DISCOVER-MANY-WORLDS: REINFORCEMENT LEARNING FOR SCIENTIFIC DISCOVERY IN AUTOMATICALLY GENERATED PHYSICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Scientific discovery in physics requires a process of designing experiments, revising hypotheses, and inferring unfamiliar laws from observations. We introduce DISCOVER-MANY-WORLDS, a reinforcement learning framework for training scientific discovery agents in automatically generated physics worlds. Our world-generation pipeline combines LLM-driven construction, procedural simulation, and verification testing to generate and validate diverse environments at various difficulty levels with no human intervention. We train a 4B-parameter Qwen model with GRPO, increasing its capabilities on DISCOVERPHYSICS up to the performance level of Claude Sonnet 4.6. We also observe improved performance on other scientific and reasoning benchmarks. We demonstrate that the RL-trained agent engages in more extensive experimentation, where it gathers more observations before committing to an explanation and considers a broader range of physical laws instead of defaulting to familiar interactions. These results demonstrate that automatically generated physical worlds can provide effective training environments for experimentation and reasoning about physical phenomena, both of which are essential for scientific discovery.

1 INTRODUCTION

Scientific discovery is an active process of hypothesizing, experimentation, and analysis. In general, one does not simply observe a static dataset and aim to interpret what is going on, but instead must decide precisely what should be measured, and how to either experimentally confirm or refute a hypothesis. Recent benchmarks have begun to test the *scientific* ability of large language model (LLM) based agents: they are tasked to probe and design experiments in simulated worlds with the goal of discovering some ground-truth model of the scientific system (Koblischke et al., 2025; Zheng et al., 2026; Wiemann et al., 2026; Chen et al., 2026). While many frontier LLMs have shown strong performance in simpler real-world environments, these same models tend to struggle when interacting with simulated environments that are complex, contain noisy observations, or have physical laws that deviate from those of our own universe (Zheng et al., 2026; Wiemann et al., 2026). The extent to which interactive discovery can be learned remains an open question (Chen et al., 2026). An ideal framework to probe this ability is physics-based environments. These environments are governed by ground truth laws, with object trajectories being governed by explicit equations of motion that one may simulate numerically.

Training an agent to improve its ability to make novel discoveries is a non-trivial task. Recent improvements in mathematics and coding skills have come from reinforcement learning (RL) with verifiable rewards, where a proof checker or a unit test decides, cheaply and unambiguously, whether a candidate solution is correct. Scientific discovery has no such verifier. There is no analogue of Lean for a physical theory: whether an explanation of an observed phenomenon is right is precisely what the scientific process aims to establish, and a theory is truly only ever confirmed against future experiments that previous theories could not have predicted. Any attempt to train a discovery agent must therefore first construct a setting in which the ground truth is known.

Real experimental data is not well suited to the task for two reasons. Running experiments in the physical world is slow and expensive, which rules out the on-policy interaction loop that RL re-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

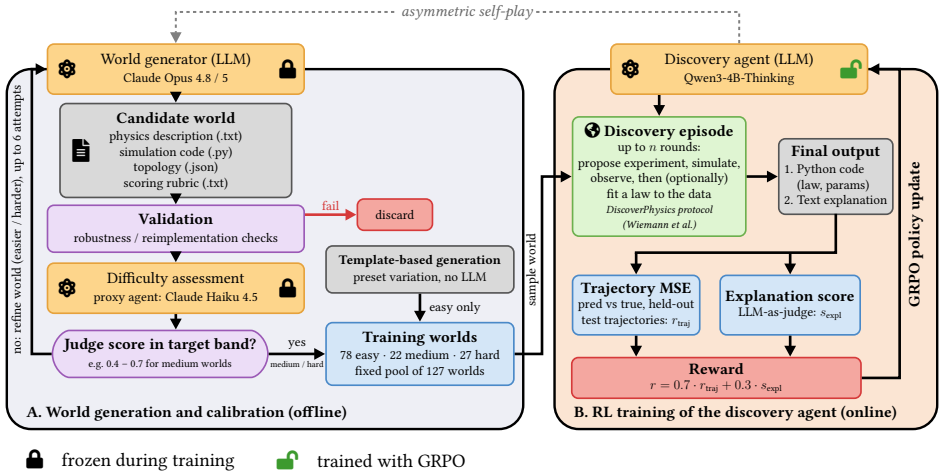

Figure 1: **Panel A:** A schematic of world generation pipeline. We automate our training data generation of physics worlds by using either a template (gray box) or a LLM (orange box). The template-based worlds are automatically verified and added to the dataset pool, while the LLM-generated worlds under validation, an optional difficulty assessment and refinement, and then get added to the pool. **Panel B:** A schematic of our RL training pipeline using the generated worlds. Our discovery agent, Qwen3 4B conducts experiments in a discovery episode, and repeats until producing final output that is evaluated, which then forms the reward for the RL training. The training data consists of the generated worlds in **A**.

quires, where an agent must be free to propose a new measurement, observe the result, and revise its hypothesis iteratively. And more fundamentally, real data can only be supervised by physics we already know, and the laws of our universe are precisely what is best represented in the pretraining corpus of an LLM. Simulated worlds governed by alternative physical laws resolve both problems at once.

Even with a known ground truth, it is not obvious how to reward the models. The most natural metric is the accuracy of the predicted trajectories, but a low trajectory MSE is a property of an effective theory, not necessarily of a correct one: a sufficiently flexible functional form with a few free parameters can interpolate the data an agent has collected without identifying any of the structure that generated it. Historically, empirically adequate models have often preceded the underlying principle, and it is the principle that generalizes beyond the experiments that motivated it. Predictive accuracy and conceptual understanding are therefore distinct axes, and a reward must account for both.

The DISCOVERPHYSICS benchmark (Wiemann et al., 2026) provides an N-body simulator and an experimental protocol, but its 22 worlds were each designed by a physicist, with a force law, a textual description, and a scoring rubric written by hand. This is far too few for RL, and every additional world costs human effort.

In this paper we introduce DISCOVER-MANY-WORLDS, a pipeline for automatically generating simulated worlds with physics that differ from our own, and demonstrate that RL on these worlds improves a small 4B open language model at scientific discovery, showing that a 4B model reaches Sonnet level performance on the DiscoverPhysics benchmark. The self-generated worlds come in three distinct difficulty levels, easy, medium, and hard, with the automated pipeline performing a robust series of validation and adversarial testing to ensure the governing force laws of these worlds are consistent with their text-based description.

2 METHODOLOGY

Our training environment has two components: a *world generator* and a *discovery agent*. The generator produces a world: a plain-text description of its physics, Python code that simulates it, and a rubric for judging explanations of it. The discovery agent runs experiments in the world and,

at the end of an episode, submits a candidate law with a text-based explanation. It follows the rules of DISCOVERPHYSICS (Wiemann et al., 2026) and is rewarded on both the predictive accuracy of its law and the quality of its explanation (Section 5.1).

In this work, only the discovery agent is trained. The generator is a fixed frontier model, which we found far more reliable than the 4B model being trained at producing worlds whose description, code and rubric agree. Worlds are generated, validated and calibrated for difficulty offline, and the resulting pool of 127 worlds is fixed throughout training. This lets every world be tested (Section 3.3), independently of the policy. An invalid world, whose law cannot be recovered or whose description contradicts its code, would otherwise contribute only noise. We also include easy worlds built from randomized hand-written force-law templates, which need no LLM and give the agent a validated set of simple worlds on which to learn the output format, the experiment interface and the fitting tool.

Our pipeline is summarized in Figure 1. This two component structure generalizes to a setting where the generator is itself a policy, rewarded for proposing worlds at the frontier of what the agent can solve, as in asymmetric self-play (Sukhbaatar et al., 2018) and self-play theorem provers (Dong & Ma, 2025). We leave an implementation of this framework to future work.

3 AUTOMATED WORLD GENERATION

Here, we describe the offline world generation pipeline used for RL training of the discovery agent. The pipeline consists of two components: (i) *template-based* generation of easy worlds, in which force laws are drawn from a fixed library of kernels and varied in their functional form, signs, couplings, particle counts, geometry, external fields and spatial dimension from a set of parameters we predetermine, and (ii) *free-form* generation of medium and hard worlds, in which the generator agent writes the force law, text based description, and rubric to evaluate the discovery agent's solution from scratch, and each candidate is then validated and calibrated for difficulty before being accepted. We describe each in turn.

To specify worlds, we follow the conventions of DISCOVERPHYSICS (Wiemann et al., 2026). A world consists of: (1) a plain text description of the physics, (2) a Python implementation of the force law, executed by the shared N-body simulator as described in Wiemann et al. (2026), (3) a topology file specifying the initial particle configuration and the experimental interface, i.e. which quantities the agent may set (initial positions, velocities, couplings, masses, measurement times) and the format of its final answer, and (4) a rubric that will be used to evaluate the conceptual understanding of the discovery agent, beyond how predictive its proposed law is. An example of a law is given in Appendix J

Section 3.1 describes the template-based generation of easy worlds, Section 3.2 the free-form generation of medium and hard worlds, and Section 3.3 the validation framework used to filter out invalid worlds.

3.1 TEMPLATE-BASED GENERATION (EASY WORLDS)

We build our templates on the DISCOVERPHYSICS simulator by varying the parameters that are passed to 3 of the 8 existing force law functions (kernels) in DISCOVERPHYSICS : power law ($F = C\, q_i q_j\, r^n$), Cornell/confining ($F = q_i q_j \left(\frac{\alpha}{r^2} + \sigma\right)$), and Plummer (softened inverse-square, $F = \frac{C\, q_i q_j\, r}{(r^2+a^2)^{3/2}}$. The parameter variation occurred over a fixed grid of possible parameters and consisted of varying C, n, a, α, σ; adding in external fields; varying the coupling type; adding multiple species of particles; changing the number of particles initially interacting in the world, and changing the spatial dimensions from 2D to 3D. The set of parameter variations (number of particles, coefficients, exponents) were chosen with Claude Opus 4.7's suggestions for easy modifications to the existing functions. The full list of modifications and an example of the format of these template, easy worlds is in the Appendix E.

3.2 FREE-FORM WORLD GENERATION (MEDIUM AND HARD WORLDS)

To produce harder, more diverse and creative physics, we prompt the generator agent to produce worlds that contain hidden latent structure: physical structure that shapes the dynamics but is never

exposed through the observable interface. The model is instructed to choose exactly one kind (as opposed to a mixture) of hidden structure to design a world around it.

3.2.1 HARD WORLDS

The hard requirements are concreteness, identifiability, well-behaved dynamics, and computational tractability. The full hard-world prompt is in Appendix F. We generate the worlds in a sequential fashion to encourage diversity. Once a world has been generated, it is passed along with the prompt and the model is asked to produce a world that differs substantially from all of the already generated ones. We create 27 hard difficulty level worlds using this method. These worlds passed the validation checks explained in Section 3.3. The generator agent was Claude Fable 5.

3.2.2 MEDIUM WORLDS

We find the original set of generated hard worlds are too difficult for weaker agents like Qwen3 4B. As a means to add curriculum to Qwen's learning, we devise a strategy to produce a batch of medium difficulty worlds that would serve as a bridge between the easy and hard worlds. We simplify the above requirements and instruct the model to include one type of latent structure aimed at a medium level difficulty.

We then conduct a difficulty assessment using the base Qwen3 4B model as the discovery agent. If Qwen solves the generated world with an evaluation score between 0.4 and 0.7, then the world is added to the pool of RL training worlds as a successfully-generated medium world. If not, it undergoes one of two possible refinement strategies outlined below. For the refinement stages, we switched the discovery agent to Claude Haiku, which is slightly stronger than Qwen3 4B at this task.

Refinement Strategy 1: Feedback. We give the generator agent feedback by returning the world that it generated along with the LLM judge's evaluation score and reasoning of why the discovery agent did not pass it. We then instruct it to construct a new world considering this feedback.

Refinement Strategy 2: Iterative improvement. We give the agent a world that was either too easy or too hard, and then we ask it to modify it accordingly so that it would be at a medium difficulty, in an interactive way. Starting worlds were abandoned after 6 iterative attempts of trying to get them in the medium scoring band and failing.

We successfully created 50 medium worlds in total, but only used the 22 two-particle worlds in RL training for easier difficulty. The medium worlds were generated by Claude Opus 4.8 (8 of the 50, including 2 of the 22 used for training) and Opus 5 (the remainder of the worlds). All prompts used to create the worlds are in Appendix F, and a table summarizing the types of worlds, how they were produced, and how many were created, can be found at Appendix A.

3.3 WORLD VALIDATION

Filtering matters because an invalid world contributes only noise to training: either its law cannot be recovered by the discovery agent, or its description and law disagree so that the evaluation score cannot be improved no matter what the agent does. Our validation filtering allows for the LLM generated examples to still be grounded in physical constraints.

Automated analytical physics checks. The template-based worlds inherited the physics-based tests that are in the original DISCOVERPHYSICS benchmark. These tests are both analytic physics trajectories compared against the numerical simulator trajectories and also physics conservation laws. Only some worlds have conservation laws that are applicable, and these are indicated in the world definition. The possible tests available include testing the numerical acceleration against the analytic calculation, linear and angular momentum conservation, keeping center-of-mass motion constant or stationary, and checking circular orbit analytical calculations.

Free-form worlds have no guarantee that the generator agent has created a textual description of the physics that matches the Python implementation: the generator may write code that does not match the textual description, it may omit details needed to reproduce the law, or the code may contain numerical instabilities. Our tests are designed to assess whether a world is discoverable if its plain text description contains enough information to constrain the law. Below, we outline

our main two tests, **Robustness** and **Reimplementation with adversarial testing**. Medium worlds underwent a **Robustness** check and the hard worlds underwent the **Reimplementation** check. The reimplementation check is more expensive, as it requires a call to an LLM and also a search for adversarial tests, so we restricted its use to the smaller number of hard worlds generated.

Robustness check We check that the generated Python implementation itself is numerically sound. We use the topology file's standard initial conditions as the starting test, and choose 8 other random initial conditions. We record any execution crashes, particles escaping the domain, and a noise floor: the MSE between the trajectory and the trajectory from initial conditions perturbed by 10^{-4} of their range. A case whose noise floor exceeds the pass threshold is dominated by numerical sensitivity, since no faithful reimplementation could match it. A medium world is discarded during the generation process if more than a quarter of its cases crash or are sensitivity-dominated.

Reimplementation check with adversarial testing. A separate agent, Claude Sonnet 4.6, implements the world's code from the generator's textual description of the world alone. We then run both implementations on the topology's fixed test cases and require the normalized MSE between their trajectories to be less than the noise floor. This check rejected three hard-difficulty-level worlds out of thirty.

Agreement on fixed test cases does not rule out regions of the input space where the two implementations may diverge (e.g., near singularities) or where the integrator becomes unstable, as these areas in the initial condition space may not have been tested with the set of fixed initial condition test cases. Therefore, we adversarially look for experimental conditions that would maximise the MSE between the original generated code and the reimplementation, at the level of the resulting trajectories. These edge cases are the most difficult physics the reimplementation must get right in order to be faithful to the generator's implementation because they result in maximal disagreement with the true trajectory of the particles. We call this maximized MSE the adversarial MSE.

The initial condition space consists of the particles' initial positions and velocities, within fixed bounds, for durations $T \in \{2, 3, 4, 5\}$. We use projected sign-gradient ascent with random restarts (Madry et al., 2018), a multi-start extension of the Fast Gradient Sign Method (FGSM, Goodfellow et al., 2015), taking gradients with respect to the initial-condition space (position, velocity, time). Because the reimplementation is arbitrary code, gradients are estimated by finite differences (6 random and 2 warm starts, 20 steps and 4000 evaluations per world). A world is invalid if the adversarial MSE between two trajectories of slightly perturbed initial conditions is 10 times larger than the simulator's own numerical noise floor and also is greater than the machine precision (10^{-8}). The noise floor is calculated by comparing the MSE between the trajectory of the ground truth at initial conditions perturbed by 10^{-4}. Any divergence would indicate an implementation artifact rather than physics, since the trajectories are not consistent. No world accepted by the previous check showed such divergence, but it is possible other worlds would have been discarded due to this method. We include it for rigor. See Figure 7 for an illustration on one of the hard worlds.

4 Reinforcement Learning training

We cast scientific discovery as a multi-turn RL problem. We follow the discovery pipeline implemented by DiscoverPhysics (Wiemann et al., 2026), whereby an LLM (Qwen3-4B-Thinking) is placed in a simulated world governed by force laws. The LLM then submits experiments, observes the resulting noisy trajectories, optionally fits candidate equations to its data, finally submitting a Python executable law together with a natural-language explanation of the underlying physics governing the particle trajectories. We optimise the policy with GRPO on a pool of 127 training worlds, using a reward that combines the normalised trajectory error of the submitted law with an LLM judge's score of the explanation.

Reward. Each episode ends with a candidate law $\hat{f}$ and a prose explanation e. Let $\varepsilon = \mathrm{MSE}(\hat{f})/\sigma^2_{\mathrm{gt}}$ denote the trajectory error of $\hat{f}$ on held-out test trajectories, normalized by the variance of the ground-truth trajectories (σ^2_{gt}) of that world (as measured from the variance of the test trajectories), with pass threshold $\tau = 0.1$. The trajectory reward is

$$r_{\mathrm{traj}}(\hat{f}) = \begin{cases} 0,\ 0.05,\ 0.1 & \text{no law / fails to run / non-finite trajectories,} \\ 0.1 + 0.9\left(\frac{1}{2}(1+\varepsilon/\tau)^{-1} + \frac{1}{2}\,\mathbf{1}[\varepsilon < \tau]\right) & \text{otherwise,} \end{cases} \tag{1}$$

where $\mathbf{1}[\cdot]$ is the indicator function, equal to 1 when the enclosed condition holds and 0 otherwise. The indicator term is the benchmark's own pass criterion, where the term $1/(1+\varepsilon/\tau)$ is a smooth monotone surrogate which is equal to 1 at $\epsilon = 0$, $1/2$ at the pass threshold, and decaying as τ/ε, so that within a GRPO group near-misses are ranked above large misses even when no rollout passes. An LLM judge scores the explanation $s_{\text{expl}}(e) \in [0, 1]$. The final reward mixes the two scores,

$$r = (1 - w)\, r_{\text{traj}}(\hat{f}) + w\, s_{\text{expl}}(e), \qquad w = 0.3, \tag{2}$$

so a correct law with a perfect explanation earns $r = 1$ and a correct law with a rejected explanation earns at most 0.7. Episodes that submit no law receive $s_{\text{expl}} = 0$. The dual reward axis is important as we aim to improve the agent's ability to understand novel physical environments, as opposed to simply improve its curve fitting ability.

Policy optimization. We train Qwen3-4B-Thinking-2507 with Group Relative Policy Optimization (GRPO; Shao et al., 2024), implemented in verl (Sheng et al., 2024) with vLLM (Kwon et al., 2023) for rollouts. Each training step samples 32 worlds from the shuffled 127-world training set and generates a group of 8 episodes per world at temperature 1.0 for 256 episodes per step. Each episode receives the reward r_i of equation 2, and GRPO uses the group-normalized advantage $\hat{A}_i = (r_i - \text{mean}(\{r_j\}))/\text{std}(\{r_j\})$. We take a single gradient step on the full batch, so every update is on-policy, using a learning rate of $1e-6$, with no KL penalty used. See Appendix B for the full policy gradient equation used for each update. Training ran on 4 nodes of 8 H200 GPUs and we perform 850 update steps, taking a total of ~ 4400 GPU hours.

5 Results

We next present the results of the trained model, subsection 5.1 covers training dynamics and behavioural changes of the science agent, subsection 5.2 presents the DiscoverPhysics results, per-world breakdown, and error analysis and subsection 5.3 explores the transfer to alternate external benchmarks.

5.1 RL training dynamics and behavioural changes

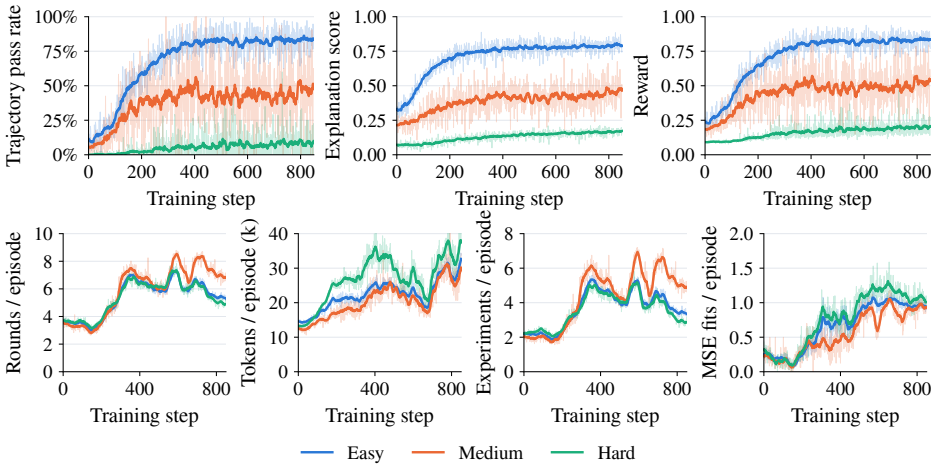

Figure 2: **Top row:** GRPO training curves for Qwen3-4B-Thinking over 850 steps on the 127-world suite, split by difficulty. Bold lines are 15-step rolling means, faint lines are raw per-step values. Panels show the fraction of episodes with normalized trajectory MSE below 0.1, mean LLM-judge score of the prose explanation, and shaped reward. **Bottom row:** Per-episode rounds, generated tokens, experiments, and fitting calls. The policy learns to experiment more, reason longer, and adopt the fitting tool it initially ignored.

We show the training curves for the automated world pass rate, explanation score, and reward from left to right in the top row of Figure 2 respectively. We show the distinction between the performance on the a priori assessed difficulty of different worlds in color. As expected, we see the strongest performance from all metrics on the easy worlds, with the performance being significantly worse on

the medium and hard worlds. It is also noteworthy that it takes roughly 200 training steps until there is any improvement on the hard worlds.

Qualitative changes in policy behavior during RL. In the bottom row of Figure 2 we show the changes to the number of experimental rounds run by the agent, the total token usage, experiments submitted, and MSE fitting calls used per episode. We see that for all difficulty levels, the policy learns to increase the number of experimental rounds from ~ 4 to $\sim 6-8$ per world solve, with the token usage also increasing by roughly a factor of 3. The number of experiments performed (as shown in the bottom row, third panel of Figure 2) has an initial share rise after ~ 200 steps, then begins to decrease after this. We see a small, but relatively monotonic rise in the number of MSE fits performed per episode rising from ~ 0.5 to ~ 1 over the course of the 850 steps.

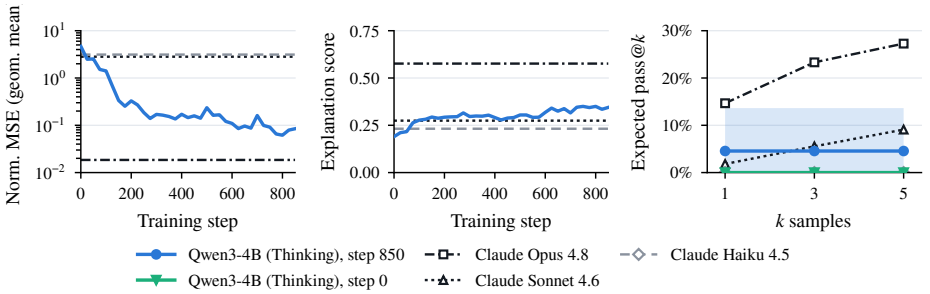

Figure 3: DISCOVERPHYSICS validation of the RL policy against Claude Opus 4.8, Sonnet 4.6, and Haiku 4.5 from the public leaderboard. We show the normalized trajectory MSE (lower is better), explanation score , and benchmark pass@k at step 850 (blue) and step 0 (green) from left to right respectively. Shaded bands are 95% intervals from a hierarchical bootstrap over worlds and over the five rollouts within each world (1000 replicates). The leaderboard reference curves are published point estimates without intervals.

5.2 DISCOVERPHYSICS PERFORMANCE

In Figure 3 we show the results on the DISCOVERPHYSICS validation set showing the mean normalized MSE, and explanation score as a function of training step, and world pass@k for the final saved checkpoint. We treat DISCOVERPHYSICS as an in-distribution test set for our RL training.

We show the RL trained Qwen3-4B model in blue, with the performance of Claude Haiku 4.5, Sonnet 4.6, and Opus 4.8 shown in the varied dashed lines. These results are taken from the public leaderboard. Initially the Qwen model is outperformed by all three Claude family models in both the MSE and explanation scores, but after training the performance improves significantly in the MSE metric (reducing by 2 orders of magnitude), as well as improving the explanation score to $\sim 0.19 \rightarrow 0.33$, beyond both Haiku (0.23) and Sonnet (0.27), but below the performance of Opus (0.58). The MSE score provides a more direct supervision signal, and this translates into Qwen getting closer to Opus in terms of MSE whilst lagging significantly behind on explanation score.

Figure 6 shows the per-world performance of the Qwen model both pre- (white circle) and post-RL training (blue circle), with the left column showing the trajectory normalised MSE, and the right the explanation score (averaged over 5 seeds for all worlds). We see a uniform improvement to the normalised MSE for every world, and see in all but one world (private 2) an improvement to the explanation score.

Error analysis. In Figure 4 we show a failure mode analysis for the base model (white) and RL trained model (blue). As in the DiscoverPhysics benchmark (Wiemann et al., 2026), we present an LLM-mediated (Claude Fable 5) summarise-then-classify pipeline for tagging model errors. A first pass compresses the multi-thousand-line run transcript into a short structured per-run summary; a second pass reads that summary and assigns the run to multiple capability gaps from a predefined list comprising of three broad categories: coding, experimenting and modelling. Detailed definitions of the categories can be found in Appendix H.

On one hand, we see that RL drastically improves the LLMs ability to run experiments and write code that not only runs but also faithfully represents its own hypothesis: runtime crashes fall from 72% to 10% of attempts, code that contradicts the model's stated law from 45% to 6%, uninformative

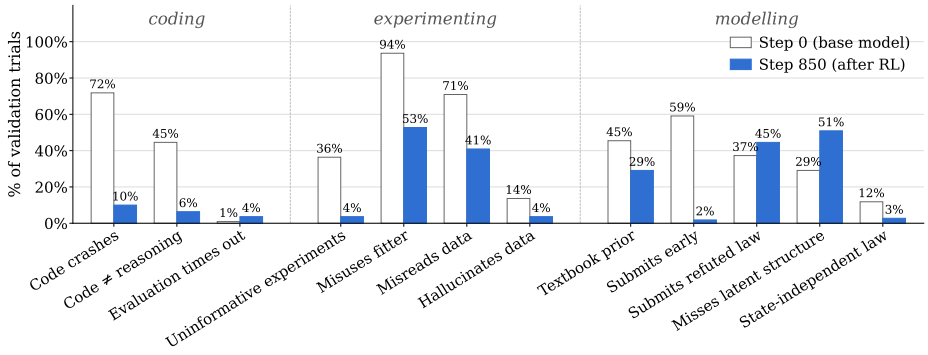

Figure 4: Failure-mode breakdown of the 110 DISCOVERPHYSICS validation trials before (step 0) and after (step 850). RL training removes most runtime crashes and wrong force laws, shifting the dominant failure to laws that reproduce the trajectory but whose explanation misses the underlying latent structure or is refuted by its own experiments.

experiments from 36% to 4%, and premature submissions from 59% to 2%. Interestingly, reliance on textbook priors when the data point elsewhere also drops, from 45% to less than 30% of the attempts. The RL model is now more prone to submitting a law that was refuted by its own experiments and missing latent structure. This is largely a conditioning effect: the base model rarely gets to a runnable law for these failure modes to apply to.

5.3 BENCHMARK TRANSFER

We perform a series of evaluation of our Qwen model both before and after RL training on a set of scientific discovery, and long-horizon planning benchmarks summarizing the results in Figure 5. We evaluate transfer on four benchmarks of decreasing similarity to our training task: NewtonBench (Zheng et al., 2026), which tests recovery of altered canonical laws in an interactive setting; GravityBench (Koblischke et al., 2025), which asks an agent to infer properties of a two-body gravitational system from a budget of self-chosen observations; HeroBench (Anokhin et al., 2025), a non-scientific long-horizon planning task in a virtual world; and LiveCodeBench (Jain et al., 2024), a coding benchmark we include as a check that general programming ability is not degraded.

We see that learning in our auto-generated worlds does lead to significant improvements on NewtonBench and HeroBench, whereas GravityBench is slightly degraded. Inspecting the GravityBench transcripts, the largest single cause is a data-access habit acquired during training. Our setup returns a handful of measurements per round, and the policy learned to print and reason about them directly. We see that on GravityBench the model uses a df.head() call in most runs, and uses the return to compute quantities from the first few rows of data, missing a large portion of the full dataset. Adding one sentence to the prompt instructing the model to use the full column removes this behaviour and lifts the policy above the base model (Figure 5), but not on every task (Figure 9). We analyse these cases in Appendix I.

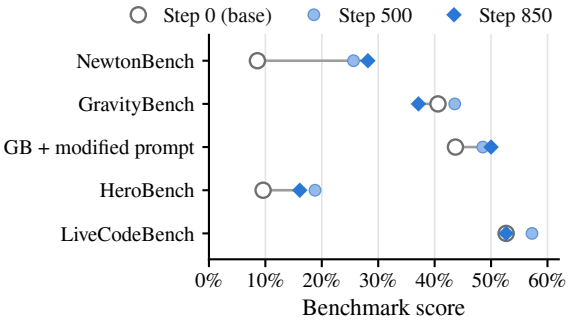

Figure 5: Benchmark scores before RL (step 0) and after 500 and 850 steps of GRPO. The modified GravityBench prompt states the data table's row count and asks for computation over the full column.

6 RELATED WORK

Agent based scientific discovery Agentic based scientific discovery is a rapidly expanding area. SR-Scientist from Xia et al. (2026) is the closest parallel to this work. The authors present a science agent trained via GRPO that writes code to analyze data, evaluate candidate equations, and iteratively refine them over long horizons. On the LSR-Synth benchmark this yields a $6 \rightarrow 35\%$

improvement over their LLM-SR and LaSR baselines. However, their RL training problems are LLM-synthesized equation skeletons over fixed variable ranges, whereas we procedurally generate full interactive physics worlds with graded difficulty and evaluate transfer to held-out discovery benchmarks.

Prabhudesai et al. (2026) present SIM2REASON, which uses a domain-specific-language to compose MuJoCo scenes, converts the simulated traces into verifiable question–answer pairs, and post-trains a series of models from 3 to 32B parameters with RL with verifiable rewards. They find zero-shot gains of 5 to 10 points on the International Physics Olympiad. However, their generated worlds serve as a static QA supervision source for single-turn physics problem solving, whereas we train agents that interact with the generated worlds directly and must discover their governing laws rather than compute answers under known ones.

Auto-generated RL environments Rather than hand-designing an RL environment, unsupervised environment design (UED) treats the environment as something to be generated or curated for the learner (Dennis et al., 2020). Dennis et al. (2020) trains an environment to maximize the regret between a protagonist and antagonist agent, resulting in solvable environments of increasing complexity. Jiang et al. (2021) obtains similar results without a learned generator by replaying randomly sampled levels in proportion to their estimated learning potential. Parker-Holder et al. (2022) combines the two by evolving high-regret levels through small random edits to maximise learning potential of the environments.

7 DISCUSSION AND CONCLUSION

Think longer, collect data The general trends seen from Figure 2 and Figure 3 are that the policy π_θ learns to think longer, gather more data, and test code implementations more often (via increased MSE fitting), which results in improvements on the iterative discovery benchmark DISCOVERPHYSICS. This is generally in line with how a scientist in practice would operate, especially in a situation when probing the environment for more data is free.

Potential for large-scale RL training We have demonstrated that using current frontier LLMs to self-generate novel physics environments through the DISCOVERPHYSICS protocol provides an adequate dataset to use for RL training to improve the scientific discovery ability of open weight LLM models (specifically Qwen3-4B-thinking), with the ability to easily generate additional training data samples. From subsection 5.2 we saw an improvement from initially sub-Haiku level performance, to a model that exceeds Claude Sonnet 4.6 in terms of both mean trajectory MSE, and mean explanation score on the DISCOVERPHYSICS benchmark (however trails in pass@5, from panel 3 of Figure 3). It is an open question the level of performance gains one may see when performing a similar style of RL training on a more powerful base model.

Future work The success of training a smaller 4B parameter model leaves the open question regarding the possible improvement to the scientific reasoning and active experimentation abilities of large proprietary models. Our method for auto-generated worlds is *offline*, to the extent that the worlds are generated at different difficulty levels a priori and then simply fed to the policy all at once. Instead, one may use an LLM in-the-loop to generate novel worlds based on the current capability of the model, potentially improving training, as well as the final model performance, specifically for stronger base models which may require *very hard* worlds at late training times.

Conclusion We introduced DISCOVER-MANY-WORLDS, an automated pipeline that uses frontier LLMs to write, validate, and calibrate the difficulty of novel physical worlds for RL. We performed RL on Qwen-3-4B-Thinking on a pool of 127 auto-generated worlds based on the DISCOVERPHYSICS n-body simulator style of physics environments. The RL training improved the scientific discovery ability of the Qwen model as measured by improvements on both trajectory (curve-fitting) and explanations of the novel physics worlds in the DISCOVERPHYSICS benchmark. The trained policy runs more experiments, reasons longer (as measured by token usage), and utilized the in-round MSE fitting tool more frequently. We believe our methodology provides a scalable way of generating grounded validated environments for scientific discovery without a human in the loop. The difficulty of the generated worlds can scale arbitrarily with the capabilities of the generating model, which in turn would enable us to train potentially superhuman agents for scientific discovery.

AI USE STATEMENT

In this work, we used generative AI tools for software development. We have reviewed all AI-assisted work, with all AI generated code being reviewed before ran, with unit testing implemented where useful. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI. We used to aid or polish writing (proofreading and editing). We used AI tools for retrieval and discovery (e.g., finding related work). We used AI to execute our research ideas by using agents to create the worlds as described in the text. These worlds were generated for our synthetic datasets.

REPRODUCIBILITY STATEMENT

We provide the details necessary for reproducing the results presented here in various appendices. The code anda data used will be released upon acceptance.

ETHICS STATEMENT

We acknowledge the ICLR code of ethics and do not expect our work to have any negative ethical implications.

REFERENCES

Petr Anokhin, Roman Khalikov, Stefan Rebrikov, Viktor Volkov, Artyom Sorokin, and Vincent Bissonnette. Herobench: A benchmark for long-horizon planning and structured reasoning in virtual worlds. *arXiv preprint arXiv:2508.12782*, 2025.

Yimeng Chen, Piotr Piękos, Mateusz Ostaszewski, Firas Laakom, and Jürgen Schmidhuber. Physgym: Benchmarking llms in interactive physics discovery with controlled priors. *Advances in Neural Information Processing Systems*, 38, 2026.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving, 2025. URL https://arxiv.org/abs/2502.00212.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015. URL https://arxiv.org/abs/1412.6572.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pp. 4940–4950. PMLR, 2021.

Nolan Koblischke, Hyunseok Jang, Kristen Menou, and Mohamad Ali-Dib. Gravity-bench-v1: A benchmark on gravitational physics discovery for agents. *arXiv preprint arXiv:2501.18411*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pp. 17473–17498. PMLR, 2022.

Mihir Prabhudesai, Aryan Satpathy, Yangmin Li, Zheyang Qin, Nikash Bhardwaj, Amir Zadeh, Chuan Li, Katerina Fragkiadaki, and Deepak Pathak. Solving physics olympiad via reinforcement learning on physics simulators. *arXiv preprint arXiv:2604.11805*, 2026.

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.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play, 2018. URL `https://arxiv.org/abs/1703.05407`.

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. *arXiv preprint arXiv:2605.26087*, 2026.

Shijie Xia, Yuhan Sun, and Pengfei Liu. Sr-scientist: Scientific equation discovery with agentic ai. In *International Conference on Learning Representations*, volume 2026, pp. 75787–75811, 2026.

Tianshi Zheng, Kiu Wai Tam, Kim Hue Nam Nguyen, Baixuan Xu, Zhaowei Wang, Cheng Jiayang, Hong Ting Tsang, Weiqi Wang, Jiaxin Bai, Tianqing Fang, et al. Newtonbench: Benchmarking generalizable scientific law discovery in llm agents. In *International Conference on Learning Representations*, volume 2026, pp. 99018–99088, 2026.

A Summary of world types created

We summarize the distribution of difficulties of worlds created for RL, their creation method, and models used in Table 1.

Table 1: The 127 worlds used for RL training.

Tier	#	How it was created	Generator	Accepted if
Easy	78	fixed grid of parameters for three of the simulator's force laws (Sec. 3.1)	none	the simulator passes analytic physics tests
Medium	22	LLM generated, then refined until a reference agent's score is in the target band (Sec. 3.2)	Claude Opus 5 (20), Opus 4.8 (2)	median Claude Haiku 4.5 explanation score in $[0.4, 0.7]$; passes the robustness screen
Hard	27	LLM generated around one kind of hidden structure (Sec. 3.2)	Claude Fable 5	passes the reimplementation and adversarial tests (27 of 30)

B RL policy details

The gradient update step is the policy gradient

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{\hat{A}_i}{|\mathcal{M}_i|}\sum_{t\in\mathcal{M}_i}\nabla_\theta \log \pi_\theta(o_{i,t} \mid o_{i,<t})\right], \tag{3}$$

where $\mathcal{M}_i$ is the set of tokens generated by the policy in episode i; tokens returned by the environment are excluded.

C PER WORLD VALIDATION PERFORMANCE IN THE DISCOVERPHYSICS BENCHMARK

Figure 6 shows the DiscoverPhysics benchmark per world normalised MSE and explanation score for the model before and after RL.

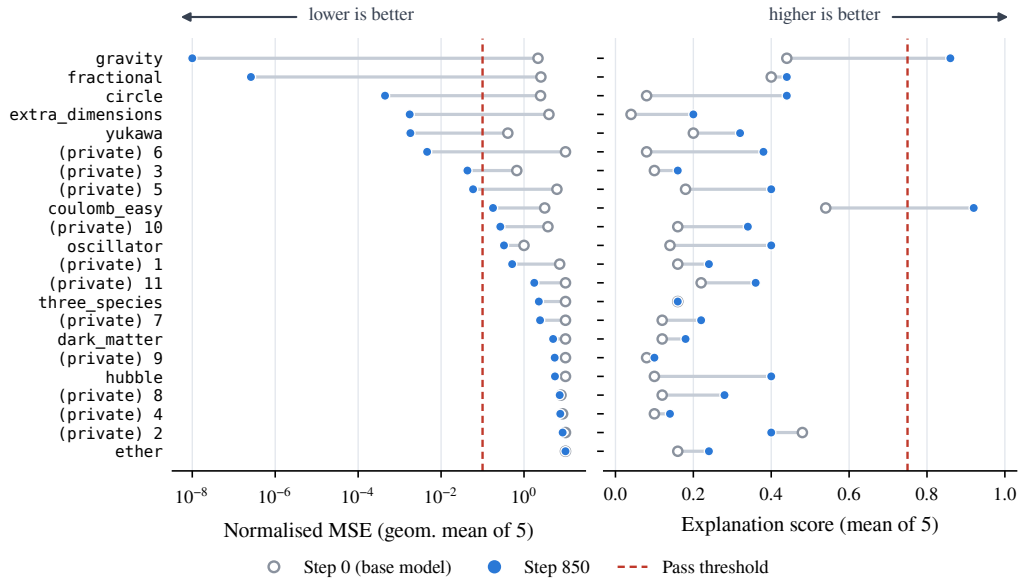

Figure 6: Per-world validation performance on the 22-world DISCOVERPHYSICS benchmark before RL training (step 0, hollow white markers) and after it (step 850, filled blue markers). **Left:** normalised trajectory MSE (lower is better). **Right:** LLM-judged explanation score (higher is better), with dashed lines marking the pass thresholds (normalised MSE < 0.1, explanation score ≥ 0.75), worlds ordered by final normalised MSE, and held-out private worlds anonymised. All results averaged over 5 seeds.

D ADVERSARIAL MSE

Figure 7 shows an example of adversarial experiments.

E EASY WORLD PARAMETER VARIATION AND EXAMPLE

The full list of parameter variation we use to create the easy worlds from our template is shown in Table 2

Our easy world template consisted of a Python dictionary shown in Figure 8.

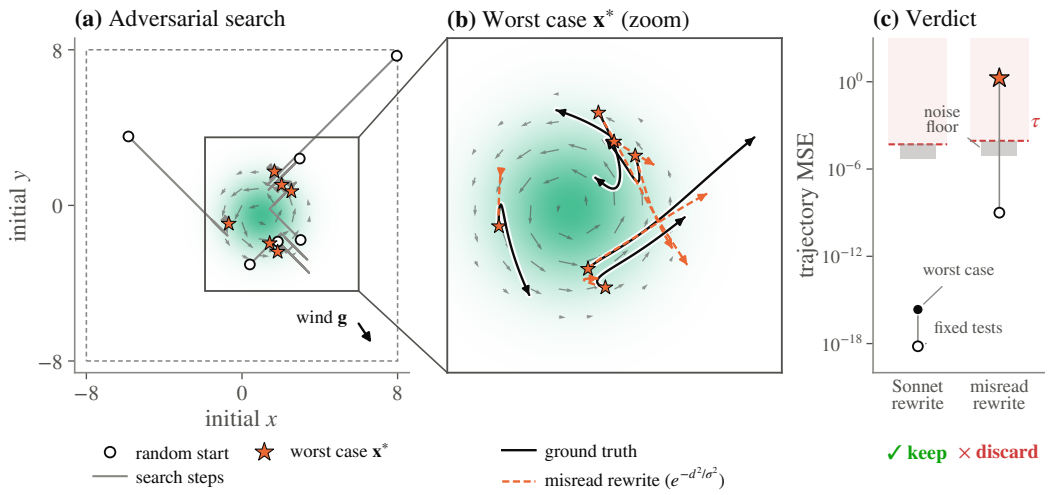

Figure 7: **Adversarial MSE test on the hard world *whirlpool in the wind*.** Six non-interacting particles are pushed by a hidden uniform wind $\mathbf{g}$ and a hidden Gaussian vortex (green shading; arrows show the azimuthal force $A\, e^{-|\mathbf{r}-\mathbf{c}|^2/2\sigma^2}\, \hat{z} \times (\mathbf{r}-\mathbf{c})$). To illustrate a failure, we adversarially attack a hypothetical reimplementation that reads the vortex's "Gaussian width σ" as e^{-d^2/σ^2}. **(a)** The search moves the particles' initial conditions (only initial positions shown; velocities and the measurement time are also searched) from a random start (open circles) into the vortex, ending at the worst case $\mathbf{x}^*$ (stars). Gray lines are search steps: each projected sign-gradient step shifts every coordinate by the same $\pm\alpha_k$, so the steps run diagonally and shrink. Dashed box: search bounds. **(b)** Zoom on the boxed region of (a): trajectories launched from $\mathbf{x}^*$ (stars). The search sends five of the six particles clockwise, against the swirl: the true law (black) turns all of them around, while the re-implementation (orange), weaker at these distances, turns only two. **(c)** On the fixed test cases, whose particles never reach the vortex, both rewrites look perfect (open circles). At the worst case found (filled), the actual Sonnet 4.6 reimplementation stays at 2×10^{-16}, far below $\tau = \max(10 \times \text{noise floor}, 10^{-8})$, and the world is kept as a robust world; the reimplementation reaches 1.8 and would be discarded. The noise floor (gray) is how much the ground truth trajectory itself changes when the initial conditions are perturbed by 10^{-4} of their range.

Table 2: Construction of the evaluation worlds by group.

Group	Construction	Number of worlds
Two-body canonical	$n \in \{-2,-1,0,1\} \times C \in \{0.5,1,2,4\}$, attractive, *except* $(n=-2,\ C=1)$	15
Two-body repulsive	$n \in \{-2,-1\} \times C \in \{1,2\}$, sign $=-1$	4
Other power laws	$n \in \{-3,2,-1.5\} \times C \in \{1,2\}$	6
Pinned source	$n \in \{-2,-1\} \times C \in \{1,2\}$, `charge_mode` $=$ `pinned`	4
Exotic kernels	Plummer $a \in \{0.5,1\}$; Cornell $\sigma \in \{0.1,0.2\}$	4
3D canonical	$n \in \{-2,-1,1\} \times C \in \{1,2\}$, dim $=3$	6
Small-N central	unit-mode mass patterns $N \in \{3,4,5,6,8,12\}$ (6); mass-mode $1/r^2$ $N \in \{3,4,5\}$ (3); mass-mode $1/r$ $N \in \{6,8\}$ (2); unit $1/r$ at varied C (3); Hooke clusters $N \in \{3,4,5,6\}$ (4)	18
Multi-species	16 explicit entries, $N \in \{2,\ldots,8\}$, 2–4 species, including three signed ($\pm$) pairs	16
External field	2 two-body + 3 full-state	5

```
spec = {
  # the pairwise interaction
  "force": {"kernel": "power_law_force",   # cornell_force | plummer_force
            "params": {"C": 1.0, "n": -2.0},
            "sign":   +1},                 # +1 attractive, -1 repulsive

  # who couples to whom
  "charges": {"mode": "mass"},             # mass | pinned | unit | species

  # the experimental interface
  "topology":           "two_body",        # two_body | full_state
  "spatial_dimensions":  2,                # 2 or 3

  # topology-dependent
  "n_particles":    6,                     # full_state
  "masses":         [1.0, 1.0, 2.0, ...],  # full_state, hidden from agent
  "n_probes":       5,                     # probes
  "geometry":       {"shape": "ring", "count": 6, "radius": 5.0},

  # optional
  "external_field": {"mode": "uniform", "params": {"alpha": 0.1}},
  "softening":      0.05,

  # which physics validity checks apply to this world
  "checks": ["faithfulness", "energy", "linear_momentum", ...],
}
```

Figure 8: Example easy world specification.

F PROMPTS FOR GENERATOR AGENT FOR MEDIUM- AND HARD-DIFFICULTY WORLDS

This appendix lists every prompt used to generate the 49 medium and hard worlds of the RL run. The easy worlds involve no LLM (Appendix E).

MEDIUM WORLDS: INITIAL BATCH

Claude Opus 4.8 generated 80 worlds, 16 with each of five prompt variants. Refinement Strategy 2 later started from 13 of these worlds to produce 13 of the medium training worlds; those 13 came from three of the variants: variant 1 (3 worlds), variant 2 (5) and variant 3 (5).

Initial Batch, Variant 1 (Two-Particle): System Prompt

You are an expert theoretical physicist designing novel 2D physics universes for an AI discovery benchmark. Each universe has exactly two particles interacting through a pairwise force in 2D space.

Particle 1 is always fixed at the origin $[0, 0]$. Particle 2 is mobile and has initial position `pos2` and velocity `velocity2`. Each particle has a scalar property: `p1` (source coupling) and `p2` (inertia).

The force on particle 2 depends on:

- The separation vector $\mathbf{r} =$ `pos2` $-$ `pos1`
- The distance $|\mathbf{r}|$
- The particle properties `p1` and `p2`

Your goal: design a NOVEL force law that is:

1. Physically interesting (not trivially simple like constant force)
2. Well-behaved (no singularities that blow up, bounded trajectories for reasonable initial conditions)
3. Clearly describable in prose (a physicist reading your explanation should be able to implement it)
4. Different from the examples shown

The implementation must use the function signature:

```
discovered_law(pos1, pos2, p1, p2, velocity2, duration)
               -> (final_pos2, final_vel2)
```

where `final_pos2` is $[x, y]$ and `final_vel2` is $[v_x, v_y]$ after simulating for `duration` time units. You **MUST** integrate the equations of motion (e.g. using `scipy.integrate.solve_ivp` or a leapfrog integrator) — do **NOT** return analytical solutions unless the ODE is trivially solvable.

Your implementation should be self-contained (all imports inside the function) and use `numpy`/`scipy` only.

NUMERICAL FAITHFULNESS REQUIREMENTS

Your `<law>` code is not an approximation of your world — it **IS** the world's ground truth. Discovery agents are scored on reproducing its outputs from independent reimplementations of the physics you describe. Any behavior that depends on **YOUR** integrator rather than on the stated physics is an artifact no agent can recover, and it invalidates the world. Design accordingly:

1. *Regularize singular forces.* If any force grows without bound as separations shrink ($1/r$, $1/r^2$, ...), allow for an explicit softening length in the law — e.g. the rubric should consider r^2 with $(r^2 + \epsilon^2)$ for small r values — and choose ϵ large enough that the softened core is fully resolved by your time step. State the softening form and the numerical value of ϵ in `<true_law>` and `<explanation>`: it is part of the physics of your world, not a hidden implementation detail.
2. *No unresolved close encounters.* Choose initial configurations, masses, and couplings so that particles do not fall into sources (or each other) faster than your integrator resolves. Cold starts collapsing onto attractors, and dense clumps of mutually attracting particles, are the classic failures.
3. *Match dt to the fastest timescale.* Estimate the shortest dynamical time anywhere in the reachable state space (strongest force, lightest particle, softened minimum separation) and use a fixed step at least $\sim$20$\times$ shorter, or an adaptive solver with tight tolerances (`rtol`/`atol` $\leq 10^{-8}$).
4. *Be integrator-robust, not chaotic.* Over the region agents probe (durations up to $\sim$10 time units, initial positions out to $|x| \sim 10$), a physicist re-implementing your stated equations with a **DIFFERENT** reasonable integrator (RK45 vs. leapfrog, or half your dt) must reproduce your trajectories

far more closely than the scoring tolerance. If nearby initial conditions diverge exponentially (three-body scrambles, chaotic clumping), no agent can ever match your world — redesign (more softening, weaker coupling, bounded potential) until regular dynamics dominate the probed region.

Before finalizing, self-check: agents run experiments **YOU** do not choose — clustered particles, cold collapses, high-speed flybys, long horizons. Mentally run 2–3 such adversarial experiments against your law. If any would hit an unresolved singularity or depend on step size, revise the world. Summarize this reasoning in a `<numerics_check>` tag (3–6 sentences: regularization used, dt justification, why the probed region is non-chaotic), placed before `<rubric>`.

DIFFICULTY TARGET: MEDIUM

Aim the world at a mid-tier discovery agent with a budget of roughly 16 experiment rounds. Calibrate accordingly:

1. *Not trivial.* The governing law should not be readable off a single observation or a one-parameter fit; recovering it should take several well-chosen experiments.
2. *Not inscrutable.* Within $\sim$16 rounds of reasonable experiments the law's form and parameters must be pinned down. Effects that only show up under an exhaustive scan of initial conditions, exotic experiment designs, or measurement precision finer than the scoring tolerance push the world out of reach — keep the signal robust and reachable.
3. Perhaps include a hidden mechanism or latent structure — one that leaves clear fingerprints in trajectories an agent will plausibly run.

Before finalizing, self-check: sketch the shortest experiment sequence that pins down your law, and confirm it fits the $\sim$16-round budget while still taking real work. Summarize this in a `<difficulty_check>` tag (2–3 sentences), placed before `<rubric>`.

Initial Batch, Variant 1: User Prompt

Here are examples of existing force laws in the benchmark (**DO NOT** copy these):

1. **2D Gravity (Laplacian/Poisson):** $F = \dfrac{p_1 p_2}{2\pi r}$ directed toward particle 1. The 2D Green's function is logarithmic, so force falls as $1/r$.
2. **Yukawa (Screened):** Like gravity but with exponential screening at long range. $F \sim p_1 p_2\, K_1(r/\lambda)/\lambda$, decays exponentially beyond $r \sim \lambda$.
3. **Coulomb (Inverse-square):** $F = k\, p_1 p_2 / r^2$, the standard 3D Coulomb law transplanted into 2D space.
4. **Fractional Laplacian:** $F \sim p_1 p_2 / r^{3-2\alpha}$, a non-local interaction with tunable decay exponent $\alpha \in (0, 1)$.
5. **Time-modulated gravity:** Standard 2D Poisson force multiplied by $G(t) = G_0 \cos(\omega t)$, so the interaction periodically changes sign.
6. **Extra dimensions:** Force transitions from $1/r$ (2D Poisson) at large r to $1/r^2$ (3D Newton) at small r due to a compactified extra dimension.

Design a NOVEL force law different from all of the above. Be creative — consider ideas like:

- Velocity-dependent forces (magnetic-like)
- History-dependent / memory forces
- Distance-dependent switching between regimes
- Anisotropic forces
- Multi-scale potentials
- Nonlinear dependence on particle properties
- Forces with angular momentum dependence

Output your proposal in **EXACTLY** this format (use the XML tags precisely):

```
<world_name>short_snake_case_name</world_name>

<description>One-line description of the force law.</description>

<true_law>
```

```
LaTeX formula(s) describing the force law. Use $...$ for inline math.
Include all constants and their values.
</true_law>

<explanation>
A clear prose explanation (3-8 sentences) of the physics. This should be
detailed enough that an expert physicist could implement it from this
description alone. Include:
- The functional form of the force
- How p1 and p2 enter (which is coupling, which is inertia)
- Any characteristic scales or dimensionless parameters
- The qualitative behaviour (what trajectories look like)
</explanation>

<law>
def discovered_law(pos1, pos2, p1, p2, velocity2, duration):
    import numpy as np
    from scipy.integrate import solve_ivp
    # ... your implementation ...
    return final_pos2, final_vel2
</law>

<rubric>
10  -- [criteria for perfect score]
7-9 -- [criteria for good score]
4-6 -- [criteria for partial understanding]
1-3 -- [criteria for poor understanding]
0   -- Empty or irrelevant.
</rubric>
```

Followed by the list of previously created worlds (Prompt 5).

Open-Ended System Prompt (Initial Batch Variant 3; Base of the Refinement-Strategy System Prompt)

You are a physicist designing a self-contained simulated world whose physics differs from our own. The goal is to produce a world that a scientific agent could plausibly discover through experimentation, meaning the physics should be *conceptually distinctive* (it teaches or tests a specific idea), not merely complicated.

Design constraints. The world will run on a single machine in minutes, so the dynamics need to be cheap enough to evolve over modest time horizons at the scale of hundreds of degrees of freedom. Particle systems, lattice or grid-based field theories, mean-field reductions of more complex dynamics, and other formalisms are all in scope: what matters is that you can write code that evolves the world forward in time without excessive compute power. If the natural description of your world is a PDE or a continuum field theory, that's fine, but you are responsible for choosing a discretization or reduction that fits the compute budget while preserving the conceptual physical system you care about.

Conceptual hook. The most interesting worlds isolate a single non-trivial idea that an observer would have to recognize rather than just curve-fit: short-range screening, non-integer power laws, latent particle species coupled differently to each other, hidden sources that exert force but cannot be observed, time-modulated or position-modulated couplings, preferred directions or frames, dimensional reductions where higher-dimensional structure leaks into observable dynamics, etc. Pick one such hook, that is distinct from the ones mentioned here, and design the world around it. The hook should be inferable from sufficiently clever experiments but not obvious from a casual look at trajectories.

Your output.

1. *World description.* A short natural-language explanation of the physics of this world: what degrees of freedom exist, what their states mean, what governs their evolution and interactions, and what the conceptual hook is. Be precise enough that another physicist could reimplement it from your description.
2. *Simulator code.* Python that evolves the full state of the world forward in time given initial conditions. You may assume a standard scientific stack (NumPy, SciPy integrators, JAX if useful). The simulator outputs the true state at requested times. Call these the *latent states*.

3. *Instrument model.* A separate function that runs on top of the simulator's latent states to produce the *observed states* an experimenter actually sees. The instrument may: hide entire degrees of freedom (latent variables whose existence must be inferred), expose only a subset of the state (e.g. a field's value but not its gradient, or particle positions but not charges or species labels), project from higher to lower dimensions, coarse-grain over space or time, add measurement noise, or introduce systematics (drifts, dead zones, finite resolution). The instrument is part of the design — choose it so that the conceptual hook is recoverable in principle but requires thoughtful experimental design to surface.

Before writing code, sketch the world's hook and the observer's epistemic situation in a few sentences. Then implement.

Initial Batch, Variant 2 (Open-Ended, Two-Particle Interface): System Prompt

```
You are a physicist designing a self-contained simulated world whose physics differs from
↪  our own. The goal is to produce a world that a scientific agent could plausibly
↪  discover through experimentation, meaning the physics should be *conceptually
↪  distinctive* (it teaches or tests a specific idea), not merely complicated.

*Design constraints.* The world will run on a single machine in minutes, so the dynamics
↪  need to be cheap enough to evolve over modest time horizons at the scale of hundreds
↪  of degrees of freedom. Particle systems, lattice or grid-based field theories,
↪  mean-field reductions of more complex dynamics, and other formalisms are all in scope:
↪  what matters is that you can write code that evolves the world forward in time without
↪  excessive compute power. If the natural description of your world is a PDE or a
↪  continuum field theory, that's fine, but you are responsible for choosing a
↪  discretization or reduction that fits the compute budget while preserving the
↪  conceptual physical system you care about.

*Conceptual hook.* The most interesting worlds isolate a single non-trivial idea that an
↪  observer would have to recognize rather than just curve-fit: short-range screening,
↪  non-integer power laws, latent particle species coupled differently to each other,
↪  hidden sources that exert force but cannot be observed, time-modulated or
↪  position-modulated couplings, preferred directions or frames, dimensional reductions
↪  where higher-dimensional structure leaks into observable dynamics, etc. Pick one such
↪  hook, that is distinct from the ones mentioned here, and design the world around it.
↪  The hook should be inferable from sufficiently clever experiments but not obvious from
↪  a casual look at trajectories.

Before writing code, sketch the world's hook and the observer's epistemic situation in a
↪  few sentences. Then implement.
```

In the initial batch every system prompt (variants 1–3) had the **NUMERICAL FAITHFULNESS REQUIREMENTS** and **DIFFICULTY TARGET: MEDIUM** blocks of Prompt 1 appended, and every user prompt had the list of previously created worlds appended (Prompt 5). The user prompt of variants 2 and 3 consisted of that list only, so these prompts gave no output format: each reply was reformatted by a separate call to the same model (Prompt 6). Variant 3 specifies no interface, so a further call first chose one (Prompt 7).

Initial Batch: Previously Created Worlds (Appended to Every User Prompt)

Every earlier world of the session, $1..N$; `law` is cut to 240 characters.

```
---
PREVIOUSLY CREATED WORLDS — you already designed the worlds below in this session. Design
↪  something GENUINELY DIFFERENT: a distinct physical mechanism / functional form, not a
↪  parameter tweak, rename, or close variant of any of these.
1. {{name}}: {{description}}
   law: {{true_law}}
N. {{name}}: {{description}}
   law: {{true_law}}
---
Now design a NEW world, different from ALL of the above, in the EXACT output format
↪  specified earlier.
```

Initial Batch: Reformatting Call (Variants 2 and 3)

System prompt:

```
You are a formatting specialist. You receive raw LLM output that was supposed
to describe a novel physics world using specific XML tags, but the output is
missing some or all of the required tags.

Your job: restructure the content into the exact XML format shown, preserving
```

```
the physics content as faithfully as possible. Do NOT change the physics —
only reformat.

Rules:
- Output ALL 6 required XML tags, even if you must synthesize a field from
  context (e.g. derive a world_name from the description, or write a rubric
  based on the explanation).
- The <law> tag must contain a complete, self-contained Python function with
  the EXACT signature specified. All imports must be inside the function body.
- If the raw output contains simulator code with a different function name or
  signature, adapt it to match the required signature.
- Do not add markdown fences inside the <law> tag — just raw Python code.
- Preserve all mathematical formulas, constants, and numerical values.
```

User prompt (two-particle interface):

```
        The following LLM output was supposed to contain a physics world
        proposal in XML format, but is missing required tags. Restructure
        it into the exact format shown, preserving the physics faithfully.

        --- RAW OUTPUT ---
        {{generator reply}}
        --- END RAW OUTPUT ---

        Required function signature for the <law> tag:
        def discovered_law(pos1, pos2, p1, p2, velocity2, duration):
    # pos1: position of particle 1 (always at the origin)
    # pos2: initial position of particle 2
    # p1: scalar property of particle 1
    # p2: scalar property of particle 2
    # velocity2: initial velocity of particle 2
    # duration: float, simulate from t=0 to t=duration
    # return: (final_pos2, final_vel2)
    return final_pos2, final_vel2

        Topology: two_particle
        Number of particles: 2
        Spatial dimensions: 2

        Here is a complete example of the expected output format:

            <world_name>example_force</world_name>

    <n_particles>2</n_particles>

    <dimensions>2</dimensions>

    <description>One-line description of the force law.</description>

    <true_law>
    $F = ...$ with constants $k = ...$
    </true_law>

    <explanation>
    3-8 sentences explaining the physics. Include the functional form,
    how particle properties enter, characteristic scales, and qualitative
    trajectory behavior.
    </explanation>

    <law>
    def discovered_law(pos1, pos2, p1, p2, velocity2, duration):
# pos1: position of particle 1 (always at the origin)
# pos2: initial position of particle 2
# p1: scalar property of particle 1
# p2: scalar property of particle 2
# velocity2: initial velocity of particle 2
# duration: float, simulate from t=0 to t=duration
# return: (final_pos2, final_vel2)
return final_pos2, final_vel2
    </law>

    <rubric>
    10 — Perfect reimplementation matching all trajectories.
     7-9 — Correct functional form with minor numerical differences.
     4-6 — Partially correct physics.
     1-3 — Wrong physics.
       0 — Empty or irrelevant.
    </rubric>

        Now restructure the raw output above into this exact format.
        Output ONLY the XML tags, nothing else.
```

Initial Batch: Interface Selection (Variant 3)

System prompt:

```
You are a physics simulation architect. You receive a free-form description
of a novel physics world and must choose the best simulator interface
(topology) for it from a fixed set of options. You also determine the
number of particles and spatial dimensions the world requires.

Choose the topology whose function signature most naturally expresses the
world's physics. Prefer richer topologies when the world needs them —
do not force an N-body system into a two-particle interface.
```

User prompt:

```
        The following is a free-form description of a novel physics world.
        Choose the simulator topology that best fits this world's physics.

        --- WORLD DESCRIPTION ---
        {{generator reply}}
        --- END WORLD DESCRIPTION ---

        Available topologies:

        ### two_particle
Description: One fixed source particle at the origin, one mobile probe particle. Each has
↪  a scalar property (p1, p2). Best for simple pairwise force laws between exactly two
↪  bodies where one is stationary.
Function signature:
```
def discovered_law(pos1, pos2, p1, p2, velocity2, duration):
    # pos1: position of particle 1 (always at the origin)
    # pos2: initial position of particle 2
    # p1: scalar property of particle 1
    # p2: scalar property of particle 2
    # velocity2: initial velocity of particle 2
    # duration: float, simulate from t=0 to t=duration
    # return: (final_pos2, final_vel2)
    return final_pos2, final_vel2
```

### multi_particle
Description: N identical particles, all mobile, interacting via pairwise or collective
↪  forces. No per-particle properties beyond position and velocity. Best for N-body
↪  systems with identical particles, pairwise memory, collective dynamics, or hidden
↪  relational variables.
Function signature:
```
def discovered_law(positions, velocities, duration):
    # positions: list of N [x, y, ...] coords
    # velocities: list of N velocity vectors
    # duration: float, simulate from t=0 to t=duration
    # return: list of N final position vectors
    return final_positions
```

### probes
Description: N particles split into background sources and mobile probes. All evolve
↪  together but scoring focuses on probe particles. Best when there are fixed or heavy
↪  background sources and lighter test particles exploring a field.
Function signature:
```
def discovered_law(positions, velocities, duration):
    # positions: list of N [x, y, ...] coords (background + probes)
    # velocities: list of N velocity vectors
    # duration: float, simulate from t=0 to t=duration
    # return: list of N final position vectors
    # NOTE: scored on the last K (probe) entries
    return final_positions
```

### probes_with_masses
Description: Like probes, but each particle has a per-particle mass that affects the
↪  dynamics. Best when inertia varies across particles and mass is a meaningful dynamical
↪  parameter.
Function signature:
```
def discovered_law(positions, velocities, masses, duration):
    # positions: list of N [x, y, ...] coords
    # velocities: list of N velocity vectors
    # masses: list of N per-particle masses
    # duration: float, simulate from t=0 to t=duration
    # return: list of N final position vectors
```
```

````
    return final_positions
```

### coulomb_hard
Description: N particles with signed scalar charges that determine attraction vs
↪  repulsion. Best when particles carry charge-like properties with sign-dependent
↪  interactions.
Function signature:
```
def discovered_law(positions, velocities, charges, duration):
    # positions: list of N [x, y, ...] coords
    # velocities: list of N velocity vectors
    # charges: list of N signed scalar charges
    # duration: float, simulate from t=0 to t=duration
    # return: list of N final position vectors
    return final_positions
```

        Based on the world description, choose the topology whose function
        signature most naturally expresses this world's physics. Also
        determine the number of particles and spatial dimensions.

        IMPORTANT: Output ONLY the three XML tags below. No explanation,
        no reasoning, no preamble. Just the three tags:

        <topology>topology_name</topology>
        <n_particles>integer</n_particles>
        <dimensions>integer</dimensions>
````

MEDIUM WORLDS: REFINEMENT STRATEGIES

Refinement strategies. Refinement Strategy 1 (Feedback) and Refinement Strategy 2 (Iterative improvement) share the system prompt and output format below; they differ only in what is appended to the user prompt. Text in {{...}} is filled in per attempt.

Refinement Strategies 1 and 2: System Prompt

Prompt 3, followed by the **NUMERICAL FAITHFULNESS REQUIREMENTS** block of Prompt 1, followed by:

```
DIFFICULTY TARGET: MEDIUM

This world is evaluated by a mid-tier discovery agent (Claude Haiku) with a
budget of EXACTLY 16 experiment rounds. Your goal is a world on which that
agent recovers the GENERAL physical picture but misses or mis-fits some
structural or quantitative detail — a partial, incomplete explanation scoring
roughly 0.4-0.7 out of 1.0. Calibrate accordingly:

1. Not trivial. The governing law must not be readable off a single
   observation or a one-parameter fit; pinning it down should take several
   well-chosen experiments.

2. Not inscrutable. Within 16 rounds of reasonable experiments the law's form
   and leading parameters must be reachable. Effects that only surface under
   an exhaustive scan of initial conditions, exotic experiment designs, or
   measurement precision finer than the scoring tolerance push the world out
   of reach — keep the signal robust and reachable.

3. The hidden mechanism or latent structure should leave a clear fingerprint
   in trajectories an agent will plausibly run.

Before finalizing, self-check: sketch the shortest experiment sequence that
pins down your law and confirm it fits the 16-round budget while still taking
real work. Summarize this in a <difficulty_check> tag (2-3 sentences), placed
before <rubric>.


CALIBRATION — MAKE THIS GENUINELY MEDIUM, NOT HARD

Worlds designed under the guidance above have consistently come out FAR TOO
HARD: the evaluation agent recovers almost nothing (near-zero scores). The
single most common cause is over-design — stacking multiple mechanisms, exotic
functional forms, and near-chaotic dynamics into one world. Correct for this:
START FROM AN EASY IDEA -- a simple, easily discoverable mechanism -- and add a
complication on top of it if needed.

- KEEP THE EFFECT FIRST-ORDER VISIBLE. Any complication you add must leave an
  obvious fingerprint in a handful of straightforward experiments — not require
```
```

```
  a fine parameter scan or precision beyond the scoring tolerance.

- PREFER A SMALL SHIFT OVER AN INVENTION. A medium world is usually a familiar
  law with one term or parameter moved into unfamiliar territory -- not a
  brand-new exotic physics.

Final gate: would a competent physicist, given ~16 experiments, most likely get
the GENERAL form right but miss or mis-fit ONE detail? If they would nail it
completely, it is too easy. If they would be lost, it is too hard — strip back
to one visible twist.
```

Refinement Strategies 1 and 2: Output Format (Two-Particle Interface)

Every user prompt begins with the topology's output-format block. For the two_particle topology (all 22 medium training worlds):

```
Output your proposal in EXACTLY this format (use the XML tags precisely):

<world_name>short_snake_case_name</world_name>

<n_particles>integer number of particles</n_particles>

<dimensions>integer spatial dimensions (e.g. 2 or 3)</dimensions>

<description>One-line description of the force law.</description>

<true_law>
LaTeX formula(s) describing the force law. Use $...$ for inline math.
Include all constants and their values.
</true_law>

<explanation>
A clear prose explanation (3-8 sentences) of the physics. This should be
detailed enough that an expert physicist could implement it from this
description alone.
</explanation>

<law>
def discovered_law(pos1, pos2, p1, p2, velocity2, duration):
    # pos1: position of particle 1 (always at the origin)
    # pos2: initial position of particle 2
    # p1: scalar property of particle 1
    # p2: scalar property of particle 2
    # velocity2: initial velocity of particle 2
    # duration: float, simulate from t=0 to t=duration
    # return: (final_pos2, final_vel2)
    return final_pos2, final_vel2
</law>

<rubric>
10 — [criteria for perfect score]
 7-9 — [criteria for good score]
 4-6 — [criteria for partial understanding]
 1-3 — [criteria for poor understanding]
   0 — Empty or irrelevant.
</rubric>

The implementation must integrate the equations of motion (e.g. using
↪  scipy.integrate.solve_ivp or a leapfrog integrator). Your implementation should be
↪  self-contained (all imports inside the function) and use numpy/scipy only.
```

Refinement Strategy 1 (Feedback): User Prompt Additions

Appended after Prompt 9, in this order.
Seed world (fixed):

```
---
SEED WORLD — a fixed, simple jumping-off idea (the SAME easy world is shown for every
↪  generation). This is an EASY world that a weak agent solves almost immediately:

  name:        gravity_2body_2d
  description: 2D two-body world: attractive inverse-square central force, F = C*m1*m2/r^2
  ↪  with C = 1.0; both particles mobile.
  true_law:    $F = -C\, m_1 m_2\, \hat r / r^2$, C = 1.0 (Newtonian inverse-square
  ↪  gravity)
  physics:     Two masses attract along the line joining them with magnitude proportional
  ↪  to 1/r^2; bound pairs trace closed ellipses, unbound pairs hyperbolas. No hidden
  ↪  species, fields, or modulation — the plain inverse-square law.
```

```
Use this only as a starting point. Following the CALIBRATION above, produce a MEDIUM world
↪  by adding a complication on top of a simple idea like this if needed. Do not simply
↪  reproduce the seed unchanged.
---
```

Calibration feedback, from the median score of the recent worlds:
first attempt of a session:

```
CALIBRATION FEEDBACK: This is the first world of the session. Aim squarely for the middle
↪  of the target band (≈0.55).
```

every later attempt:

```
CALIBRATION FEEDBACK: Your recent worlds have been TOO HARD for the evaluation agent
↪  (recent median explanation score {{recent median}}; target 0.4-0.7). Make THIS world
↪  EASIER: lean on the simple idea, weaken or simplify any complication, and make its
↪  fingerprint more obvious within a few experiments.
```

Previously created worlds (every earlier attempt of the session, 1..N; `law` is cut to 240 characters). The first Strategy 1 run used this version, with scores only:

```
---
PREVIOUSLY CREATED WORLDS (this session), with how the evaluation agent scored on each.
↪  Design something GENUINELY DIFFERENT (a distinct physical mechanism / functional form,
↪  not a parameter tweak or rename), and LEARN from the outcomes below — avoid repeating
↪  designs marked "too hard":
1. {{name}}: {{description}}
   law: {{true_law}}
   evaluation-agent score: {{median}}  [{{too hard | too easy | on target}}]
N. {{name}}: {{description}}
   law: {{true_law}}
   evaluation-agent score: {{median}}  [{{too hard | too easy | on target}}]
---
Now design a NEW world, different from ALL of the above, in the EXACT output format
↪  specified earlier.
```

The second Strategy 1 run used this version, which adds the judge's critique for the 10 most recent worlds:

```
---
PREVIOUSLY CREATED WORLDS (this session), with how the evaluation agent scored on each.
↪  Design something GENUINELY DIFFERENT (a distinct physical mechanism / functional form,
↪  not a parameter tweak or rename). The most recent worlds also include the evaluation
↪  agent's OWN feedback (what it recovered vs. what it missed) — READ it and use it to
↪  course-correct so this world lands in the target band, and avoid repeating designs
↪  marked "too hard":
1. {{name}}: {{description}}
   law: {{true_law}}
   evaluation-agent score: {{median}}  [{{too hard | too easy | on target}}]
N. {{name}}: {{description}}
   law: {{true_law}}
   evaluation-agent score: {{median}}  [{{too hard | too easy | on target}}]
   agent feedback: {{judge critique}}
---
Now design a NEW world, different from ALL of the above, in the EXACT output format
↪  specified earlier.
```

Refinement Strategy 2 (Iterative Improvement): User Prompt Addition

Appended after Prompt 9, in place of the Strategy 1 additions. `{{median}}` is the world's median evaluation score.

```
---
REFINE THE PREVIOUS WORLD (do NOT design an unrelated new world).

Its measured difficulty was explanation score {{median}} (target 0.4-0.7).
This world is TOO HARD for the evaluation agent. SIMPLIFY it: reduce the complexity of the
↪  force law and/or the latent/hidden structure — remove or weaken a term, drop a hidden
↪  species / field / modulation, use an integer exponent, or cut the number of
↪  interacting mechanisms — so the governing law is easier to recover. Make the SMALLEST
↪  change likely to lift the score into the band; do not overshoot into triviality.

Keep the SAME topology and the SAME output format; you may keep or lightly adjust the
↪  name. Change only what is needed to move the measured score into the band. The world
↪  to refine:

  name: {{name}}
  description: {{description}}
  true_law: {{true_law}}
  explanation: {{explanation}}
  evaluation-agent's critique (what it got right / missed): {{judge critique}}
  reference law — the world's ground-truth code, refine THIS:
    {{reference-law code}}
```

```
---
Now output the REFINED world in the EXACT output format specified earlier.
```

If the median is above the band, the `TOO HARD` paragraph is replaced by:

```
This world is TOO EASY. ADD DIFFICULTY: introduce ONE modest extra structure (a mild
↪  latent coupling, a small non-integer exponent, a weak hidden influence) so recovery
↪  takes more work — without making it unsolvable.
```

HARD WORLDS

Hard Worlds: System Prompt (Claude Fable 5)

```
You are a physicist designing a self-contained simulated world for an AI discovery
↪  benchmark. The defining feature of your world is HIDDEN LATENT STRUCTURE: physical
↪  structure that shapes the dynamics but is never exposed through the observable
↪  interface. An experimenter can only set the initial positions and velocities of the N
↪  visible particles and observe their resulting trajectories; everything else about the
↪  world must be inferred from how those trajectories bend.

*The hidden latent.* Choose exactly ONE kind of hidden structure and design the world
↪  around it. Examples of the kind of thing we mean (pick something in this spirit, not
↪  necessarily from this list):
- Hidden static sources: K fixed invisible bodies at specific positions that exert force
↪  on the visible particles but are never reported and cannot be moved.
- Hidden per-particle properties: the visible particles carry unequal masses, charges, or
↪  species labels assigned by particle index, but the labels are never exposed — only
↪  positions and velocities are.
- A hidden background field: a fixed spatial potential, lattice, or preferred direction
↪  that modulates the interaction.
- Hidden modulation: an interaction constant that varies with position or time according
↪  to a fixed rule.

*Hard requirements.*
1. The hidden structure must be HARD-CODED inside discovered_law as concrete numerical
↪  constants (source positions, per-particle property arrays, field parameters). The
↪  function signature exposes only (positions, velocities, duration) — nothing else.
↪  Anything not in the signature is hidden by construction; do NOT add extra arguments.
2. State every hidden numerical value EXACTLY in <true_law> and <explanation> (e.g. "two
↪  hidden sources of strength 4.0 at [2.0, -1.0] and [-3.0, 2.5]"). A physicist reading
↪  the explanation alone must be able to reimplement the world and reproduce its
↪  trajectories to near-zero error. An explanation that says "there are hidden
↪  attractors" without their count, positions, and strengths is a failure.
3. Identifiability: the latent must be recoverable in principle from trajectory
↪  experiments alone. Avoid degenerate designs where different latents produce
↪  indistinguishable dynamics in the reachable region (e.g. two hidden sources that act
↪  like a single heavier source at their centroid from everywhere the probes can reach,
↪  or a hidden uniform force indistinguishable from rescaled inertia). Ask yourself: what
↪  experiment would distinguish my latent from the simplest rival hypothesis? If none
↪  exists, redesign.
4. The world will be probed with particles placed in roughly [-5, 5]^d with speeds up to
↪  ~0.5, observed at times up to ~5. The hidden structure must produce clearly measurable
↪  effects in that region — hidden sources far outside it, or effects much smaller than
↪  typical displacements, are undiscoverable.
5. Well-behaved dynamics: no singular blow-ups for generic initial conditions in the probe
↪  region (soften potentials near hidden sources, e.g. r -> sqrt(r^2 + eps^2)), bounded
↪  or slowly-escaping trajectories, and dynamics smooth enough that trajectories are
↪  reproducible under tiny perturbations of the initial conditions — prefer STATIC
↪  hidden structure over hidden mobile bodies, and avoid strongly chaotic configurations.
6. Keep it cheap: the implementation must integrate N ~ 4-10 particles over durations ~5
↪  in well under a second.

If the task message assigns you a specific HIDDEN-LATENT FAMILY, you MUST design your
↪  world within that family, even where it refines the guidance above — the family fixes
↪  the kind of hidden structure; the specific design within it is yours.

Before writing code, state in a few sentences: the hidden latent you chose, why it is
↪  identifiable, and the experiment an ideal scientist would run to expose it.
```

Hard Worlds: User Prompt

```
Your world must be implementable as an N-body particle simulation. The implementation must integrate the equations of motion for all N visible particles simultaneously (e.g. using scipy.integrate.solve_ivp or a leapfrog integrator). Your implementation should be self-contained (all imports inside the function) and use numpy/scipy only. Hard-code the hidden structure as literal numerical constants inside the function body.

Output your proposal in EXACTLY this format (use the XML tags precisely):

<world_name>short_snake_case_name</world_name>

<n_particles>integer number of visible particles (e.g. 6)</n_particles>

<dimensions>integer spatial dimensions (e.g. 2)</dimensions>

<description>One-line description of the physics, naming the hidden structure.</description>

<true_law>
LaTeX or precise mathematical description of the force law / evolution rule.
Include all constants and ALL hidden values (source positions, strengths,
per-particle properties) with their exact numbers.
</true_law>

<explanation>
A clear prose explanation (3-8 sentences) of the physics. Detailed enough that
an expert physicist could reimplement it from this description alone. Include:
- The functional form of the forces or evolution rules
- The hidden structure: what it is, and every hidden numerical value exactly
  (counts, positions, strengths, per-particle assignments)
- Any characteristic scales, dimensionless parameters, or constants
- The qualitative behaviour (what trajectories look like, and how the hidden
  structure betrays itself)
</explanation>

<law>
def discovered_law(positions, velocities, duration):
    import numpy as np
    from scipy.integrate import solve_ivp
    # positions: list of N position vectors, each [x, y, ...] (list of floats)
    # velocities: list of N velocity vectors, each [vx, vy, ...] (list of floats)
    # duration: float, simulate from t=0 to t=duration
    # return: list of N final position vectors (same shape as input positions)
    ...
    return final_positions
</law>

<rubric>
10 — [reserve for correctly identifying the hidden structure: its kind, count,
      and numerical values, plus the functional form of the interaction]
 7-9 — [correct kind of hidden structure and functional form, values partially
       or approximately recovered]
 4-6 — [detected that hidden structure exists and characterized its effect,
       but misidentified its kind or values]
 1-3 — [fit the visible dynamics without recognizing any hidden structure]
  0 — Empty or irrelevant.
</rubric>
```

Hard Worlds: Per-Seed Suffix and Hidden-Latent Families

Appended to Prompt 13, separated by blank lines. Seed s is pinned to family $((s-1) \bmod 10)+1$ from the list below and is shown every world generated earlier in the batch. The second block is omitted for the first seed, when that list is still empty.

Per-seed suffix:

```
ASSIGNED HIDDEN-LATENT FAMILY (mandatory — design your world within this family):
{{family text}}

ALREADY-GENERATED WORLDS in this batch. Your world MUST differ substantially from ALL of them in both mechanism and observable behaviour (a different functional form, not just different constants):
- {{name}}: {{description}}
- ...
```

The ten hidden-latent families, in seed order:

```
1. Hidden static point sources: K >= 2 fixed invisible bodies at specific positions (a mix of attractive and repulsive, unequal strengths) exert softened central forces on the visible particles.
```

```
2. Hidden per-particle properties: the visible particles interact pairwise, but carry
↪  unequal hidden scalar properties (masses, charges, or coupling strengths) assigned by
↪  particle index and never exposed.
3. Hidden anisotropic background: a fixed global preferred direction or multipole-like
↪  static potential (no point sources) shapes every trajectory.
4. Hidden position-dependent coupling: the strength of the interaction (pairwise or to a
↪  fixed center) varies smoothly with position according to a fixed hidden spatial rule.
5. Hidden time-modulated coupling: an interaction constant varies deterministically in
↪  time with a hidden frequency, phase, or waveform.
6. Hidden periodic lattice: a fixed egg-crate / washboard potential with hidden spacing,
↪  orientation, and depth superposed on otherwise simple dynamics.
7. Hidden velocity-coupled field: a magnetic-like force v x B(x) with a hidden, spatially
↪  non-uniform field B(x); state its exact functional form and constants.
8. Hidden dissipative or gain medium: a localized patch (or smooth gradient) of
↪  velocity-dependent drag with hidden center, size, and strength, superposed on
↪  conservative dynamics.
9. Hidden species-dependent interaction: a pairwise force whose range, sign, or strength
↪  depends on hidden species labels assigned to the particles by index.
10. Hidden composite structure: one weak global background field superposed on one
↪  localized hidden structure; both components must be stated exactly and both must be
↪  identifiable.
```

G EXAMPLE RL ROLLOUT

Below is a complete discovery episode by the RL-trained Qwen3-4B-Thinking policy on a two-particle 2D world with a $1/r$ central force. Each round shows the policy's reasoning (abridged; character counts are for the full reasoning trace), the action it emitted, and the environment's response. The episode uses 7 rounds: 5 experiments, 1 MSE fit, and a final submission. Over the first five rounds the agent isolates the dependence on p_1, r, and p_2 one variable at a time before fitting; the fitted constant $k \approx 0.159 \approx 1/2\pi$ is the 2D Poisson coefficient.

Mission

Two particles in a 2D universe.
Your task: discover the law of motion governing particle 2.

Round 1

Thinking (2,999 characters, abridged):
[. . .] measurement_times=[0, 0.5, 1, 2, 3, 4, 5]
This should give the initial acceleration. If it's gravity, starting from rest, the acceleration should be towards the center, so in this case, towards (0,0), so acceleration in the negative x-direction.
Let's run this first experiment to get some data.
Wait, but I need to be careful with noise. The problem says there's Gaussian observation noise, so maybe multiple measurements at the same time or longer durations help.
Let me start with this first experiment.
So first, run an experiment with initial velocity zero, small initial separation, to see the initial acceleration direction and magnitude.
Let me write down the first experiment.

Action:

```
<run_experiment>
[
  {
    "p1": 1.0,
    "p2": 1.0,
    "pos2": [5.0, 0.0],
    "velocity2": [0.0, 0.0],
    "measurement_times": [0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0]
  }
]
</run_experiment>
```

Environment (<experiment_output>, abridged):

```
[{"measurement_times":[0.0,0.5,...,5.0],
 "pos1":[[0.0,0.0],[0.0,0.0],...,[0.0,0.0]],
 "pos2":[[5.0,0.0],[4.996,0.0],...,[4.5966,0.0]],
 "velocity1":[[0.0,0.0],[0.0,0.0],...,[0.0,0.0]],
 "velocity2":[[0.0,0.0],[-0.0159,0.0],...,[-0.1636,0.0]]}]
```

Round 2

Thinking (4,105 characters, abridged):
[. . .] experiment: started at 5 units. Let's try starting at 10 units.
Third experiment: p1=1, p2=1, pos2=[10,0], velocity2=[0,0], measurement_times=[0,0.5,. . . ,5]
If it's inverse square, then at 10 units, acceleration should be (5/10)^2 = 1/4 of the acceleration at 5 units.
So if at 5 units, initial acceleration was ~0.0318, then at 10 units, it should be ~0.00795.
Let me run that.
Wait, but first, let's get more data points. Let's start with the second experiment to test p1 dependence.
Wait, the first experiment had p1=1, so if I change p1 to 2, and see if acceleration doubles.
Let me write down the second experiment.
Second experiment: p1=2, same initial position and velocity.
So:

Action:

```
<run_experiment>
[
  {
    "p1": 2.0,
    "p2": 1.0,
    "pos2": [5.0, 0.0],
    "velocity2": [0.0, 0.0],
    "measurement_times": [0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0]
  }
]
</run_experiment>
```

Environment (<experiment_output>, abridged):

```
[{"measurement_times":[0.0,0.5,...,5.0],
 "pos1":[[0.0,0.0],[0.0,0.0],...,[0.0,0.0]],
 "pos2":[[5.0,0.0],[4.992,0.0],...,[4.1814,0.0]],
 "velocity1":[[0.0,0.0],[0.0,0.0],...,[0.0,0.0]],
 "velocity2":[[0.0,0.0],[-0.0318,0.0],...,[-0.3374,0.0]]}]
```

Round 3

Thinking (1,992 characters, abridged):
[. . .] to check how the acceleration depends on distance. In gravity, acceleration should be proportional to $1/r^2$, so if you double the distance, acceleration should drop to a quarter.
Let me try an experiment with a larger initial position. Let's start particle 2 further away, say at (10, 0), same p1=1, initial velocity zero.
So third experiment: p1=1, p2=1, pos2=[10,0], velocity2=[0,0], measurement_times at 0, 0.5, 1, 2, 3, 4, 5.
If it's inverse square law, at 10 units, acceleration should be (5/10)^2 = 1/4 of the acceleration at 5 units.
In the first experiment, at 5 units, initial acceleration was ~0.0318 m/s^2, so at 10 units, it should be around ~0.00795 m/s^2.
Let me run that experiment.

Action:

```
<run_experiment>
[
  {
    "p1": 1.0,
    "p2": 1.0,
    "pos2": [10.0, 0.0],
    "velocity2": [0.0, 0.0],
    "measurement_times": [0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0]
  }
]
</run_experiment>
```

Environment (<experiment_output>, abridged):

```
[{"measurement_times":[0.0,0.5,...,5.0],
 "pos1":[[0.0,0.0],[0.0,0.0],...,[0.0,0.0]],
 "pos2":[[10.0,0.0],[9.998,0.0],...,[9.8004,0.0]],
 "velocity1":[[0.0,0.0],[0.0,0.0],...,[0.0,0.0]],
 "velocity2":[[0.0,0.0],[-0.008,0.0],...,[-0.0801,0.0]]}]
```

Round 4

Thinking (6,967 characters, abridged):
[. . .] distance.
Wait, but maybe first try varying p2. Wait, in the previous experiments, p2 was 1 each time. Let's see if p2 affects the motion.
Wait, the problem states that p2 is a scalar property of the probe particle. If the law depends on p2, then changing p2 should change the motion.
In the first two experiments, p2 was 1. Let's try changing p2 to see if acceleration depends on it.
For example, first experiment with p2=2, same p1=1, initial position (5,0), velocity 0.
Let me run that experiment.
Fourth experiment: p1=1, p2=2, pos2=[5,0], velocity2=[0,0], measurement_times same as before.
If acceleration depends on p2, then doubling p2 should change acceleration.
Let's do that.

Action:

```
<run_experiment>
[
  {
    "p1": 1.0,
    "p2": 2.0,
    "pos2": [5.0, 0.0],
    "velocity2": [0.0, 0.0],
    "measurement_times": [0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0]
  }
]
</run_experiment>
```

Environment (<experiment_output>, abridged):

```
[{"measurement_times":[0.0,0.5,...,5.0],
 "pos1":[[0.0,0.0],[0.0,0.0],...,[0.0,0.0]],
 "pos2":[[5.0,0.0],[4.998,0.0],...,[4.7997,0.0]],
 "velocity1":[[0.0,0.0],[0.0,0.0],...,[0.0,0.0]],
 "velocity2":[[0.0,0.0],[-0.008,0.0],...,[-0.0807,0.0]]}]
```

Round 5

Thinking (15,017 characters, abridged):
[. . .] p2) = 0.159 * 1 / (5 * 3) ≈ 0.159 /15 ≈ 0.0106 m/s^2.
So at t=0.5, velocity should be ~ -0.0106 * 0.5 = -0.0053? Wait, no: acceleration is a, so velocity at time t is a * t. Wait, initial velocity is 0, so at time t, velocity is a * t.
Wait, in Exp 1, acceleration was ~0.0318 m/s^2, so at t=0.5, velocity was 0.0318 * 0.5 = 0.0159, but direction was negative, so -0.0159. That matches.
So for the new experiment with p2=3, p1=1, r=5:
Expected acceleration a = 0.159 / (5 * 3) = 0.159 /15 ≈ 0.0106 m/s^2.
At t=0.5, velocity should be ~ -0.0106 * 0.5 = -0.0053 m/s.
Let's run that experiment to check.
Fifth experiment: p1=1, p2=3, pos2=[5,0], velocity2=[0,0], measurement_times=[0,0.5,1,2,3,4,5]

Action:

```
<run_experiment>
[
  {
    "p1": 1.0,
    "p2": 3.0,
    "pos2": [5.0, 0.0],
    "velocity2": [0.0, 0.0],
    "measurement_times": [0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0]
  }
]
</run_experiment>
```

Environment (<experiment_output>, abridged):

```
[{"measurement_times":[0.0,0.5,...,5.0],
 "pos1":[[0.0,0.0],[0.0,0.0],...,[0.0,0.0]],
 "pos2":[[5.0,0.0],[4.9987,0.0],...,[4.8668,0.0]],
 "velocity1":[[0.0,0.0],[0.0,0.0],...,[0.0,0.0]],
 "velocity2":[[0.0,0.0],[-0.0053,0.0],...,[-0.0535,0.0]]}]
```

Round 6

Thinking (22,206 characters, abridged):
[. . .] 0.16)

```
    # Define RHS
    def rhs(t, y):
        x, y_pos, vx, vy = y
        r_sq = x**2 + y_pos**2
        a_x = - (k * p1 * x) / (r_sq * p2)
        a_y = - (k * p1 * y_pos) / (r_sq * p2)
        return [vx, vy, a_x, a_y]

    # Initial state
    y0 = [x0, y0, vx0, vy0]
    sol = solve_ivp(rhs, [0, duration], y0, method='RK45', rtol=1e-4, atol=1e-4)
    return [sol.y[0, -1], sol.y[1, -1]], [sol.y[2, -1], sol.y[3, -1]]

def fit_parameters():
    return {
        "k": {"init": 0.16, "bounds": [0.1, 1.0]}
    }
```

Then, run this with the existing experiments.
Let's run an MSE fit with this.
So, the next step is to run a run_mse_fit with these functions.
Let me write that.

Action:

```
<run_mse_fit>
def discovered_law(pos1, pos2, p1, p2, velocity2, duration, **params):
    import numpy as np
    from scipy.integrate import solve_ivp

    # Extract initial position and velocity
    x0, y0 = pos2
    vx0, vy0 = velocity2

    # Get the fitted k from parameters
    k = params.get('k', 0.16)

    # Define the RHS for the ODE solver
    def rhs(t, y):
        # y = [x, y, vx, vy]
        x, y_pos, vx, vy = y
        # Calculate distance squared
        r_sq = x**2 + y_pos**2
        # Acceleration components (magnitude: k * p1 / (r * p2))
        a_x = - (k * p1 * x) / (r_sq * p2)
        a_y = - (k * p1 * y_pos) / (r_sq * p2)
        return [vx, vy, a_x, a_y]

    # Initial state vector: [x, y, vx, vy]
    y0 = [x0, y0, vx0, vy0]
    # Solve the ODE from t=0 to t=duration
    sol = solve_ivp(rhs, [0, duration], y0, method='RK45', rtol=1e-4, atol=1e-4)
    final_x = sol.y[0, -1]
    final_y = sol.y[1, -1]
    final_vx = sol.y[2, -1]
    final_vy = sol.y[3, -1]
    return [final_x, final_y], [final_vx, final_vy]

def fit_parameters():
    return {
        "k": {"init": 0.16, "bounds": [0.1, 1.0]}
    }
</run_mse_fit>
```

Environment (`<mse_fit_output>`):


```
{"loss_before": 1.2398432528979188e-06,
 "loss_after": 4.6188290271413835e-12,
 "fitted_params": {"k": 0.15914605713715754},
 "declared_params": {"k": {"init": 0.16, "bounds": [0.1, 1.0]}},
 "n_training": 5,
 "error": null}
```

Round 7 (final submission)

Thinking (7,655 characters, abridged):
[. . .] fit already optimized k to ~0.159, so the discovered_law function will use that value.
Let me make sure the RHS function is correctly written.
In the RHS, the acceleration components are computed as:
a_x = - (k * p1 * x) / ((x^2 + y^2) * p2)
Yes, because r_sq = x^2 + y^2, so the denominator is r_sq * p2.
Let me test with the first experiment.
First experiment: p1=1, p2=1, initial x=5, y=0.
At t=0, y = [5, 0, 0, 0].
r_sq = 25.
a_x = - (k * 1 * 5) / (25 * 1) = -5k /25 = -k/5.
With k≈0.159, this is -0.0318, which matches the first experiment's initial acceleration (0.0318 m/s^2 magnitude).
Yes, that's correct.
So, the RHS should be correctly implemented.
Let me draft the final law.

Action (`<final_law>`):

```
from scipy.integrate import solve_ivp
import numpy as np

def discovered_law(pos1, pos2, p1, p2, velocity2, duration, **params):
    """
    Simulates particle 2's motion under a central force law where
    acceleration magnitude scales with p1, inversely with distance r
    and p2, with constant k ~ 0.159. Direction always points toward
    particle 1.
    """
    # Extract initial position and velocity
    x0, y0 = pos2
    vx0, vy0 = velocity2

    # Get fitted parameter k (from MSE fit)
    k = params.get('k', 0.159)

    # Define RHS for ODE solver:
    # [x, y, vx, vy] -> [dx/dt, dy/dt, dvx/dt, dvy/dt]
    def rhs(t, y):
        x, y_pos, vx, vy = y
        # Calculate squared distance from particle 1
        r_sq = x**2 + y_pos**2
        # Acceleration components
        # (magnitude proportional to k*p1, direction toward particle 1)
        a_x = - (k * p1 * x) / (r_sq * p2)
        a_y = - (k * p1 * y_pos) / (r_sq * p2)
        return [vx, vy, a_x, a_y]

    # Initial state vector: [x, y, vx, vy]
    y0 = [x0, y0, vx0, vy0]
    # Solve ODE from t=0 to t=duration
    sol = solve_ivp(rhs, [0, duration], y0, method='RK45', rtol=1e-4, atol=1e-4)
    final_x, final_y = sol.y[0, -1], sol.y[1, -1]
    final_vx, final_vy = sol.y[2, -1], sol.y[3, -1]
    return [final_x, final_y], [final_vx, final_vy]

def fit_parameters():
    """Initializes k with value ~0.16 based on prior experiments
    (MSE fit confirmed k~0.159)."""
    return {
        "k": {"init": 0.16, "bounds": [0.1, 1.0]}
    }
```

Action (`<explanation>`): Through systematic experiments, I discovered that particle 2's acceleration magnitude scales with the source property p_1 (doubling p_1 doubled acceleration), inversely with distance r (doubling r halved acceleration, not quartered as in gravity), and inversely with the probe property p_2 (doubling p_2 halved acceleration). This ruled out inverse-square laws (gravity) and confirmed linear scaling with $1/r$. An MSE fit with 5 experiments yielded a constant $k \approx 0.159$, consistent across all tests. The direction of acceleration always points toward particle 1, confirming a central force. The law is thus $a = \frac{kp_1}{rp_2}$ in magnitude, with direction radial inward.

H ERROR ANALYSIS

Every validation trial (22 worlds $\times$ 5 samples per checkpoint, 220 in total) was read in full and tagged against the taxonomy below. Each tag is an independent yes/no per trial, so a trial can carry multiple tags. The bars in Figure 4 report, for each tag, the fraction of the 110 trials per checkpoint in which it appears. Tags are grouped by the skill that failed: *coding*, *experimenting* and *modelling*.

CODING

Code crashes.
The submitted `discovered_law` fails or returns a buggy output regardless of the physics it encodes. Some examples are: a two-dimensional array passed as the ODE state, the acceleration used as a velocity (a first-order ODE), swapped derivative order, a wrong return shape, an index offset into the wrong particles, division by zero for the particle at the origin, or an undefined variable.

Code $\neq$ reasoning.
The code runs and is internally consistent but implements a different law from the one in the explanation or in the reasoning that produced it: a different exponent, $p_1 p_2$ where p_1/p_2 had been found, a flipped sign, a claimed term that is absent, or a stated fitted constant that is not the one in the code.

Evaluation times out.
The evaluator's `eval_failed` flag: the submitted simulation was too slow to be scored within the time limit (typically an $O(N^2)$ pure-Python force loop over 26–36 particles).

EXPERIMENTING

Uninformative experiments.
The run, as designed, could not return an informative answer to any hypothesis: a start inside the singular core ($r \leq 2$, so the probe falls through the origin within a second and everything afterwards is a slingshot), several particles placed on one point, or wrongly formatted input that returns an error.

Misuses fitter.
Either the fit request is defective , or the fitter's answer is ignored: an unchanged mse, a parameter sitting on its bound, or a very large mse accepted as success.

Misreads data.
The numbers are in front of the model and it draws the wrong conclusion from them: the time-major output array is read as per-particle rows, a pass through the origin is read as oscillation or repulsion, or a sign or arithmetic error is made on values it quotes.

Hallucinates data.
The model reasons from outputs it imagined rather than received, or reports numbers that do not appear in the output, This includes pre-writing the result of an experiment and then treating it as observed.

MODELLING

Textbook prior.
The model defaults to a canonical form (Newton, Coulomb, Hooke) or coupling ($p_1 p_2$) without testing it, or keeps it after its own data has contradicted it.

Submits early.
A law is submitted with most of the ten-round budget unused and the hypothesis untested.

Submits refuted law.
The model ran a controlled comparison (one variable changed), computed the number that contradicts its law, and submitted that law anyway, whether or not it labelled the discrepancy noise. Often it is labelled as noise.

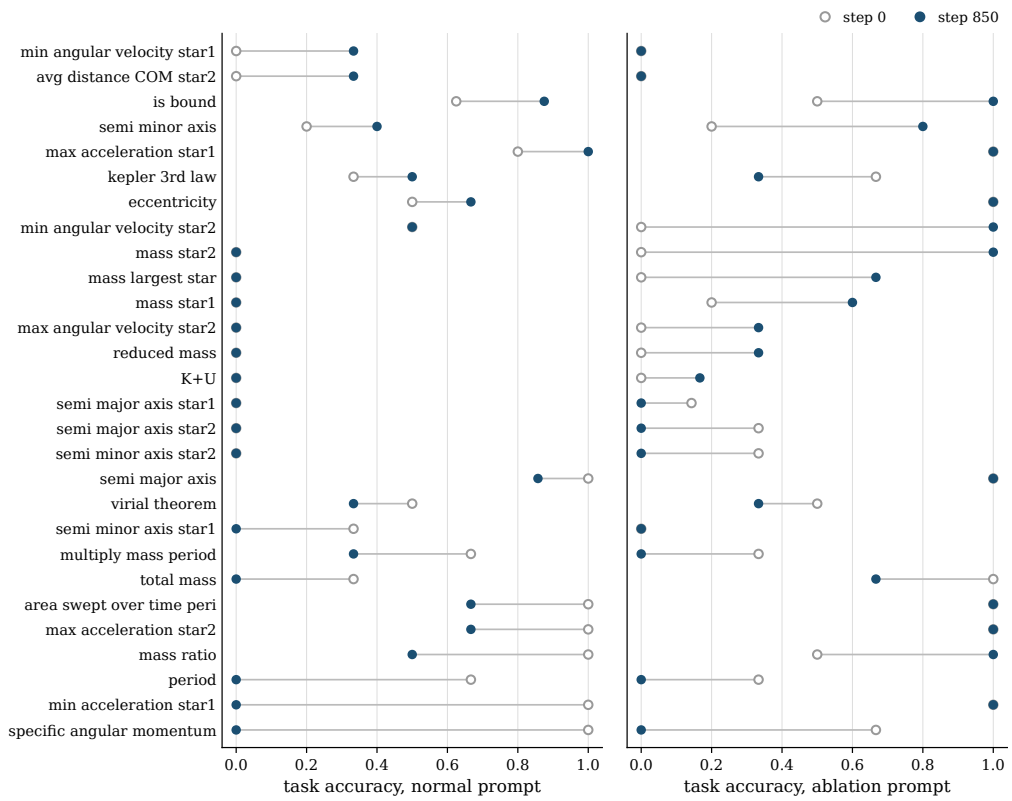

Figure 9: Differences in performance across scenarios in GravityBench with a prompt ablation instructing the model to not use df.head().

Misses latent structure.
The hidden mechanism of the world (time dependence, a hidden mass, species-specific couplings, a background flow, a two-component force, an asymmetric interaction) is never proposed, even when its signature appears in the model's own explanation.

State-independent law.
The submitted law has no dependence on the dynamical state that matters: zero force or a free particle, a constant-magnitude acceleration, a purely kinematic rule (rigid rotation, fixed radius), an algebraic map from initial to final position with no ODE, or a force with a hard-coded direction.

I Analysis of individual scenarios in GravityBench

Figure 9 shows per-task accuracy before and after RL, under the original prompt (left) and with an added instruction to base every calculation on the full column (right). Here we examine the tasks on which the base model outperforms the RL policy, first under the original prompt and then those that remain after the prompt change.

I.1 Under the original prompt.

Several regressions in the left panel trace to one habit. The RL policy calls `df.head()` or indexes rows 0 and 1 in 84% of runs, against 1% for the base model, and then builds its answer based on this small set of printed rows rather than the table. This is probably a consequence that in training the environment's output *is* the data, and reasoning about the printed values in the trace is rewarded.

I.2 PERIOD ESTIMATION: PERIOD, MULTIPLY_MASS_PERIOD, TOTAL_MASS.

To estimate the orbital period the policy computes the orbital angle with `arctan2`, differences it, averages the angular velocity, and divides into 2π. This is the natural estimator for the short trajectories of five to seven points that experiments return in training, but over many orbits the angle crosses the $\pm\pi$ branch cut, the jumps of -2π cancel the accumulated rotation, and the mean angular velocity collapses toward zero. The resulting periods are wrong and the error propagates into anything derived from the period. The base model instead reaches for an FFT or `scipy.signal.find_peaks`, which are immune to the wrap. The policy's alternative attempts, hand-written autocorrelations with an off-by-one lag search, fail in the same runs.

I.3 PER-STAR ORBITAL ELEMENTS: SEMI_MAJOR_AXIS_STAR1/2, SEMI_MINOR_AXIS_STAR2.

Here the policy's derivation is correct up to its last step. It extracts the relative orbit, obtains its semi-major axis from apoapsis and periapsis, and then splits it between the stars assuming equal masses, submitting $a/2$ where the answer is $a\,m_2/M$. Across the seven `semi_major_axis_star1` variants and the `semi_major_axis_star2`, `semi_minor_axis_star2`, `avg_distance_COM` and `min_angular_velocity_star1` tasks, 17 submissions differ from the truth by exactly the ratio of $1/2$ to the true m_i/M. We hypothesise that this reflects the training environment, where couplings and masses are set by the agent rather than being hidden properties to be inferred, but we have not tested this.

I.4 YES/NO QUESTIONS: KEPLER_3RD_LAW, VIRIAL_THEOREM.

On the Newtonian variants of `kepler_3rd_law` the base model computes a single T^2/a^3 ratio, which by itself tests nothing, and returns the textbook answer. The policy attempts an actual test by examining the rate of change of the separation over the first hundred rows, observes it increasing, and concludes that the system is unbound and the law does not apply. The same pattern produces the wrong verdict on `virial_theorem` in the drag world. This is the reduced reliance on textbook priors that Figure 4 shows RL selecting for, and it is what lets the policy pass both modified-gravity variants of `kepler_3rd_law` (exponents 1.97 and 2.03) that the base model fails; on a benchmark where the textbook prior is correct for 26 of 28 task types it is an overall loss. The tests themselves are also built from short windows of data, the same habit as `df.head()`.

I.5 CONVENTION: SPECIFIC_ANGULAR_MOMENTUM.

The policy reads "specific angular momentum of the system" as angular momentum per unit total mass and computes $L_{\mathrm{tot}}/M_{\mathrm{tot}}$, usually with unit masses. The base model recalls the binary-star convention $h = |\mathbf{r}_{\mathrm{rel}} \times \mathbf{v}_{\mathrm{rel}}|$ and writes it down. The policy's definition is defensible but is not the one the grader uses, and since $L_{\mathrm{tot}} = \mu h$ about the centre of mass, supplying the true masses would still give $(\mu/M)\,h$ rather than h.

J EXAMPLE MEDIUM WORLD: EQUIVALENCE_PRINCIPLE_VIOLATION

description

```
A fixed central attractor pulls a mobile probe with a (lightly softened)
↪  inverse-square acceleration whose strength is proportional to the
↪  probe's OWN scalar property, so heavier-labelled probes fall faster
↪  — the equivalence principle fails.
```

true_law

```
$$\vec{a}_2 \;=\;
↪  -\,\frac{G\,p_1\,p_2}{\left(r^2+\epsilon^2\right)^{3/2}}\,\vec{r},
\qquad \vec{r}=\vec{x}_2-\vec{x}_1,\quad r=|\vec r|$$
with $G = 1.0$ and softening length $\epsilon = 0.15$. Particle 1 is
↪  immobile at the origin ($\vec a_1 = 0$), so the magnitude of the
↪  probe's acceleration is
```

```
$$|\vec a_2| = \frac{G\,p_1\,p_2\,r}{(r^2+\epsilon^2)^{3/2}}
↪  \;\xrightarrow[r\gg\epsilon]{}\; \frac{G\,p_1\,p_2}{r^2}.$$
```

explanation

```
The world contains one immobile attractor (particle 1, fixed at the
↪  origin, scalar property $p_1$) and one mobile probe (particle 2,
↪  scalar property $p_2$). The probe is accelerated toward the attractor
↪  with the Plummer-softened inverse-square law $\vec a_2 = -G p_1 p_2
↪  \vec r/(r^2+\epsilon^2)^{3/2}$, with $G=1$ and $\epsilon=0.15$; for
↪  $r\gtrsim 1$ this is indistinguishable from a plain $1/r^2$
↪  attraction to within a percent, and the softening only matters inside
↪  $r\sim0.3$, where the acceleration turns over and goes linearly to
↪  zero at the origin. The conceptual hook is that the ACCELERATION, not
↪  merely the force, carries a factor of the probe's own scalar property
↪  $p_2$: two probes released from the same point with the same velocity
↪  but different $p_2$ fall at different rates, and orbital periods
↪  scale as $(p_1p_2)^{-1/2}$. In ordinary Newtonian gravity the probe's
↪  mass would cancel between coupling and inertia; here it does not, so
↪  the universality of free fall is violated and an observer who assumes
↪  it will fit a "central mass" that drifts linearly with the probe's
↪  label. Everything else is standard: the force is central and
↪  isotropic, motion stays planar, angular momentum about the origin is
↪  conserved, and bound orbits close almost exactly (with a small
↪  softening-induced prograde precession) as long as they stay outside
↪  $r\sim0.5$.
```

rubric

```
10 — Identifies the central, isotropic attraction with acceleration
↪  magnitude $G p_1 p_2/(r^2+\epsilon^2)^{3/2}\cdot r$ (i.e.
↪  inverse-square at large $r$ with Plummer softening), recovers
↪  $G\approx1$ and $\epsilon\approx0.15$ within ~30%, states the
↪  bilinear $p_1p_2$ coupling, AND explicitly notes that the probe's own
↪  scalar enters its acceleration so free-fall rates / orbital periods
↪  depend on $p_2$ (equivalence-principle violation, period $\propto
↪  (p_1p_2)^{-1/2}$).
 7–9 — Gets the inverse-square central attraction with $p_1p_2$ coupling
 ↪  and $G\approx1$, and notes the $p_2$-dependence of the acceleration
 ↪  (UFF violation), but misses, mis-fits, or only qualitatively
 ↪  describes the softening scale $\epsilon$ (or its functional form).
 4–6 — Correctly identifies a central attractive force toward the origin
 ↪  whose strength grows with both scalars, and gets the radial
 ↪  dependence approximately right (power law near $-2$, or
 ↪  "inverse-square outside a small core"), but does not articulate the
 ↪  equivalence-principle violation or gets the coupling/exponent only
 ↪  roughly.
 1–3 — Recognizes only that motion is central/bounded-or-hyperbolic about
 ↪  the origin, but proposes a qualitatively wrong radial law (e.g.
 ↪  harmonic restoring force, damping terms, non-central or
 ↪  velocity-dependent forces) or misses the dependence on the scalar
 ↪  properties.
   0 — Empty or irrelevant.
```

law_stub

```
def discovered_law(pos1, pos2, p1, p2, velocity2, duration):
    # pos1: position of particle 1 (always at the origin)
    # pos2: initial position of particle 2
    # p1: scalar property of particle 1
    # p2: scalar property of particle 2
    # velocity2: initial velocity of particle 2
    # duration: float, simulate from t=0 to t=duration
    # return: (final_pos2, final_vel2)
    return final_pos2, final_vel2
```

discovered_law

```
def discovered_law(pos1, pos2, p1, p2, velocity2, duration):
    import numpy as np
    from scipy.integrate import solve_ivp

    G = 1.0
    eps = 0.15

    x1 = np.asarray(pos1, dtype=float)
    x0 = np.asarray(pos2, dtype=float)
    v0 = np.asarray(velocity2, dtype=float)

    k = G * float(p1) * float(p2)

    def rhs(t, s):
        dx = s[0] - x1[0]
        dy = s[1] - x1[1]
        r2 = dx * dx + dy * dy
        inv = k / (r2 + eps * eps) ** 1.5
        return [s[2], s[3], -inv * dx, -inv * dy]

    if duration == 0:
        return x0.copy(), v0.copy()

    sol = solve_ivp(rhs, [0.0, float(duration)],
                    [x0[0], x0[1], v0[0], v0[1]],
                    method='DOP853', rtol=1e-11, atol=1e-12)
    sf = sol.y[:, -1]
    return np.array([sf[0], sf[1]]), np.array([sf[2], sf[3]])
```