilled skills succeed. Both agents face the same three questions: which side to strike the basket from, at what height, and what to change after a miss. The retrieved RAG episodes answer none of them in a form that transfers. The agent copies seed 33's *absolute* placement (step 2), although this seed's basket sits 0.36 units further left. It then drops the ball on top of the basket (step 3), strikes from the right although the blue ball is on the left (steps 4–12), and finally settles on the ramp, fine-tuning the radius between 0.55 and 0.60 for its last nine calls while the green ball keeps stopping on the basket rim. HEXA's bank states the same experience as rules conditioned on the geometry. The side rule (Skill #2) and the height rule (Skill #3) produce a sensible first placement, x=basket$_x-0.5$, y=2.0, r=0.7, computed rather than copied. When the first strike barely moves the basket (0.065 units), Skill #5 and Mistake #3 read this as a wrong contact angle rather than too little force. The agent therefore keeps the radius fixed and steps y down its fallback ladder ($0.5 \to 1.0 \to 1.3$); the two blocked placements cost no simulation. The third height succeeds.

M WORKING EXAMPLE: MEMRL VS. HEXA ON WORLD CIRCLE OF DISCOVERPHYSICS

Our third example compares HEXA with MemRL on the `circle` world of DISCOVERPHYSICS (Appendix E.3), seed 300. One centre particle and ten ring particles interact under a hidden fractional-Laplacian law. The agent sets only the ring radius, a shared tangential velocity and the measurement times, and within 10 rounds must submit an executable law together with a prose explanation. An episode passes if normalised MSE is below 0.1 and the explanation score, from the independent `claude-opus-4-6` judge, is at least 0.75. Both arms use the same actor and prompts (Appendix J.3) and frozen memory built within the world. They differ only in the retrieved block: HEXA receives 11 skills and 6 mistakes from its frozen bank, and MemRL two archived episodes (4,689 added tokens). Across the 15 held-out seeds the judge's mean explanation score is 0.887 for HEXA and 0.540 for MemRL; HEXA passes on 9 seeds and MemRL on none.

The benchmark logs the agent's thoughts with numbers and law names masked (`<n>`, `<law-form>`), and we keep that masking. Submitted laws and explanations are shown unmasked. Experiment

actions are condensed to one entry per experiment (radius `r`, tangential velocity `v`, measurement times `t`), raw observations are omitted, and submitted code is cut to its key lines (`....`). Highlighting follows §K.6.

M.1 RETRIEVED MEMORY (CIRCLE SEED 300)

```
=== RETRIEVED SKILLS (titles only) ===
- [dis_ev_5_000] Map the measured exponent onto a named field-theoretic/operator
  formalism, not just 'anomalous decay'
- [dis_ev_3_000] Never declare a cross-validated constant/exponent as fittable ---
  including 'derived' ones
- [dis_ev_1_001] A structural interpretation ambiguity is not evidence of numerical
  uncertainty
- [dis_sk_002] Extract exponents from early-time, negligible-displacement acceleration
- [dis_sk_003] Use zero tangential velocity to isolate the central force
- [dis_ev_2_004] Use nonzero tangential velocity to test velocity-dependence and
  conservation laws
- [dis_ev_2_008] Conserved-quantity checks as an independent cross-check of
  centrality/velocity-independence
- [dis_ev_5_009] A tight, drift-free power-law fit is good evidence but does not by
  itself guarantee full credit --- still check for curvature before over-trusting the
  naive familiar-law framing
- [dis_ev_4_002] Trajectory accuracy and explanation quality are scored and gated
  independently
- [dis_ev_4_004] State the 'uniform coupling across all particles' assumption explicitly
  in the explanation
- [dis_ev_4_010] Cross-check exponent/constant invariance across radius range, but
  recognize it does not by itself resolve the causal/formalism ambiguity

=== RETRIEVED MISTAKES (first sentence only) ===
- [dis_evo_000] Explanations describe the discovered law only in terms of the
  exponent/coupling relative to familiar gravity ('softer/steeper than usual',
  'anomalous decay') and the ring-collapse method used, without naming the theoretical
  operator/formalism the exponent implies or mapping it to the standard parameter (e.g.
  alpha) for that formalism.
- [dis_evo_001] An episode achieves near-perfect or excellent trajectory error yet still
  fails overall because the explanation score is judged inadequate.
- [dis_err_002] Agent declares an already well-constrained constant (coupling strength,
  exponent), including ones only 'derived' or backed out geometrically rather than
  directly measured, as a 'fittable parameter' via **params, letting the evaluator
  re-fit it on the episode's own training data --- with unpredictable (sometimes better,
  sometimes much worse) results.
- [dis_evo_003] A clean, cross-validated power-law fit (consistent exponent across
  several tested radii) is treated as fully sufficient for the explanation, when what's
  still missing is connecting that exponent to a named physical/operator formalism.
- [dis_err_003] Agents repeatedly tried to invoke an MSE-fitting tool (run_mse_fit) late
  in the session and got 'trajectory CSV logging disabled' errors, wasting a round, then
  fell back to manual/eyeball fitting without adjusting confidence or explanation
  accordingly.
- [dis_err_004] Agents wasted an entire round attempting to shell out via a 'no_tag'
  bash-like action, which is not a valid action in this protocol, forfeiting a round
  with no data gained.

=== RETRIEVED MEMORIES (2 of 2; bodies omitted except one sentence) ===
### Memory 1 --- FAILURE (world W6, seed 214, utility Q=-0.987, updates=16)
### Memory 2 --- SUCCESS (world W6, seed 202, utility Q=-0.987, updates=15)

Memory 1, "What went wrong" (excerpt): ... Future runs should sweep
  initial_tangential_velocity systematically at fixed ring_radius across several
  fractions of the naive circular value and record long time series (many revolutions)
  via measurement_times, since orbit shape (closed vs. precessing vs. escaping) over
  long horizons is far more diagnostic than early-time acceleration alone. ...
```

M.2 MEMRL TRAJECTORY (CIRCLE SEED 300)

```
--- Round 1 ---
Thought: I'll start by measuring early-time radial dynamics at zero tangential velocity
  across several ring radii to extract the force scaling with distance.
Action: run_experiment (4 experiments shown; log truncated after these): r=2 v=0
  t=0.1..5 (10 times); r=4 v=0 t=0.1..5 (10 times); r=6 v=0 t=0.1..5 (10 times); r=8 v=0
  t=0.1..5 (10 times)
Observation: [noisy positions and velocities of all 11 particles at each time]
  ....

--- Round 2 ---
Thought: Data at t=<n> across ring radii <n>/<n>/<n>/<n> shows early radial velocity
```

```
  magnitudes that fit a*r^<n> ~= constant almost perfectly (checked via a*r^<n> giving
  ~<n> in all four cases, and the <n>x radius ratio giving the predicted <n> velocity
  ratio to within <n>%). This points to an attractive all-pairs force ~<n>/d^<n>. I'll
  now refine this over a wider radius range with finer early-time sampling, and check
  orbital shape at nonzero tangential velocity for consistency.
Action: run_experiment (5 experiments): r=1 v=0 t=0.01..0.1 (4 times); r=3 v=0
  t=0.01..0.1 (4 times); r=10 v=0 t=0.01..0.1 (4 times); r=20 v=0 t=0.01..0.1 (4 times);
  r=5 v=0.4 t=2..40 (10 times)
Observation: [noisy positions and velocities of all 11 particles at each time]
  ....

--- Round 3 ---
Thought: The exponent fits cleanly to n~=<n> (force ~ <n>/d^<n>) with an aggregate
  radial constant K~=<n> consistent across radii <n>--<n> I'll now test a predicted
  circular-orbit velocity to confirm the force law and check for velocity-independence.
Action: run_experiment (3 experiments): r=5 v=0.6128 t=5..90 (10 times); r=5 v=0
  t=0.01..0.1 (10 times); r=15 v=0 t=0.01..0.05 (5 times)
Observation: [noisy positions and velocities of all 11 particles at each time]
  ....

--- Round 4 ---
Thought: The r=<n> circular-orbit test with v=<n> (predicted from K~=<n>, n=<n>) shows
  particles maintaining constant radius ~<n> out to t=<n>, confirming the force law. The
  r=<n> early-time data also matches the predicted K/r^<n> scaling within noise. I'll
  run one more precision check to nail down the exponent and coupling constant before
  finalizing.
Action: run_experiment (4 experiments): r=1.5 v=0 t=0.005..0.02 (4 times); r=7 v=0
  t=0.005..0.02 (4 times); r=12 v=0 t=0.005..0.02 (4 times); r=5 v=2 t=1..12 (5 times)
Observation: [noisy positions and velocities of all 11 particles at each time]
  ....

--- Round 5 ---
Thought: The n=<n> power-law fits cleanly across radii <n>--<n> (log-log slope ~=<n>),
  and the r=<n> orbit at the predicted circular velocity stayed circular through t=<n>,
  confirming velocity-independence of the radial force. I'll now test a sub-circular
  velocity at r=<n> over many periods to check for orbital precession, which will
  further validate the exponent via Bertrand's theorem behavior.
  <- follows Memory 1's advice (long, multi-period orbits at sub-circular velocities). The
     orbit tests confirm the power law already measured in round 2; nothing in the memory
↪ asks
     for the operator, the coupling structure or which parameters to leave free
Action: run_experiment (1 experiment): r=5 v=0.45 t=5..120 (10 times)
Observation: [noisy positions and velocities of all 11 particles at each time]
  ....

--- Round 6 ---
Thought: The v=<n> sub-circular orbit (below the ~<n> circular velocity) shows periodic
  radial <law-form> with precession --- consistent with an attractive power-law force
  with exponent !=<n> (non-<law-form>ian), matching my n~=<n> fit. Combined with the
  clean r^<n> scaling across radii <n>--<n> and the confirmed circular orbit at the
  predicted velocity, I'm confident in the functional form and will submit.
Action: submit_final_law
def fit_parameters():
    return {
        "G": {"init": 0.84, "bounds": [0.1, 3.0]},
        "q": {"init": 1.5, "bounds": [1.0, 2.5]},
    ....
def discovered_law(positions, velocities, duration, **params):
    ....
    G = params.get("G", 0.84)
    q = params.get("q", 1.5)
    ....
        inv = 1.0 / (dist ** (q + 1.0))
    ....
<explanation> The universe contains an all-to-all attractive interaction: every particle
  pulls every other particle toward it with an acceleration whose magnitude falls off as
  a power law G/d^q of their separation d, with the best-fit exponent near q~=1.5
  (softer than an inverse-square law). Ring particles collapse radially under zero
  initial velocity with radial acceleration scaling cleanly as r^-1.5 across nearly two
  decades of radius, and orbits at the predicted circular velocity remain stable
  circles, while sub-circular velocities produce precessing (non-closed) elliptical-like
  orbits --- a signature that the force law departs from the inverse-square (Keplerian)
  case as expected from Bertrand's theorem. The center particle participates in this
  same pairwise coupling as an ordinary member of the system, not as a fixed potential
  source.
  <- same exponent as HExA, but no operator is named, and both G and q are left free
Observation: norm_mse=3.990587765259953, expl_score=0.8, passed=False
  Opus-4.6 judge: explanation 0.5; FAIL (nMSE 3.99 > 0.1, explanation < 0.75)
```

M.3 HEXA TRAJECTORY (CIRCLE SEED 300)

```
--- Round 1 ---
Thought: I'll start with zero-tangential-velocity experiments at multiple radii to
  isolate the central radial force, using early-time snapshots to measure acceleration
  before significant displacement.
  <- dis_sk_003 (zero tangential velocity isolates the central force) + dis_sk_002 (early-
↪ time,
     negligible-displacement acceleration)
Action: run_experiment (3 experiments): r=3 v=0 t=0.05..10 (10 times); r=6 v=0
  t=0.05..10 (10 times); r=10 v=0 t=0.05..10 (10 times)
Observation: [noisy positions and velocities of all 11 particles at each time]
  ....

--- Round 2 ---
Thought: Clean signal: acceleration scales as r^<n> across all three radii with a highly
  consistent effective constant (~=<n> = a*r^<n>). This matches a fractional power-law
  central attraction, though the ring topology makes it ambiguous whether this is purely
  from the center particle or a pairwise sum over all <n> particles with equal coupling.
  Next I'll test with nonzero tangential velocity to check angular momentum conservation
  and centripetal balance, which will validate that the force is purely radial and
  velocity-independent.
  <- dis_ev_1_001 (structural ambiguity) + dis_ev_2_004 (nonzero velocity tests conservation
↪  laws)
Action: run_experiment (4 experiments): r=3 v=0.698 t=0.5..10 (7 times); r=3 v=0.3
  t=0.5..10 (7 times); r=1 v=0 t=0.05..0.4 (4 times); r=20 v=0 t=0.05..0.4 (4 times)
Observation: [noisy positions and velocities of all 11 particles at each time]
  ....

--- Round 3 ---
Thought: Angular momentum is well conserved in the tangential-velocity runs (L stays
  near its initial value across all measured times), confirming the force is purely
  central and velocity-independent. Velocity data (less noisy than positions) at early
  times show acceleration scaling as r^<n> with an effective constant ~=<n>, consistent
  across radii <n>--<n> (r=<n> was noisier due to faster collapse). Let me get one more
  wide radius sweep at very early times to nail down the exponent and constant precisely
  before finalizing.
  <- dis_ev_2_008 (conserved-quantity cross-check) + dis_ev_4_010 (invariance across radius
↪ range)
Action: run_experiment (5 experiments): r=2 v=0 t=0.02..0.06 (3 times); r=4 v=0
  t=0.02..0.06 (3 times); r=8 v=0 t=0.02..0.06 (3 times); r=15 v=0 t=0.02..0.06 (3
  times); r=30 v=0 t=0.02..0.06 (3 times)
Observation: [noisy positions and velocities of all 11 particles at each time]
  ....

--- Round 4 ---
Thought: The wide radius sweep (r=<n> to <n>, spanning a factor of <n>) confirms the
  acceleration follows a clean power law a ~= <n>/r^<n> with essentially no drift
  (regression slope <n>, R^2 near <n>). Angular momentum conservation from the earlier
  tangential-velocity runs confirms the force is purely central and
  velocity-independent. This points to an attractive pairwise interaction with a
  fractional decay exponent (a*r^<n>=const), consistent with a fractional-order field
  operator of order alpha~=<n> in <n>D (rather than the standard <n>D
  <law-form>/<law-form> analog which would give <n>/r). I'm confident in the functional
  form; I'll finalize now.
  <- dis_ev_5_000 (map the exponent onto a named operator formalism)
Action: submit_final_law
def discovered_law(positions, velocities, duration, **params):
    ....
    C = params.get("C_pair", 0.167)
    ....
            factor = C / dist ** 2.5
    ....
def fit_parameters():
    return {
        "C_pair": {"init": 0.167, "bounds": [0.02, 0.6]},
    ....
<explanation> All 11 particles interact through a uniform, all-pairs attractive coupling
  whose magnitude falls off as 1/r^1.5 with pairwise separation --- steeper than the
  natural 2D Newtonian analog (1/r, from the standard 2D Poisson equation) but shallower
  than an inverse-square law. This decay rate is consistent with a fractional-order
  field operator (a fractional Laplacian of order alpha~=0.75 in the -(-Delta)^alpha
  convention), rather than the integer-order Laplacian that gives the familiar
  logarithmic/1/r 2D potential. I measured the exponent from early-time
  (negligible-displacement) accelerations across radii spanning 2--30 with zero
  tangential velocity, finding a consistent power-law slope of -1.50 with no curvature
  across that entire range, and confirmed the force is purely central and
  velocity-independent by checking angular momentum conservation in runs with nonzero
  tangential velocity. The ring symmetry makes the effective per-particle constant
  degenerate between a "force from center only" and "pairwise sum over all particles"
```

```
  interpretation; I resolved this by adopting the more general pairwise model with a
  uniform coupling constant applied between every particle pair (including the center),
  leaving that coupling constant as the only fittable parameter since its value depends
  on how the geometric ring-sum decomposes.
  <- dis_ev_5_000: names the operator and its order
  <- dis_ev_4_004: states the uniform-coupling assumption
  <- dis_ev_3_000: exponent fixed at 1.5 in the code; only C_pair is free
Observation: norm_mse=2.2647538526815615e-06, expl_score=1.0, passed=True
  Opus-4.6 judge: explanation 1.0; PASS at both the 0.75 and 0.90 gates
```

Same measurement, different submission. Both agents open with the same zero-velocity ring collapse and both measure the same $1/r^{1.5}$ all-pairs attraction. The difference lies in what the memory makes them do next. MemRL's retrieved failure reflection tells it to trust long, multi-period orbits over early-time scaling, so it spends rounds 3–6 on circular-orbit and precession tests that only confirm the exponent it already has. It then submits a generic power law G/d^q with both G and q free, and an explanation that never names the operator; it fails on both axes. HExA's bank encodes what the scorer actually checks. Its experiments follow the *procedural* skills: zero velocity to isolate the central force, early times for the exponent, conserved angular momentum as a cross-check and a wide radius sweep for invariance. Its submission follows the *reporting* skills: it names the fractional Laplacian with $\alpha \approx 0.75$, states the uniform-coupling assumption and fixes the measured exponent in code, leaving only the coupling free. It passes in 4 rounds with normalised MSE 2×10^{-6} and an explanation score of 1.0.

N Working Example: Episodic RAG vs. HExA on Mastermind with Qwen3-8B

Our last example is Mastermind (Appendix E.1) with the small Qwen3-8B actor. The agent must find a hidden 3-digit code (digits 1–6, no repeats) in 3 guesses per episode, and each instance is played over 3 episodes with the *same* code. After each guess it receives black pegs (right digit, right place) and white pegs (right digit, wrong place). Both arms use frozen memory and identical prompts, apart from the retrieved block. HExA injects four short strategies and one mistake; Episodic RAG injects two complete past rollouts from other instances. On instance `59cedcb50e280335` (secret code `5 1 4`), HExA solves all three episodes, taking 3, 3 and then 1 turn. RAG fails all three. Across the six saved evaluation runs of each arm, HExA solves this instance in all six and RAG in two.

Qwen3-8B reasons at length (2k–24k characters per turn), so we show a few verbatim sentences per turn, with . . . . marking omitted text. The observation after each guess is reduced to its peg feedback. Highlighting follows §K.6; in the RAG transcript, red marks the deduction errors.

N.1 Retrieved memory (Mastermind instance 59cedcb50e280335)

```
=== HExA: LEARNED PROBLEM-SOLVING STRATEGIES (titles; examples omitted) ===
1. [Strategy: Cross-Episode Feedback Utilization] Leverage historical feedback to
   eliminate impossible digit combinations across episodes.
2. [Strategy: White Peg Inference] Track digits via white peg feedback to narrow
   possible positions, even if misplaced.
3. [Strategy: Progressive Constraint Narrowing] Systematically eliminate impossible
   combinations by combining black and white peg feedback across guesses.
4. [Strategy: Feedback-Driven Guess Adjustment] Adjust guesses based on all feedback
   from previous attempts, not just immediate results.
Mistake 1. Failing to use white peg feedback to eliminate misplaced digits and track
   code composition across guesses.

=== Episodic RAG: Retrieved Past Attempts (condensed to guesses and feedback) ===
Past attempt 1 --- SUCCESS (instance e7308e8bdfffd6f3, code 3 5 2)
  ep1: 1 2 3 (0B/2W) -> 2 1 4 (0B/1W) -> 3 5 2 SOLVED;  ep2: 1 2 3 -> 3 5 2;  ep3: 3 5 2
Past attempt 2 --- FAILURE (instance b401c1e7503ff9d3)
  ep1: 1 2 3 (0B/1W) -> 2 4 5 (1B/1W) -> 3 4 2;  ep2: 5 4 2 -> 2 5 1 -> 5 2 4;
  ep3: 5 2 4 -> 2 5 4 -> 4 5 2   (all FAILED)
.... The hidden part of the task differs from instance to instance, so nothing above
tells you this instance's answer -- read them for which search behaviour paid off
and which wasted episodes, and do not replay a past action sequence.
```

N.2 Episodic RAG trajectory (3 episodes, never solved)

```
--- Episode 1, turn 1 ---
Reasoning (6,054 chars; excerpt): .... Looking at the past attempts, the first one was
  successful. .... So the code was 3 5 2. .... Wait, but in the first instance, the code
  was 3 5 2, so maybe that's a common code? ....
Guess: 1 2 3   ->   0B/1W

--- Episode 1, turn 2 ---
Reasoning (10,397 chars; excerpt): .... But wait, in the past attempt, they tried 2 1 4
  after 1 2 3. .... Wait, the previous successful attempt used 2 1 4 as the second
  guess. .... So I'll choose 2 1 4 as the next guess. ....
Guess: 2 1 4   ->   2B/0W
  <- replays the retrieved rollout's second guess, although that rollout had seen 0B/2W here,
↪  not 0B/1W

--- Episode 1, turn 3 ---
Reasoning (10,848 chars; excerpt): .... So the code is 2 _ 4, with the middle digit not
  being 2 or 4. .... So the third guess would be 2 3 4. ....
Guess: 2 3 4   ->   out of turns (episode failed)
  <- 3 is already excluded by turn 1; the consistent codes were 2 5 4, 2 6 4, 5 1 4 and 6 1
↪ 4

--- Episode 2, turn 1 ---
Reasoning (15,927 chars; excerpt): .... From Episode 1, the code is 2 1 X. .... So the
  answer is \boxed{1 3 2}. ....
Guess: 1 3 2   ->   0B/1W
  <- 2 1 X contradicts turn 1 (it would put two of 1, 2, 3 in the code)

.... condensed (reasoning omitted):
  [Ep 2, turn 2] guess 2 4 1 -> 0B/2W
  [Ep 2, turn 3] guess 2 5 4 -> out of turns (episode failed)
  [Ep 3, turn 1] guess 2 1 5 -> 1B/1W
  [Ep 3, turn 2] guess 2 5 4 -> 1B/1W

--- Episode 3, turn 3 ---
Reasoning (16,951 chars; excerpt): .... Looking at the past attempts, the first instance
  was solved quickly by guessing 3 5 2. .... I think I'll have to guess 2 5 4, even
  though it doesn't fit all the clues. ....
Guess: 2 5 4   ->   out of turns (episode failed)
  <- repeats its previous guess; never tests 5 1 4, a code consistent with its own episode-1
↪  feedback
```

N.3 HEXA TRAJECTORY

```
--- Episode 1, turn 1 ---
Reasoning (1,965 chars; excerpt): .... Since there's no prior information, I should
  start with a combination that covers different digits to maximize the information from
  the feedback. ....
Guess: 1 2 3   ->   0B/1W

--- Episode 1, turn 2 ---
Reasoning (9,856 chars; excerpt): .... So the code has exactly one of these digits, and
  the other two are different digits not in the guess. ....
Guess: 1 4 5   ->   0B/3W

--- Episode 1, turn 3 ---
Reasoning (14,751 chars; excerpt): .... So the code has 1, 4, and 5. .... So the code
  can't have 1 in position 1, 4 in position 2, or 5 in position 3. .... So let's list
  all permutations of 1,4,5 and eliminate those that violate these constraints. ....
Guess: 5 1 4   ->   SOLVED
  <- Skill #2 White Peg Inference + #3 Progressive Constraint Narrowing: 3W on 1 4 5 fixes
↪ the
     digits, the zero black pegs fix the positions

--- Episode 2, turn 1 ---
Reasoning (10,796 chars; excerpt): .... But the strategies mention leveraging historical
  feedback. .... So each episode is the same task, but the code is different each time.
  ....
Guess: 1 2 3   ->   0B/1W
  <- cites Skill #1 but misreads the setup: the code is fixed across episodes

--- Episode 2, turn 2 ---
Reasoning (7,412 chars; excerpt): .... In Episode 1, after guessing 1 2 3 (0B, 1W), then
  1 4 5 (0B, 3W), then 5 1 4 (correct). .... But maybe the user is trying to apply the
  strategies mentioned, like cross-episode feedback. ....
Guess: 1 4 5   ->   0B/3W

--- Episode 2, turn 3 ---
```

```
Reasoning (8,347 chars; excerpt): .... Looking at the history from Episode 1, the
  correct code was [5,1,4]. .... Therefore, the code is a permutation of 1,4,5. ....
Guess: 5 1 4   ->   SOLVED

--- Episode 3, turn 1 ---
Reasoning (11,652 chars; excerpt): .... Looking at the previous episodes, in Episode 1
  and 2, the correct code was [5, 1, 4]. .... Let's choose 5,1,4 as the third guess.
  ....
Guess: 5 1 4   ->   SOLVED
  <- Skill #1 Cross-Episode Feedback Utilization: solved in 1 turn
```

Why a small actor needs rules, not transcripts. Both agents open with `1 2 3` and receive the same 0B/1W. RAG then copies the second guess of the retrieved success (`2 1 4`), although that rollout reached it after different feedback, and despite the prompt's instruction not to replay a past action sequence. It then draws a deduction its own first guess rules out (`2 3 4`). In episodes 2 and 3 it keeps reasoning about "`2 1 X`", a pattern turn 1 already excludes, and ends by repeating a guess it has just played. It never tries `5 1 4`, one of only four codes consistent with its own episode-1 feedback. HExA's bank holds no past moves, only how to read the pegs. With Skill #2 and Skill #3, a probe that scores 0B/3W pins down the digits and the zero blacks pin down the positions. With Skill #1 it reuses the solved code in later episodes. It misreads the episode structure in episode 2 and re-derives the code, but by episode 3 it submits `5 1 4` immediately.

O Working Example: Episodic RAG vs. HExA on MathGAP

Our final example is a MathGAP word problem solved by `gpt-oss-120b` (reasoning effort `high`). The problem mixes irrelevant sentences with count statements, transfers ("X gives N to Y") and signed-difference equalities; the answer depends on reading the statements in order, since a transfer changes the counts that later statements refer to. Held-out item 22 of run 0 is one of the 20 items that HExA answers correctly in all three evaluation runs and Episodic RAG answers wrongly in all three. The paper evaluations stored only the parsed answers, so we regenerated this item through each arm's own frozen-evaluation code, with the frozen round-29 bank and the frozen RAG store and with the generation seed each arm drew in the paper run. Both regenerations reproduce the logged answers: HExA gives **27** (correct) and RAG gives **79** (wrong). The reasoning traces run to 40k (HExA) and 32k (RAG) characters; we show verbatim sentences, with `....` marking omitted text.

O.1 Problem (MathGAP held-out item 22)

```
Wilbert has 58 more broccolis than Hillary. Boyce has 58 more broccolis than Samuel.
  Richard and Joan have been married for 13 years. Ethelyn is highly introverted and
  prefers minimal communication with others. Jon has 168 broccolis. Clarence has 168
  broccolis. The number of broccolis that Samuel has more than Boyce is equal to the
  difference between the number of broccolis that Wendy and Clarence have. The number of
  broccolis that Hillary has more than Wilbert is equal to the difference between the
  number of broccolis that Vivian and Jon have. Wendy gives 52 broccolis to Richard.
  Samuel is 21 years old. Lois is 46 years old. Joanne has 12 more broccolis than Joan.
  Vivian gives 52 broccolis to Addie. Beatrice is very outgoing and frequently hosts
  parties at home. Clarence is highly introverted and prefers minimal communication with
  others. Addie and Jean are good friends who often go fruit or vegetable picking
  together on weekends. Carrol has 12 more broccolis than Kathrine. Angelica and Wilbert
  are good friends who often go fruit or vegetable picking together on weekends. Hillary
  is 52 years old. Wendy and Olivia are good friends who often go fruit or vegetable
  picking together on weekends. The number of broccolis that Joan has more than Joanne
  is equal to the difference between the number of broccolis that Olivia and Wendy have.
  The number of broccolis that Kathrine has more than Carrol is equal to the difference
  between the number of broccolis that Nels and Vivian have. Vivian tends to be
  laid-back and prefers staying in rather than going out. Jon is 42 years old. Ethelyn
  has 19 more broccolis than Angelica. Wade and Boyce are good friends who often go
  fruit or vegetable picking together on weekends. Joanne is very generous and enjoys
  sharing food with others. Kathrine is very generous and enjoys sharing food with
  others. Beatrice has 19 more broccolis than Lois. Carrol is highly introverted and
  prefers minimal communication with others. Nels is very generous and enjoys sharing
  food with others. The number of broccolis that Angelica has more than Ethelyn is equal
  to the difference between the number of broccolis that Wade and Olivia have. The
  number of broccolis that Lois has more than Beatrice is equal to the difference
  between the number of broccolis that Jean and Nels have. How many broccolis does Jean
  have?
```

With Samuel−Boyce=−58 and Hillary−Wilbert=−58, Wendy and Vivian start at 168−58=110, and both give away 52. The Nels statement comes after Vivian's transfer, so Nels=58−12=46 and Jean=46−19=27. Using Vivian's count before the transfer gives Nels=98 and Jean=79.

O.2 RETRIEVED MEMORY

```
=== HExA: LEARNED PROBLEM-SOLVING STRATEGIES (titles only; 10 strategies, 1 mistake) ===
1. Directionality of more/fewer
2. Target variable dependency analysis
3. Solve signed difference equations with unknowns
4. Signed difference equality parsing
5. Chronological evaluation
6. Sequential transfer handling
7. Conditional transfer handling
8. Implicit zero initialization for unknown persons in transfers
9. Equal redistribution and splitting handling
10. Direct count assignment
Mistake 1. Treating signed difference statements as absolute differences or swapping
  term order

=== Episodic RAG: Retrieved Past Rollouts (bodies omitted) ===
Past rollout 1 --- SUCCESS (seed ...a_6-s_60-idx_19; a tomato puzzle, answer 11)
  .... Given the pattern from previous successful rollout (mathgap), we need to
  interpret statements about "has X tomatoes" as initial counts (i.e., before any
  transfers). ....
Past rollout 2 --- FAILURE (seed ...a_1-s_0-idx_11; answer 79, INCORRECT)
  same numbers and structure as the target puzzle, with renamed people and no
  irrelevant sentences; its correct answer is also 27
```

O.3 EPISODIC RAG REASONING

```
[reading the retrieved success] .... Given the pattern from the successful rollout of a
  similar puzzle (mathgap), we need to interpret statements like "X has Y more broccolis
  than Z" as a difference: X = Z + Y (or X - Z = Y). We need to track each individual's
  initial count (before any transfers). ....

[which counts do the differences use?] .... In the successful rollout, they interpreted
  relational equations as referring to final counts after all transfers? Then they
  considered relational constraints that refer to counts after any prior transfers? Or
  maybe all relational constraints refer to the final counts after all transfers. Given
  the pattern from previous successful rollout (mathgap), we need to interpret
  statements about 'has X tomatoes' as initial counts (i.e., before any transfers). ....
  <- quoting rollout 1, a different puzzle; the question of timing is never settled from the
↪  text itself

[Nels, Jean on initial counts] .... So -19 = Je_i - 98 => Je_i = 98 - 19 = 79. Thus
  Jean's initial count = 79. Thus the answer is 79. Since there are no transfers
  affecting Jean, final count = 79. ....
  <- Ne_i = 98 uses Vivian's count before she gives away 52 (Vi_i = 110); the Nels statement
↪  comes after that transfer

[final check] .... But before finalizing, we should verify the pattern from the previous
  successful rollout: they solved a similar puzzle with many statements. Thus answer is
  79. ....
  <- 79 is also the recorded (INCORRECT) answer of retrieved rollout 2, the same puzzle with
↪  renamed people

Final: The final answer is $\boxed{79}$.
```

O.4 HEXA REASONING

```
[setup] .... Also we need to consider that after transfers, the counts for those persons
  change. We must apply transfers in the order they appear. So we need to parse the
  statements in order. ....
  <- Skill #5 Chronological evaluation, #6 Sequential transfer handling

[Wendy, Vivian before their transfers] .... Thus we can determine Wendy's count before
  any transfers. However note that later Wendy gives 52 to Richard, which will affect
  her count after transfer. But the equation (7) uses the counts at that point (before
  transfer) because it's earlier in order. Thus Vivian's count before any transfers =
  110. ....

[the transfers] .... We have Vivian's count before transfer = 110 (from equation 8).
```

```
  Thus after transfer, Vivian's count = 58. ....

[Nels, Jean] .... So -12 = N - Vivian => N = Vivian - 12 = 58 - 12 = 46. Thus Nels = 46.
  So -19 = Je - N => Je = N - 19 = 46 - 19 = 27. Thus Jean = 27. ....

[self-check] .... Now we must confirm that the equations that involve Wendy and Vivian
  (like those that appear after the transfers) indeed use the updated counts. According
  to "Chronological evaluation" strategy, later statements must use counts after earlier
  statements have been applied. Actually equation 8 (the difference between Vivian and
  Jon) is before the transfer. Let's check the order: The statement about "The number of
  broccolis that Hillary has more than Wilbert is equal to the difference between the
  number of broccolis that Vivian and Jon have." appears before the "Vivian gives 52
  broccolis to Addie." So we used Vivian's pre-transfer count (110) for that equation.
  ....
  <- names Skill #5 while re-checking which count each statement sees

Final: The final answer is $\boxed{27}$.
```

The same puzzle in memory is not enough. RAG's store holds this very puzzle, with renamed people, as retrieved rollout 2, together with its verdict: INCORRECT, 79. It still reproduces that answer. The retrieved success is a different puzzle, and what RAG takes from it is an interpretation, that "has" statements give counts before any transfer, which it then applies to the difference statements as well. It debates the timing question several times but never settles it from the problem text, and it computes Nels from Vivian's pre-transfer count. HEXA's bank states the rule as a rule (Skill #5: later statements use the counts after all earlier statements have been applied). The actor applies it while solving and cites it by name while re-checking, which yields Nels=46 and the correct answer of 27.