PhysCodeBench: Benchmarking Physics-Aware Symbolic Simulation of 3D Scenes via Self-Corrective Multi-Agent Refinement

Tianyidan Xie[1,⋆], Peiyu Wang[2], Jiaxin Hu[1], Yuyi Qian[1], Yuxuan Wang[1], Shenyi Wang[1], Rui Ma[3], Yanlun Peng[4], Lanjun Wang[5], Ying Tai[1], Jian Yang[1], and Zili Yi[1,⋆]

[1] Nanjing University
[2] Skywork AI
[3] Jilin University
[4] Great Wall Motor
[5] Tianjin University

Abstract. Translating natural-language descriptions of physical phenomena into executable simulation code requires both programming expertise and physical reasoning. Current large language models (LLMs) lack this combination: they frequently produce code that runs but simulates the wrong physics. We introduce PhysCodeBench, the first benchmark for this task, with 1,200 expert-validated examples spanning four physical domains. Its evaluation suite, PhysCodeEval, goes beyond executability and visual fidelity to measure physical correctness directly from the engine state via conservation-law residuals and expert-written assertions, and supports cross-engine evaluation to disentangle physics reasoning from API fluency. As a reference method, we propose the Self-Corrective Multi-Agent Refinement Framework (SMRF), which decouples physics-aware error correction from code generation through specialized agents. This design is motivated by our finding that targeted correction, rather than generic iterative refinement, is the key driver of physical accuracy. SMRF nearly triples the physical-assertion pass rate of the best proprietary baseline (70.6% vs. 23.8%) and retains its advantage under cross-engine transfer.

Keywords: Physics Simulation · Code Generation · Multi-Agent Systems · Benchmark

1 Introduction

Encoding physical laws in executable simulation code is central to robotics, embodied AI, and scientific computing, yet it remains a formidable challenge that demands both programming expertise and physical reasoning. Whether large

⋆ Corresponding authors: Tianyidan Xie (sealical@outlook.com) and Zili Yi (yi@nju.edu.cn).

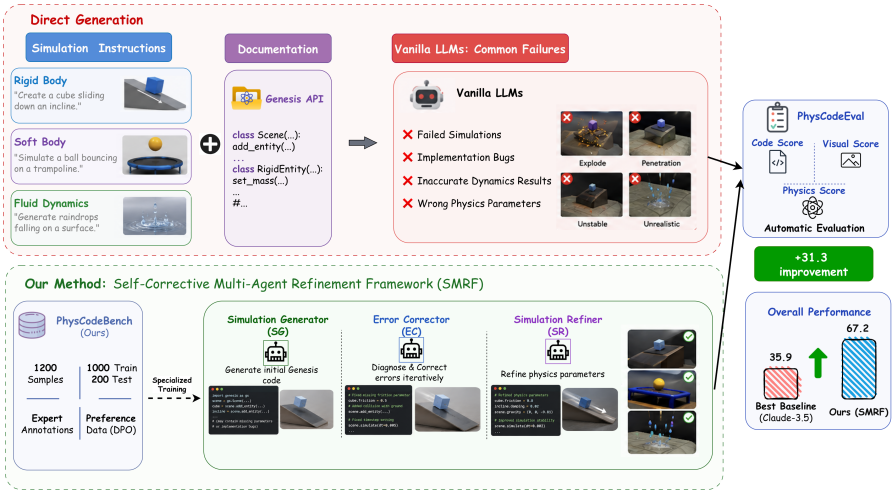

Fig. 1: Overview of PhysCodeBench and SMRF. Top: vanilla LLMs prompted with a simulation instruction and engine documentation produce failed, unstable, or physically incorrect simulations. Bottom: the SMRF reference framework decomposes the task across specialized generation, error-correction, and refinement agents. All outputs are scored by PhysCodeEval across code-level, visual, and direct physical-correctness axes.

language models (LLMs) can bridge this gap is an open question, but answering it requires a benchmark in which physical understanding is *observable* in the code and the trajectories it produces.

By symbolic simulation we mean writing a program that instantiates a physical 3D scene in a simulator, so that the physics is carried by the code rather than by a learned generative model. The task compounds three difficulties: the code must execute without runtime errors, it must implement the intended physical laws and parameters, and it must respect boundary conditions and numerical-stability constraints.

Consider seemingly simple tasks like creating a cube sliding down an incline, simulating a ball bouncing on a trampoline, or generating raindrops falling on a surface. These scenarios require precise implementation of gravity, elasticity, surface tension, and collision dynamics. As illustrated in Figure 1, vanilla LLMs struggle with these tasks, producing failed simulations, implementation bugs, inaccurate dynamics, and incorrect physics parameters. Even when the code executes successfully, GPT-4o [16] and Claude-3.5-Sonnet [1] frequently generate simulations whose object interactions deviate from the described scenario: execution is only the first hurdle.

Existing code-generation benchmarks [4, 29] lack the evaluation criteria this task demands. Visual plausibility alone is insufficient: a smooth video can hide a violated conservation law. What is needed is *direct* measurement of physical correctness in the simulator's state space, together with cross-engine evaluation that disentangles physics reasoning from memorized API fluency.

We address this gap with three key contributions:

- **PhysCodeBench**: the first benchmark for physics-aware symbolic simulation of 3D scenes, comprising 1,200 expert-validated examples across rigid-body, soft-body, fluid dynamics, and mechanics domains, with seed-level train/test isolation and a three-fold contamination audit.
- **PhysCodeEval**: an evaluation suite that complements execution and visual metrics with direct physical-correctness measurements read from the engine state, including conservation-law residuals and expert-written physical assertions, and supports cross-engine evaluation on a MuJoCo-native subset.
- **SMRF**: a reference multi-agent framework that decouples physics-aware error correction from code generation, nearly tripling the assertion pass rate of the best proprietary baseline and retaining its advantage across engines.

2 Related Work

Physics simulation and symbolic code generation. Prior work studies physical reasoning through video prediction [3,33,38,39] or domain-specific solvers [24,30], but none targets executable simulation code. LLM-based approaches invoke engines for reasoning [6,19] or generate control programs, reward functions, and scene or task specifications [13,14,18,21,34,35,40], yet none requires authoring a complete physics simulation whose physical correctness is then measured.

Code generation and its evaluation. Code generation has advanced with models like Codex [4], CodeLlama [29], DeepSeek-R1 [11], and Qwen-2.5 [37]. These are evaluated on general programming benchmarks such as HumanEval [4] and MBPP [2], where correctness is fully captured by input–output tests. Physics simulation breaks this assumption: code can pass every syntactic check yet simulate the wrong world. Fine-tuning approaches like WizardCoder [20] show specialization improves domain performance, motivating our multi-agent approach, but evaluation methodology for *physically* correct code has been missing. PhysCodeEval fills this gap with state-space checks.

Multi-agent systems and preference alignment. Multi-agent frameworks like AutoGen [36] and ChatDev [23] demonstrate how specialized agents collaborate on complex tasks. In code generation, approaches like Self-Debugging [5] and CodeChain [17] show iterative refinement improves quality, but lack physics-specific knowledge. We show that generic self-refinement leaves a 26-point gap to specialized correction. Preference alignment (RLHF [22], DPO [26]) improves LLM quality generally, and we apply it to physics-specific agent roles with expert simulation preferences.

3 The PhysCodeBench Benchmark

3.1 Overview

PhysCodeBench comprises 1,200 examples spanning rigid-body physics (410), soft-body physics (310), fluid dynamics (275), and mechanics (205), with two

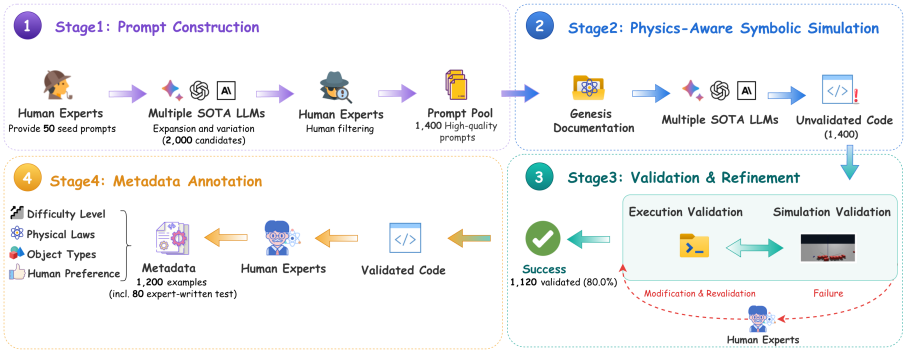

Fig. 2: PhysCodeBench dataset curation pipeline. Stage 1: 50 expert seed prompts are expanded by multiple LLMs into 2,000 candidate instructions, and human filtering retains a pool of 1,400. Stage 2 (physics-aware symbolic simulation): multiple LLMs generate implementations against the Genesis documentation. Stage 3: two-stage validation (execution, then simulation faithfulness) with an expert modification loop. In total, 430 candidates (30.7%) pass both stages on the first attempt and 1,120 (80.0%) are retained after at most three expert revision rounds. Together with 80 expert-authored test instructions, this yields the final 1,200 examples. Stage 4: experts annotate difficulty, physical laws, object types, and preferences.

difficulty levels (709 easy / 491 hard). Hard examples involve three or more interacting physical laws, multi-phase scenarios, or parameter regimes requiring precise tuning. Each example pairs a natural-language instruction with an expert-validated reference implementation for the Genesis physics engine [8], plus difficulty, physical-law, object-type, and preference metadata. The dataset is split into 1,000 training and 200 testing examples with balanced domain coverage (test: 68 rigid-body, 52 soft-body, 46 fluid, 34 mechanics) and 116 easy / 84 hard examples. Dataset examples and complete statistics are provided in the technical appendix.

3.2 Dataset Curation Process

Construction follows a four-stage pipeline (Figure 2).

Stage 1: Prompt construction. Human experts first draft 50 seed prompts arranged in a 4×2 domain-by-difficulty design covering 32 distinct combinations of physical laws. These seed prompts are then provided to multiple state-of-the-art proprietary models [1, 9, 10, 16] for expansion and variation. Expansion is constrained to stay within a single seed's scenario family, forbidding cross-seed template merging. Human experts subsequently filter the machine-generated candidates for clarity, physical well-posedness, and diversity to form the Prompt Pool. Through this process, we generated 2,000 instruction candidates from the 50 seed prompts and retained a pool of 1,400.

Stage 2: Physics-aware symbolic simulation. For each prompt in the Prompt Pool, we generate simulation code with the same set of proprietary models, each given the prompt, the Genesis documentation, and the formatting requirements. Using multiple models keeps the reference implementations from inheriting a single model's coding style.

Stage 3: Code validation and refinement. The generated code undergoes a two-stage validation process: execution validation and simulation validation. Execution validation tests whether the code executes without errors in the Genesis environment. Simulation validation examines whether the executed code produces simulation visualizations that match the intended physical scenario. Only 430 of the 1,400 pooled candidates (30.7%) pass both validations on the first attempt. When validation fails, human experts manually modify the code and repeat the validation process, with up to three modification attempts allowed. In total, 1,120 pooled instructions (80.0%) yield a validated implementation and 280 are discarded. All retained code is additionally rewritten by experts (mean CodeBLEU [27] 0.46 against the original LLM draft), so reference solutions are not verbatim LLM outputs. Adding 80 test instructions authored, together with their reference implementations, entirely by experts (see Test-Set Integrity) yields the final 1,200 examples.

Stage 4: Metadata annotation. After validating the prompt–code pairs, we enrich each example with comprehensive metadata. Three annotators label difficulty (mean pairwise Cohen's $\kappa = 0.78$) and physical-law tags (mean pairwise Jaccard-based $\kappa = 0.71$). To prepare preference data for training, we generate two different validated implementations for 600 scenarios. Five experts provide 1,980 pairwise preference judgments over these 600 implementation pairs (3.3 raters per pair, Krippendorff's $\alpha = 0.68$), and the majority label per pair yields the preference data used for training.

3.3 Test-Set Integrity

Because both the candidate instructions and draft code originate partly from proprietary LLMs, we protect the test split by construction and audit it post hoc.

Seed isolation. 15 of the 50 seed prompts are reserved for the test split: their expansions never enter training data. Of the 1,120 pool-derived examples, 1,000 (from the 35 training seeds) form the training set and 120 (from the 15 held-out seeds) enter the test set. The remaining 80 test instructions are newly authored by experts with no LLM in the loop. All Error-Corrector tetrads and DPO preference scenarios are drawn exclusively from the 1,000 training instructions. No artifact derived from a test instruction appears in any agent's training data.

Contamination audit. We measure train–test proximity three ways: (i) *semantic*: embedding cosine similarity between each test instruction and its nearest training neighbor has mean 0.41 and max 0.63, and zero pairs exceed the 0.85 near-duplicate threshold; (ii) *lexical*: BM25 retrieval [28] gives a maximum token-level Jaccard overlap of 0.27; (iii) *code-level*: zero-shot samples (five per prompt) from the strongest baselines never exceed CodeBLEU 0.7 against the corresponding test reference implementations, giving no evidence that test solutions are memorized. Note that pretraining contamination, if present, would inflate the *proprietary baselines* rather than our open-weight reference method, making the reported gaps conservative.

4 The PhysCodeEval Evaluation Suite

PhysCodeEval scores a generated simulation on three axes. Its first two form the 100-point score Total $= S_{\text{code}} + S_{\text{visual}}$ used throughout our experiments. The third axis measures physical correctness directly and is reported as a first-class metric alongside the total.

Code quality score S_{code} (50 pts). S_{code} evaluates whether the generated code executes successfully (S_{exec}, 25 points, awarded in full or not at all) and produces the expected simulation artifact (S_{file}, 25 points, scored by a fixed partial-credit rubric over existence, size, and format compliance).

Simulation fidelity score S_{visual} (50 pts). S_{visual} assesses the generated simulation quality through two metrics: S_{clip} measures semantic alignment between simulation video and instruction using ClipScore [12], while S_{motion} evaluates physical realism via a flow-based motion-smoothness measure inspired by VBench [15], detecting anomalies such as jitter, unrealistic accelerations, or object intersections. Each component contributes up to 25 points. These perceptual metrics are useful but insufficient alone: a visually smooth video can violate conservation laws. This motivates the third axis.

Direct physical-correctness measurements. Both measurements below are computed from the *engine state interface* rather than from rendered video, eliminating vision-pipeline error.

- *Assertion score S_{phys}.* During annotation, experts wrote 1–3 executable physical assertions per test instruction (436 in total), e.g., "first rebound height $\in [0.6h_0, 0.95h_0]$" or "$\geq$3 concentric ripples within 0.3 s of impact." Assertions are authored from the instruction text alone, blind to any implementation, and each is reviewed by a second expert. The expert reference implementations pass 98.6% of them. S_{phys} is the fraction of assertions passed, where every assertion of a program that fails to execute counts as failed. Because assertion counts vary per scenario, S_{phys} is reported alongside the 100-point total rather than folded into it.

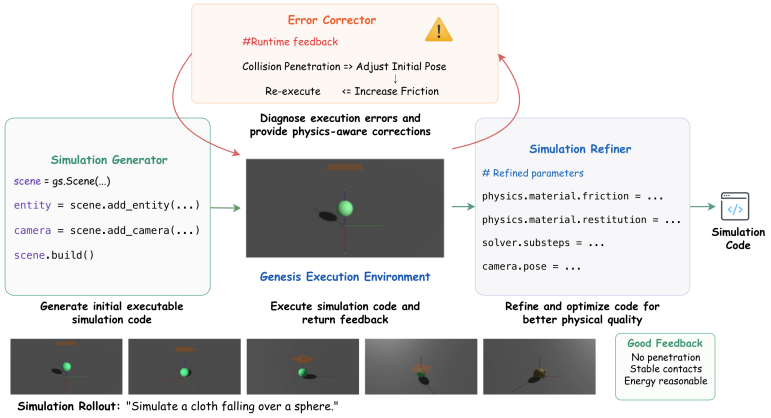

Fig. 3: The Self-Corrective Multi-Agent Refinement Framework (SMRF). The Simulation Generator produces initial code, which is executed in the Genesis environment. Runtime feedback drives the Error Corrector, which diagnoses execution errors and provides physics-aware corrections (up to three rounds). Once execution succeeds, the Simulation Refiner tunes physical parameters and code quality under preference alignment. Bottom: a simulation rollout for "Simulate a cloth falling over a sphere."

– *Conservation checks (companion diagnostic).* For every step we read center-of-mass positions, velocities, and angular velocities. For scenario labels with well-defined invariants we compute residuals: momentum drift $\|\Delta p\|/\|p_0\|$ for collision scenarios, fitted acceleration versus g for free fall, and kinetic-energy retention for elastic versus dissipation bounds for inelastic contact, each passed at a 5% tolerance and reported on the subset where invariants are defined. Unlike S_{phys}, which is a score and so counts every assertion of a non-executing program as failed, these residuals are a diagnostic evaluated only on runs that produced a trajectory, which is what lets them separate physical error from execution failure. For dissipative scenarios (soft bodies, fluids) where global conservation is ill-defined, we check scenario-appropriate necessary conditions instead: monotone energy dissipation, terminal-state stability, and non-interpenetration.

Cross-engine protocol. To separate physics reasoning from single-engine API fluency, 120 test instructions carry MuJoCo-native reference implementations and assertion sets, enabling the code, perceptual, and assertion axes to be computed on MuJoCo [32]. This makes cross-engine evaluation a property of the *benchmark*, independent of any particular method's adaptation strategy.

5 Methodology

The Self-Corrective Multi-Agent Refinement Framework (SMRF) splits generation, correction, and refinement across specialized agents (Figure 3). All agents are

initialized from DeepSeek-R1-Distill-Qwen-32B (DRDQ-32B) [7] and specialized through role-specific training.

Agents and control flow. The **Simulation Generator (SG)** is trained on PhysCodeBench's 1,000 training pairs using supervised fine-tuning (SFT) and produces initial code based on the instruction prompt, with the standard objective

$$\mathcal{L}_{\text{SFT}} = -\sum_{i=1}^{N} \log p_\theta(y_i \mid x_i), \tag{1}$$

where x_i represents the instruction prompt and y_i the reference implementation. The generated code is executed in the Genesis environment to identify runtime errors or execution failures. When execution fails, the **Error Corrector (EC)** diagnoses specific errors and proposes corrections based on its training on 520 tetrads $(x, c_{\text{bug}}, e, c_{\text{fix}})$ of (instruction, incorrect code, error description, corrected code), harvested from the validation stage of dataset curation and restricted to training instructions (43% API misuse, 28% parameter miscalibration, 18% boundary conditions, 7% temporal discretization, 4% other), with loss computed only on c_{fix} tokens. The EC attempts to correct errors up to 3 times. If errors persist after these attempts, the framework returns a failure. When execution succeeds, the **Simulation Refiner (SR)** further optimizes the code by refining physical parameters while maintaining code structure. The SR is first trained with SFT, then aligned with human preferences on the 600 majority-labeled preference pairs (1,980 judgments) through direct preference optimization (DPO) [26]:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}\left[\log \sigma\left(\beta \log \frac{p_\theta(y_w|x)}{p_{\text{ref}}(y_w|x)} - \beta \log \frac{p_\theta(y_l|x)}{p_{\text{ref}}(y_l|x)}\right)\right], \tag{2}$$

where y_w and y_l are the preferred and less preferred implementations and $\beta = 0.1$.

6 Experiments

6.1 Experimental Setup

Models and baselines. We compare our approach against several strong baselines. For proprietary models, we evaluate GPT-4o [16], Claude-3.5-Sonnet [1], and Gemini-2.0-Pro [10]. From the open-source community, we test DeepSeek-R1 [11], DRDQ-32B [7], Qwen-2.5-32B [37], and QwQ-32B [25], spanning general-purpose and reasoning-oriented open models. We also create fine-tuned single-agent variants of DRDQ-32B using SFT and DPO, which operate without the multi-agent framework. Our SMRF implementation includes three variants: SMRF (base) with unspecialized agents, SMRF + SFT, and SMRF + SFT + DPO (our complete framework).

Training and inference. All agents are specialized through role-specific LoRA fine-tuning on 2 NVIDIA A100 80GB GPUs, using a learning rate of 10^{-5}, batch size of 2, 5 epochs for SFT and 3 epochs for DPO, requiring approximately 34 GPU-hours for the main configuration. During inference, each model is provided

Model	Code	Visual	Total	S_{phys} (%)
Proprietary (zero-shot)				
GPT-4o	15.7	18.1	33.8	21.6
Claude-3.5-Sonnet	17.0	18.9	<u>35.9</u>	<u>23.8</u>
Gemini-2.0-Pro	14.8	16.8	31.6	19.7
Open-source (zero-shot)				
DeepSeek-R1	13.8	15.7	29.5	17.9
DRDQ-32B	12.0	15.6	27.6	16.2
Qwen-2.5-32B	0.8	1.1	1.9	0.9
QwQ-32B	6.7	8.8	15.5	7.4
Single-agent fine-tuned				
DRDQ-32B + SFT	17.2	18.2	35.4	28.9
DRDQ-32B + SFT + DPO	18.4	19.1	37.5	32.1
Multi-agent (SMRF)				
Base (vanilla agents)	16.2	17.5	33.7	30.4
+ SFT	30.5	31.1	61.6	63.5
+ SFT + DPO	**33.2**	**34.0**	**67.2**	**70.6**
Expert reference (oracle)	50.0	44.1	94.1	98.6
vs. best proprietary baseline			**+31.3**	**+46.8**

Table 1: Results on the PhysCodeBench test set (200 examples, avg@5). Code and Visual are each out of 50, Total out of 100. S_{phys} is the physical-assertion pass rate. The oracle row scores the expert reference implementations and gives the attainable ceiling. Bold = best system, underline = best proprietary baseline. DRDQ-32B = DeepSeek-R1-Distill-Qwen-32B.

with Genesis physics engine documentation: models with long context windows receive the full ≈100K-token corpus, while 32K-context models receive ≈20K tokens of BM25-retrieved task-relevant sections (a context ablation justifying this protocol is in the technical appendix). Each model generates responses with a maximum length of 4,096 tokens and temperature of 0.1. We perform 5 inference passes per test prompt and report the average score (avg@5). Complete configurations are in the technical appendix.

6.2 Quantitative Analysis

Table 1 presents the performance of all approaches on the PhysCodeBench test set. Several key findings emerge from these results. First, the task is far from solved: the best zero-shot model, Claude-3.5-Sonnet, reaches 35.9 out of 100 points and passes only 23.8% of physical assertions despite full engine documentation in context. Second, direct physical measurement changes the picture. The gap between the strongest fine-tuned single agent and SMRF is 29.7 points on the 100-point total but 38.5 points on S_{phys}. The two axes even reorder systems: DRDQ-32B+SFT ranks below Claude-3.5-Sonnet on the total (35.4 vs. 35.9) yet above it on S_{phys} (28.9% vs. 23.8%), and SMRF (base) shows the same inversion against GPT-4o (33.7 vs. 33.8 on the total, 30.4% vs. 21.6% on S_{phys}). A leaderboard built only from execution and perceptual scores would therefore

Model	Easy	Hard
GPT-4o	43.0	21.1
Claude-3.5-Sonnet	45.2	23.1
Gemini-2.0-Pro	40.7	19.0
DRDQ-32B + SFT + DPO	46.9	24.5
SMRF (base)	42.8	21.1
SMRF + SFT	72.1	47.1
SMRF + SFT + DPO	**77.5**	**53.0**
Improvement over strongest single agent	+30.6	+28.5

Table 2: Total score by difficulty (116 easy / 84 hard test examples). The improvement row is computed against DRDQ-32B+SFT+DPO, the strongest non-SMRF system.

rank these systems wrongly on physics, which is precisely what the third axis is for. Qwen-2.5-32B is an outlier at 1.9 points: it rarely emits a complete runnable Genesis script, so execution succeeds on under 5% of samples and both S_{code} and S_{phys} collapse. This is a property of the model rather than of our harness, since two close relatives score far higher under the identical protocol, namely QwQ-32B at 15.5 and the DeepSeek-R1 distillation of the same backbone at 27.6. Notably, SMRF with vanilla agents nearly doubles S_{phys} over its DRDQ-32B backbone (16.2%→30.4%) while the total rises only 6.1 points, showing that execution-feedback-driven correction improves physics disproportionately to executability. Third, while SFT alone provides substantial improvements (increasing performance from 33.7 to 61.6 points), adding DPO further enhances performance by an additional 5.6 points by aligning the Simulation Refiner's behavior with expert preferences.

On the conservation subset (102 rigid-body and mechanics test cases), SMRF satisfies momentum- and energy-conservation residuals within the 5% tolerance in 81.4% of checks versus 35.3% for Claude-3.5-Sonnet and 42.2% for the fine-tuned single agent. Because these residuals are measured only on runs that executed, they isolate the failure mode this benchmark targets: Claude-3.5-Sonnet produces a running simulation that nevertheless violates momentum or energy conservation in roughly two thirds of the cases where the invariant is defined. Per-residual breakdowns are in the technical appendix.

6.3 Performance Across Difficulty Levels

Table 2 reveals that all models exhibit a substantial performance drop when moving from easy to hard tasks (three or more interacting laws, multi-phase scenarios). Claude-3.5-Sonnet loses 48.9% of its easy-task score on hard tasks (45.2→23.1), while SMRF + SFT + DPO loses 31.6% (77.5→53.0) and keeps a 28.5-point lead over the strongest single agent, where compound failures of syntax, physics, and stability are most likely.

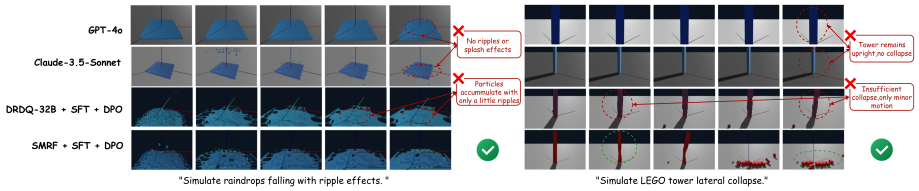

Fig. 4: Qualitative comparison. Left ("raindrops falling with ripple effects"): baselines render a nearly static water sheet, while SMRF produces droplets, splashes, and expanding surface waves. Right ("LEGO tower lateral collapse"): baselines leave the tower intact. Only SMRF produces progressive toppling with scattering bricks.

Configuration	Code	Visual	Total	S_{phys} (%)
Full SMRF + SFT + DPO	33.2	34.0	**67.2**	**70.6**
— without EC	27.4	28.5	55.9	52.8
— without SR	28.8	29.6	58.4	59.8
— SR without DPO	30.5	31.1	61.6	63.5

Table 3: Ablation study of SMRF components (200 examples, avg@5).

6.4 Ablation Studies

Table 3 reveals the importance of each SMRF component, and every ablation costs more on S_{phys} than on the total. Removing the Error Corrector is the largest single-component effect, at 11.3 points of total and 17.8 points of S_{phys}, since failed executions are never recovered and their assertions all count as failed. Removing the Simulation Refiner costs 8.8 points of total and 10.8 of S_{phys}, reflecting its role in physical-parameter tuning. DPO alignment adds 5.6 points of total and 7.1 of S_{phys} over the SFT-only refiner, which is the SMRF + SFT row of Table 1 because DPO acts on the SR alone. This is the ablation evidence behind our claim that targeted correction, rather than generic refinement, is what drives physical accuracy.

6.5 Cross-Engine Evaluation

Two claims must be separated: whether the *benchmark* can evaluate across engines, and whether a particular *method* transfers. The MuJoCo-native subset addresses the first by construction. The proprietary baselines drop about 7 points from their Genesis test-set scores (e.g., Claude-3.5-Sonnet 35.9→28.5), and their assertion pass rates remain low on both engines (Claude: 20.1% on MuJoCo). For the second claim, Table 4 shows that SMRF trained purely on Genesis transfers to MuJoCo at 41.2 points zero-shot, above every zero-shot baseline, and recovers 87.9% of its Genesis performance on the same 120 instructions (58.3 vs. 66.3) with only 250 API-mapping samples. Zero-shot retention is 62.1%. The EC's correction success rate degrades only from 84% on Genesis to 79% under zero-shot transfer: its non-API failure modes (invalid masses, out-of-range coefficients,

Approach	Adapt.	Total
GPT-4o (zero-shot)	0	26.8
Gemini-2.0-Pro (zero-shot)	0	24.9
Claude-3.5-Sonnet (zero-shot)	0	28.5
DRDQ-32B + SFT + DPO (adapted)	250	36.7
SMRF (Genesis-trained, transfer)	0	41.2
SMRF + adaptation samples	250	**58.3**

Table 4: Cross-engine results on the 120-example MuJoCo-native test subset. "Adapt." = number of MuJoCo API-mapping samples used (0 = zero-shot).

Model	Phys.	Read.	Useful.
GPT-4o	3.1	3.5	3.4
Gemini-2.0-Pro	3.0	3.6	3.2
Claude-3.5-Sonnet	3.2	3.7	3.4
DRDQ-32B + SFT + DPO	3.4	3.5	3.6
SMRF + SFT + DPO	**4.5**	**4.2**	**4.6**

Table 5: User study results (10 participants, 40 prompts, 1–5 scale): physical accuracy, code readability, overall usefulness.

dimensional inconsistencies) are properties of physics rather than of any engine. PhysCodeBench performance thus reflects transferable physical reasoning, with engine-specific knowledge bridgeable by a small sample budget.

6.6 User Study and Qualitative Analysis

We conducted a user study with 10 participants (6 students, 4 professional developers) with physics simulation experience, who rated implementations for 40 prompts on which all compared systems produced executable output. Participants rated each implementation on a 1–5 scale across three dimensions: physical accuracy, code readability, and overall usefulness. SMRF received the highest ratings on all three dimensions (Table 5). Inter-rater reliability is ICC(2,1)= 0.74. To validate our automatic metrics, we computed Spearman correlations with human judgments over 120 paired observations (40 prompts × 3 systems: Claude-3.5-Sonnet, DRDQ-32B+SFT+DPO, and SMRF): S_{clip} correlated with physical accuracy ratings ($\rho = 0.82$), and code execution with usefulness ratings ($\rho = 0.76$). These correlations are driven largely by differences between systems, so they support the perceptual axes for system-level ranking rather than as substitutes for the state-based measurements. Study protocol details, including recruitment and consent, are in the technical appendix.

Figure 4 shows the same gap qualitatively: the baselines leave a nearly static fluid surface and an intact tower, while SMRF produces splashes, propagating waves, and progressive toppling with contact-consistent debris. Additional examples with error-correction traces are in the technical appendix.

6.7 Robustness and Integrity Checks

Statistical robustness. Bootstrap 95% confidence intervals (1,000 resamples) around per-domain totals do not overlap between SMRF and any baseline. Seed-grouped five-fold cross-validation over the 1,120 pool-derived examples (all expansions of a seed share a fold, preserving seed isolation) yields a total-score standard deviation of 1.4 points.

Prompt-style control. As a stylistic control (no test instruction shares a seed with training data by construction), SMRF's hard-subset score on LLM-expanded held-out-seed instructions (52.1) is within 1.8 points of its score on expert-authored instructions (53.9), so performance does not depend on test prompts sharing the LLM-expansion style of the training data.

Single-agent refinement with equal budget. Giving GPT-4o, Claude-3.5-Sonnet, and the fine-tuned DRDQ-32B the same three-round self-correction budget yields at most 41.0 points, 26.2 below SMRF, showing the gain comes from specialized decomposition rather than iteration count (full table, including a CLIP-selected best-of-5 baseline, in the technical appendix).

7 Conclusion

We introduce PhysCodeBench, the first benchmark for physics-aware symbolic simulation of 3D scenes: 1,200 expert-validated examples, an evaluation suite that reads conservation residuals and expert assertions directly from engine state, and a MuJoCo-native subset that makes cross-engine evaluation a property of the benchmark itself. Our reference framework SMRF reaches 67.2 points, lifts assertion pass rates from 23.8% to 70.6%, and retains 87.9% of its performance across engines. Limitations remain: the benchmark targets high-level APIs rather than solver implementation, and the error corrector fires only on execution failure. We will release the dataset, SMRF implementation, evaluation framework, and a semi-automated expansion protocol.

References

1. Anthropic: Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet (2024), accessed: 2026-07-01
2. Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al.: Program synthesis with large language models. arXiv preprint arXiv:2108.07732 (2021)
3. Bear, D.M., Wang, E., Mrowca, D., Binder, F.J., Tung, H.Y.F., Pramod, R., Holdaway, C., Tao, S., Smith, K., Sun, F.Y., et al.: Physion: Evaluating physical prediction from vision in humans and machines. arXiv preprint arXiv:2106.08261 (2021)
4. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.D.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)

5. Chen, X., Lin, M., Schärli, N., Zhou, D.: Teaching large language models to self-debug. arXiv preprint arXiv:2304.05128 (2023)
6. Cherian, A., Corcodel, R., Jain, S., Romeres, D.: LLMPhy: Parameter-identifiable physical reasoning combining large language models and physics engines. arXiv preprint arXiv:2411.08027 (2024)
7. DeepSeek-AI: Deepseek-r1-distill-qwen-32b. https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-32B (2025), accessed: 2026-07-01
8. Genesis Authors: Genesis: A universal and generative physics engine for robotics and beyond. https://github.com/Genesis-Embodied-AI/Genesis (December 2024), accessed: 2026-07-01
9. GitHub: Github copilot. https://github.com/features/copilot (2025), accessed: 2026-07-01
10. Google DeepMind: Gemini 2.0 model updates: 2.0 flash, flash-lite, pro experimental. https://blog.google/innovation-and-ai/models-and-research/google-deepmind/gemini-model-updates-february-2025/ (2025), accessed: 2026-07-01
11. Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al.: DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. arXiv preprint arXiv:2501.12948 (2025)
12. Hessel, J., Holtzman, A., Forbes, M., Bras, R.L., Choi, Y.: CLIPScore: A reference-free evaluation metric for image captioning. arXiv preprint arXiv:2104.08718 (2021)
13. Hu, Z., Iscen, A., Jain, A., Kipf, T., Yue, Y., Ross, D.A., Schmid, C., Fathi, A.: SceneCraft: An LLM agent for synthesizing 3d scenes as Blender code. In: Proceedings of the 41st International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 235, pp. 19252–19282. PMLR (2024)
14. Huang, W., Wang, C., Zhang, R., Li, Y., Wu, J., Fei-Fei, L.: Voxposer: Composable 3d value maps for robotic manipulation with language models (2023), https://arxiv.org/abs/2307.05973
15. Huang, Z., He, Y., Yu, J., Zhang, F., Si, C., Jiang, Y., Zhang, Y., Wu, T., Jin, Q., Chanpaisit, N., et al.: VBench: Comprehensive benchmark suite for video generative models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 21807–21818 (2024)
16. Hurst, A., Lerer, A., Goucher, A.P., Perelman, A., Ramesh, A., Clark, A., Ostrow, A., Welihinda, A., Hayes, A., Radford, A., et al.: GPT-4o system card. arXiv preprint arXiv:2410.21276 (2024)
17. Le, H., Chen, H., Saha, A., Gokul, A., Sahoo, D., Joty, S.: CodeChain: Towards modular code generation through chain of self-revisions with representative sub-modules. arXiv preprint arXiv:2310.08992 (2023)
18. Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., Zeng, A.: Code as policies: Language model programs for embodied control. In: 2023 IEEE International Conference on Robotics and Automation (ICRA). pp. 9493–9500 (2023). https://doi.org/10.1109/ICRA48891.2023.10160591
19. Liu, R., Wei, J., Gu, S.S., Wu, T.Y., Vosoughi, S., Cui, C., Zhou, D., Dai, A.M.: Mind's Eye: Grounded language model reasoning through simulation. arXiv preprint arXiv:2210.05359 (2022)
20. Luo, Z., Xu, C., Zhao, P., Sun, Q., Geng, X., Hu, W., Tao, C., Ma, J., Lin, Q., Jiang, D.: WizardCoder: Empowering code large language models with Evol-Instruct. arXiv preprint arXiv:2306.08568 (2023)
21. Ma, Y.J., Liang, W., Wang, G., Huang, D.A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., Anandkumar, A.: Eureka: Human-level reward design via coding large language models. In: International Conference on Learning Representations (2024)

22. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al.: Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems **35**, 27730–27744 (2022)
23. Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., et al.: ChatDev: Communicative agents for software development. arXiv preprint arXiv:2307.07924 (2023)
24. Qiu, S., Guo, S., Song, Z.Y., Sun, Y., Cai, Z., Wei, J., Luo, T., Yin, Y., Zhang, H., Hu, Y., et al.: Phybench: Holistic evaluation of physical perception and reasoning in large language models. arXiv preprint arXiv:2504.16074 (2025)
25. Qwen Team: Qwq-32b: Embracing the power of reinforcement learning. https://qwenlm.github.io/blog/qwq-32b/ (2025), accessed: 2026-07-01
26. Rafailov, R., Sharma, A., Mitchell, E., Manning, C.D., Ermon, S., Finn, C.: Direct preference optimization: Your language model is secretly a reward model. Advances in Neural Information Processing Systems **36**, 53728–53741 (2023)
27. Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., Sundaresan, N., Zhou, M., Blanco, A., Ma, S.: CodeBLEU: A method for automatic evaluation of code synthesis. arXiv preprint arXiv:2009.10297 (2020)
28. Robertson, S., Zaragoza, H.: The probabilistic relevance framework: BM25 and beyond. Foundations and Trends in Information Retrieval **3**(4), 333–389 (2009). https://doi.org/10.1561/1500000019
29. Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X.E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., et al.: Code Llama: Open foundation models for code. arXiv preprint arXiv:2308.12950 (2023)
30. Takamoto, M., Praditia, T., Leiteritz, R., MacKinlay, D., Alesiani, F., Pflüger, D., Niepert, M.: PDEBench: An extensive benchmark for scientific machine learning. Advances in Neural Information Processing Systems **35**, 1596–1611 (2022)
31. Teed, Z., Deng, J.: RAFT: Recurrent all-pairs field transforms for optical flow. In: Computer Vision – ECCV 2020. pp. 402–419. Springer (2020)
32. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 5026–5033 (2012). https://doi.org/10.1109/IROS.2012.6386109
33. Tung, H.Y., Ding, M., Chen, Z., Bear, D., Gan, C., Tenenbaum, J., Yamins, D., Fan, J., Smith, K.: Physion++: Evaluating physical scene understanding that requires online inference of different physical properties. Advances in Neural Information Processing Systems **36**, 67048–67068 (2023)
34. Wang, L., Ling, Y., Yuan, Z., Shridhar, M., Bao, C., Qin, Y., Wang, B., Xu, H., Wang, X.: GenSim: Generating robotic simulation tasks via large language models. In: International Conference on Learning Representations (2024)
35. Wang, Y., Xian, Z., Chen, F., Wang, T.H., Wang, Y., Fragkiadaki, K., Erickson, Z., Held, D., Gan, C.: RoboGen: Towards unleashing infinite data for automated robot learning via generative simulation. In: Proceedings of the 41st International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 235, pp. 51936–51983. PMLR (2024)
36. Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., et al.: AutoGen: Enabling next-gen LLM applications via multi-agent conversation. arXiv preprint arXiv:2308.08155 (2023)
37. Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al.: Qwen2.5 technical report. arXiv preprint arXiv:2412.15115 (2024)

38. Yi, K., Gan, C., Li, Y., Kohli, P., Wu, J., Torralba, A., Tenenbaum, J.B.: CLEVRER: Collision events for video representation and reasoning. arXiv preprint arXiv:1910.01442 (2019)
39. Zheng, Z., Yan, X., Chen, Z., Wang, J., Lim, Q.Z.E., Tenenbaum, J.B., Gan, C.: ContPhy: Continuum physical concept learning and reasoning from videos. arXiv preprint arXiv:2402.06119 (2024)
40. Zhou, Y., Simon, M., Peng, Z., Mo, S., Zhu, H., Guo, M., Zhou, B.: Simgen: Simulator-conditioned driving scene generation. In: Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., Zhang, C. (eds.) Advances in Neural Information Processing Systems. vol. 37, pp. 48838–48874. Curran Associates, Inc. (2024), https://proceedings.neurips.cc/paper_files/paper/2024/file/57c5a7c83b056d74bc97b7db36bd3649-Paper-Conference.pdf

Technical Appendix

A Dataset Details

A.1 Examples and Composition

Figure 5 shows representative examples. Table 6 gives the domain-by-difficulty composition of all 1,200 examples. Table 7 shows the distribution of governing physical laws. Counts use the primary classification, and 376 examples (140 easy / 236 hard) additionally carry a secondary domain label (e.g., rigid bodies coupled with fluid).

The "Other" category covers less frequent principles such as surface tension, buoyancy, and material phase transitions. The test split (200 examples) contains 68 rigid-body, 52 soft-body, 46 fluid, and 34 mechanics examples (116 easy / 84 hard), mirroring the overall composition.

A.2 Curation Funnel and Success Rates

The full funnel is: 50 expert seed prompts (15 held out for the test split) → 2,000 LLM-expanded candidate instructions → 1,400 instructions after human filtering (clarity, physical well-posedness, diversity) → two-stage validation with expert modification, retaining **1,120 validated examples** (80.0%, with 280 instructions discarded after three failed revision rounds). Of these 1,120 pool-derived examples, 1,000 (from training seeds) form the training set and 120 (from held-out seeds) enter the test set. The 15 held-out seeds were deliberately expanded at lower volume than the 35 training seeds, since their role is to fill a fixed-size test split rather than to maximize training data. Adding **80 expert-authored test instructions** (written, together with their reference implementations, entirely by experts without LLM involvement) yields the final **1,200 examples**. Only 430/1,400 (30.7%) of pooled candidates pass both validation stages on the first attempt. Tables 8 and 9 report stage-wise rates and intervention causes.

Fluid dynamics and mechanics required the most revision rounds on average (2.4 and 2.1) versus rigid-body (1.3) and soft-body (1.8) scenarios.

The rates in Table 8 are per instruction over the four generating models, counting an instruction as successful if any model's draft succeeds. They are therefore not comparable to the per-sample code scores of a single model in the main results table, which are lower.

Construction effort. Curation was performed by five domain experts (three with physics backgrounds, two with computer-graphics backgrounds) over approximately 1,280 person-hours: 90 for seed authoring and candidate filtering, 700 for validation and code revision, 340 for metadata, preference, and assertion annotation, and 150 for the MuJoCo-native reference implementations and ported assertion sets. The dominant cost is validation, at roughly 35 minutes per retained example, since each revision round requires re-running the simulation and inspecting the rendered result.

A.3 Generation Prompt Template

All code generation (dataset curation and evaluation alike) uses the template below. Here `{user_prompt}` is the instruction, `{genesis_documentation}` the retrieved documentation context, and `{relevant_code_examples}` example programs from the official Genesis repository.

```
You are an expert programmer specializing
in physical simulations using the Genesis
physics engine. Your task is to implement
the following physical scenario:

[INSTRUCTION]: {user_prompt}

Please generate Python code that implements
this scenario using the Genesis physics
engine. Your code should:
1. Initialize the Genesis environment with
   appropriate parameters
2. Create all necessary physical objects
   with realistic properties
3. Configure the correct physical
   interactions and constraints
4. Set up an appropriate camera angle to
   visualize the phenomenon
5. Run the simulation and save the output
   video

The code should be physically accurate,
following these laws:
- Respect conservation laws (energy,
  momentum, etc.)
- Use realistic parameters for mass,
  friction, elasticity, etc.
- Implement correct collision detection
  and response
- Apply appropriate forces and constraints

For the output specifications:
- Set the resolution to 1280x640 pixels
- Use a frame rate of 60 fps
- Generate a 5-second video
- Save the output as "genesis_video.mp4"
- Set visualization parameter to False

Here are relevant examples and
documentation to help you:
[CONTEXT]: {genesis_documentation}
[EXAMPLES]: {relevant_code_examples}
```

Domain	Easy	Hard	Total
Rigid-body physics	246	164	410
Soft-body physics	186	124	310
Fluid dynamics	154	121	275
Mechanics	123	82	205
Total	709	491	1,200

Table 6: Domain and difficulty composition of PhysCodeBench (primary classification).

Physical law	Rigid	Soft	Fluid	Mech.
Collisions	294	122	49	100
Gravity	235	201	183	121
Elasticity	145	225	31	75
Friction	239	118	58	114
Fluid dynamics	20	41	206	16
Other	44	33	54	31

Table 7: Occurrence of physical-law tags by domain (2,755 tags over 1,200 examples, mean 2.3 per example). Easy examples typically carry 1–2 tags and hard examples 3 or more, though difficulty also depends on the other criteria of Section A.4.

A.4 Annotation Guidelines and Agreement

Difficulty. Easy: 1–2 physical laws, common object interactions, standard parameter settings, single phase of matter. *Hard*: 3+ physical laws, complex interactions, precise parameter tuning, multiple phases or state transitions. Three annotators label each example. Agreement is mean pairwise Cohen's $\kappa = 0.78$.

Physical-law tags. Multi-label annotation from a fixed vocabulary. Agreement is mean pairwise Jaccard-based $\kappa = 0.71$.

Preference judgments. For 600 scenarios we collect two independent validated implementations (600 implementation pairs) and ask five domain experts for pairwise preferences, yielding 1,980 preference judgments (3.3 raters per pair on average). Because the number of raters varies across pairs, we report Krippendorff's $\alpha = 0.68$ rather than Fleiss' κ, which assumes a fixed number of raters per item. Annotators weigh physical accuracy first, then code quality, then prompt adherence. The majority label per pair yields the 600 DPO training pairs for the Simulation Refiner.

Physical assertions. During test-set annotation, experts author 1–3 executable assertions per instruction (436 total), each specifying a measurable property of the correct simulation with an explicit tolerance. Assertions are written from the instruction text alone, *blind to any candidate or reference implementation*, and each assertion is reviewed by a second expert for entailment by the instruction.

Stage (criterion)	Rate (%)
First attempt: executes without error	64.2
First attempt: passes both validation stages	30.7
After expert revision ($\leq$3 rounds): both stages	80.0

Table 8: Validation success rates over the 1,400-instruction pool. The first row uses the execution-only criterion. The latter two require both execution and simulation validation.

Issue type	Share (%)
API usage errors	42.8
Physical parameter misconfiguration	28.6
Syntax errors	15.3
Incompatible object interactions	9.2
Other implementation issues	4.1

Table 9: Issues requiring code modification or regeneration during curation.

Assertions constraining parameters the instruction leaves unspecified are loosened or the constraint is added to the instruction. Where instructions specify object counts, attributes, or spatial composition, assertions check them from engine state (e.g., "exactly four legs remain in contact with the ground"). As an oracle sanity check, the expert reference implementations pass 98.6% of all assertions and 100% of the execution and artifact checks, and score 44.1 of 50 on the perceptual axis. These four values are the oracle row of the main results table, and the perceptual score stays below 50 because CLIP similarity does not saturate even for a faithful rendering.

B PhysCodeEval Details

B.1 Code-Level Scoring

S_{exec} (25 pts): the program is executed in a sandboxed Docker environment with a 120 s timeout, and any runtime error scores 0. Error categories (syntax, physics-parameter, API misuse, resource exhaustion) are logged for diagnosis but do not affect the binary score. S_{file} (25 pts): checks existence of the required video artifact, minimum size (100 KB), and format compliance (1280×640, 60 fps, 5 s). Partial credit follows a fixed rubric weighting each requirement.

B.2 Visual Scoring

S_{clip} (25 pts): 10 evenly spaced frames are embedded with CLIP, and the mean cosine similarity s to the instruction embedding is mapped to $[0, 25]$ by $25 \cdot \text{clip}((s - s_{\min})/(s_{\max} - s_{\min}), 0, 1)$ [12]. S_{motion} (25 pts): a flow-based smoothness

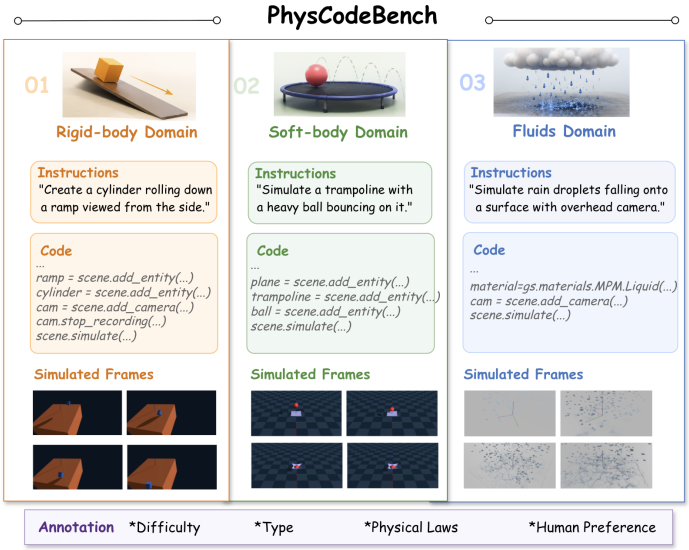

Fig. 5: Representative PhysCodeBench examples from three physical domains. Each example pairs a natural-language instruction with an expert-validated Genesis implementation, rendered simulation frames, and annotation metadata (difficulty, scenario type, governing physical laws, human preference).

measure inspired by VBench's motion-quality principle [15]. The 60 fps output video is temporally resampled to 25 fps, identically for every system so that the resampling cannot bias the comparison, and RAFT optical flow [31] is computed between consecutive frames. For each connected foreground region we take the L_2 norm of second-order velocity differences and average over regions and frames, so that scenes with more moving objects are not penalized, before mapping to $[0, 25]$. Jitter, teleportation, and interpenetration inflate the second-order term and reduce the score. In pseudocode:

```
frames = resample(video, fps=25)
flows = [RAFT(f[t], f[t+1]) for t]
j = []
for region in foreground(frames):
    v = mean_flow(region, flows)
    a = diff(v)          # acceleration
    jerk = diff(a)       # 2nd-order diff
    j.append(l2(jerk))
# lower mean jerk -> higher score
mj = mean(j)
S_motion = 25 * clip(1 - mj/J_max, 0, 1)
```

The three calibration constants $(s_{\min}, s_{\max}, J_{\max})$ are fixed once for the whole benchmark, are never tuned per system or per scenario, and ship with the evaluation toolkit so that scores are reproducible.

B.3 Direct Physical-Correctness Scoring

All checks are computed from engine state queries (positions, velocities, angular velocities of every entity at every step), never from rendered video.

Conservation checks. For scenario labels with well-defined invariants:

- *Momentum* (collision scenarios): $\|\Delta p\|/\|p_0\| \leq 0.05$ over the interaction window.
- *Free fall*: least-squares acceleration fit within 5% of g.
- *Collision energy*: kinetic-energy retention ≥ 0.95 for elastic collisions, and dissipation inside the scenario-labeled admissible interval for inelastic ones.
- *Incline dynamics*: fitted acceleration within 5% of $g(\sin\theta - \mu\cos\theta)$ for the labeled θ, μ.

For dissipative regimes (soft body, fluid) where global conservation is ill-defined, we instead verify necessary conditions: monotone energy dissipation (no spontaneous energy injection), terminal-state stability (bounded velocities at the horizon), and non-interpenetration (signed distances above a contact tolerance).

Assertion examples.

- "Ball bouncing on trampoline": first rebound height $\in [0.6h_0, 0.95h_0]$.
- "Raindrops with ripples": ≥ 3 concentric surface waves within 0.3 s of first impact.
- "Cube on incline": cube's displacement direction within 10° of the downhill direction, with no penetration below the incline plane.
- "Two-sphere elastic collision": post-impact velocities match the analytic two-body solution within 5%.
- "Four-legged table drop": exactly four legs in ground contact at rest, and the entity count matches the instruction.

Per-residual results. Table 10 breaks down the conservation subset (102 rigid-body and mechanics test cases) reported in the main paper.

B.4 Cross-Engine Protocol

For 120 of the 200 test instructions (covering all four domains, with fluid scenarios restricted to MuJoCo-supported particle approximations), experts authored MuJoCo-native reference implementations and ported the assertion sets. All PhysCodeEval axes are computed identically on MuJoCo: execution in a sandboxed MuJoCo environment, artifact checks, perceptual scores on rendered frames, and assertion checks from `mjData` state queries (on this subset, SMRF

Check	SMRF	Claude-3.5	DRDQ-32B+S+D
Momentum drift	83.7	37.5	44.8
Free-fall / incline	85.0	40.2	47.2
Energy (elastic)	76.2	28.9	35.4
Energy (inelastic)	80.7	34.6	41.4
Aggregate	**81.4**	35.3	42.2

Table 10: Pass rates (%) on trajectory-level conservation checks over the 102 rigid-body and mechanics test cases (5% tolerance). Each check is evaluated only on the cases whose scenario label defines the corresponding invariant, so the row denominators differ. Aggregate is the unweighted mean of the four checks rather than a case-level rate.

with the 250 adaptation samples passes 63.4% of ported assertions versus 20.1% for zero-shot Claude-3.5-Sonnet). Cross-engine retention percentages in the main paper use SMRF's Genesis score restricted to the same 120 instructions as the denominator. The 250 adaptation samples referenced in the main paper are API-mapping demonstrations (Genesis→MuJoCo construct correspondences) containing no new physical scenarios. The EC fix-rate comparison (84%→79%) is measured in the zero-shot-transfer condition.

C Training and Inference Details

All agents are initialized from DeepSeek-R1-Distill-Qwen-32B and fine-tuned with LoRA (rank 64, applied to attention and MLP projections) using AdamW (weight decay 0.01), learning rate 10^{-5} with cosine schedule and 100 warmup steps, batch size 2 with gradient accumulation, 5 epochs for SFT and 3 for DPO ($\beta = 0.1$), early stopping on validation loss. Training used 2×NVIDIA A100 80GB for ≈34 GPU-hours for the main configuration (cross-validation retrainings excluded). Random seed 42 is fixed for data ordering and initialization. Harvest and training scripts will be released.

At inference, long-context models (GPT-4o, Claude-3.5-Sonnet, Gemini-2.0-Pro, DeepSeek-R1) receive the full ≈100K-token Genesis documentation corpus (official examples, API reference, user guide). Models with 32K contexts (Qwen-2.5-32B, DRDQ-32B, QwQ-32B, and all SMRF agents) receive ≈20K tokens of BM25-retrieved [28] sections keyed on the instruction's entities and physical laws, plus the top-3 most similar official examples. Generation: temperature 0.1, max 4,096 new tokens. Every test instruction is attempted 5 times and we report the mean (avg@5). The same protocol applies to all baselines and SMRF. The generation cap is a real constraint for the zero-shot reasoning-style open models (DeepSeek-R1, QwQ-32B, and DRDQ-32B), whose chains of thought can consume a large share of the budget before any code is emitted. It does not affect the comparison against Claude-3.5-Sonnet or against the fine-tuned single agent, both of which emit code directly, but we flag it in Section F rather than claim it away.

Context given to Claude-3.5-Sonnet	Total
Full documentation (≈100K tokens)	35.9
Naive truncation (30K tokens)	33.8
BM25-retrieved sections (20K tokens)	34.7

Table 11: Context ablation for the documentation protocol.

Software environment. Genesis `0.2.1`, MuJoCo `3.3.2`, PyTorch `2.6.0` with CUDA `12.4`, Python `3.10`, and the official RAFT reference implementation. Every engine version is pinned, because simulation results are version-sensitive. Evaluation runs inside a fixed Docker image whose full dependency manifest ships with the toolkit.

Context ablation. To verify that the asymmetric documentation protocol neither handicaps nor favors either group, we ablate the context given to Claude-3.5-Sonnet: full 100K corpus 35.9, naive truncation to 30K 33.8, BM25-retrieved 20K 34.7. Retrieval recovers most of the full-context score, so the 32K-context models' protocol is not the driver of the SMRF–baseline gap (Table 11).

D SMRF Implementation Details

D.1 Agent Loop

Given instruction x, the SG produces c_0, which is executed in the engine. On failure with runtime feedback e_t, the EC produces $c_{t+1} = \mathrm{EC}(x, c_t, e_t)$, for at most 3 rounds, after which the framework returns failure. On success, the SR produces the final program $\hat{c} = \mathrm{SR}(x, c_t)$, adjusting physical parameters and code quality while preserving structure. Each agent is a separately fine-tuned LoRA specialization of the same backbone, so the framework's memory footprint is one backbone plus three adapters.

D.2 Error-Corrector Tetrads

The EC trains on 520 tetrads $(x, c_{\text{bug}}, e, c_{\text{fix}})$ harvested from Stage-3 validation failures, restricted to the 1,000 training instructions (no tetrad derives from any test instruction): x is the instruction, c_{bug} the failing program, e the runtime feedback (exception trace or expert-written physical diagnosis), and c_{fix} the expert-corrected program. Loss is computed only on c_{fix} tokens. Category distribution: 43% API misuse, 28% parameter miscalibration, 18% boundary conditions, 7% temporal discretization, 4% other. This taxonomy groups tetrads by the physical or programmatic failure mode that the fix addresses, and is therefore not the same partition as the intervention-cause taxonomy of Table 9, which counts why a curation example needed expert attention at all.

Three abbreviated examples:

```
# Tetrad 1 (API misuse)
# e: TypeError: add_entity() got an
#    unexpected keyword 'friction_coef'
- surface = scene.add_entity(
-     gs.morphs.Plane(),
-     friction_coef=0.5)
+ surface = scene.add_entity(
+     gs.morphs.Plane(),
+     material=gs.materials.Rigid(
+         friction=0.5))

# Tetrad 2 (parameter miscalibration)
# e: expert diagnosis: restitution 1.4
#    injects energy; ball gains height
#    every bounce
- mat = gs.materials.Rigid(
-     restitution=1.4)
+ mat = gs.materials.Rigid(
+     restitution=0.85)

# Tetrad 3 (boundary condition)
# e: fluid particles escape the domain;
#    simulation diverges at t=1.8s
- scene = gs.Scene(
-     sim_options=gs.options.SimOptions())
+ scene = gs.Scene(
+     sim_options=gs.options.SimOptions(),
+     boundary=gs.options.Boundary(
+         lower=(-2,-2,0), upper=(2,2,4)))
```

D.3 Semi-Automated Benchmark Expansion

To address the cost of expert curation, we fine-tune a prompt evaluator (accept/reject with rationale) on the Stage-1 filtering decisions over the 2,000 candidate instructions. In a held-out trial on 200 fresh candidates, routing candidates through the evaluator before expert review reduced expert filtering time by 64% at equal acceptance quality. The released toolkit packages this evaluator with the validation harness, so community contributions require one expert pass rather than full pipeline review.

E Additional Results

E.1 Performance by Physical Domain

Table 12 shows that all systems find fluid dynamics hardest (multi-phase behavior, stability constraints) and mechanics/rigid-body easiest. SMRF leads the best proprietary baseline (Claude-3.5-Sonnet) by 29.5–32.8 points and the strongest fine-tuned single agent (DRDQ-32B+SFT+DPO) by 27.9–31.2 points in every domain, so the advantage is not an artifact of one scenario family.

Model	Rigid	Soft	Fluid	Mech.
GPT-4o	37.6	30.8	27.9	38.8
Claude-3.5-Sonnet	39.9	32.9	29.9	40.8
Gemini-2.0-Pro	35.5	28.5	25.6	36.5
DeepSeek-R1	33.4	26.5	23.5	34.4
DRDQ-32B	31.5	24.6	21.7	32.5
Qwen-2.5-32B	2.7	1.2	0.7	2.9
QwQ-32B	18.3	13.4	11.4	18.9
DRDQ-32B + SFT	39.3	32.4	29.5	40.3
DRDQ-32B + SFT + DPO	41.4	34.5	31.5	42.4
SMRF (base)	37.6	30.7	27.7	38.5
SMRF + SFT	66.2	58.4	54.3	67.1
SMRF + SFT + DPO	**72.3**	**63.2**	**59.4**	**73.6**

Table 12: Total score by physical domain on the test set.

Error type	Freq. (%)	Fixed (%)
API usage errors	41	88
Parameter miscalibration	30	84
Boundary condition errors	17	79
Temporal discretization	8	75
Other	4	71

Table 13: Errors identified by the EC during SMRF inference on the test set and correction success rates. The frequency distribution is measured at inference time and differs from the training-tetrad distribution in Section D.2, which is harvested from curation-stage failures of frontier LLMs.

E.2 Error Correction Analysis

Typical EC diagnoses include mismatched API parameter names, camera configurations that miss the phenomenon, and physically inadmissible values ("restitution 1.4 injects energy" or "gravity $50\,\mathrm{m/s^2}$ unrealistic for this scenario"). Frequencies and success rates by category are in Table 13.

E.3 Single-Agent Iterative Refinement and Best-of-k

With an identical correction budget, the best single-agent self-refinement reaches 41.0, and boosting Claude-3.5-Sonnet with best-of-5 sampling (selecting by CLIPScore) reaches only 40.4 at $5\times$ the generation cost (Table 14). The gap to SMRF (+26.2 over the strongest alternative) isolates the value of *specialized* correction and refinement from mere iteration or sampling.

E.4 VLM-Based Physical Plausibility

Because GPT-4o is also an evaluated baseline, we cross-check with a second judge from a different model family (Gemini-2.0-Pro). Rankings and margins agree

Approach	Total
GPT-4o (3 corrections)	38.1
Claude-3.5-Sonnet (3 corrections)	39.8
Claude-3.5-Sonnet (best-of-5, CLIP-selected)	40.4
DRDQ-32B + SFT + DPO (3 corrections)	41.0
SMRF + SFT + DPO	**67.2**
Improvement over strongest alternative	**+26.2**

Table 14: Single-agent self-refinement (3 correction rounds, same budget as SMRF) and CLIP-selected best-of-5 sampling vs. SMRF.

	GPT-4o judge		Gemini judge	
Aspect	**SMRF**	**Cl-3.5**	**SMRF**	**Cl-3.5**
Gravity	4.6	3.8	4.5	3.7
Collision dynamics	4.4	3.5	4.2	3.5
Fluid behavior	4.1	3.2	4.1	3.1
Object motion	4.5	3.7	4.3	3.5
Temporal consistency	4.2	3.6	4.1	3.5
Overall (mean)	**4.4**	3.6	**4.2**	3.5

Table 15: VLM-rated physical plausibility (1–5) on 100 sampled test videos, 3 runs per sample (std < 0.3), under two judges. Overall = mean of the five aspects.

across judges (Table 15). We note that both judges are themselves evaluated baselines, so these ratings are supplementary evidence only: the non-circular physical check is the state-based measurement of Section B.3, which involves no model judge.

E.5 User-Study Protocol

The ten participants (six graduate students, four professional developers, all with physics-simulation experience) were recruited from the authors' institutions, participated voluntarily with informed consent, and were compensated at standard local hourly rates. Each rated anonymized, randomly ordered implementations, and raters were blind to which system produced which output. Every participant rated all 200 implementations (40 prompts $\times$ 5 systems) over four sessions of at most 90 minutes, so the design is fully crossed and ICC(2,1) applies. The metric-validity correlation in the main paper uses the best proprietary baseline, the strongest single agent, and SMRF.

E.6 Case Studies

Rigid body. For an unequal-mass elastic collision, the SG's draft used incorrect collision-response coefficients. The EC flagged the API misuse and restored the

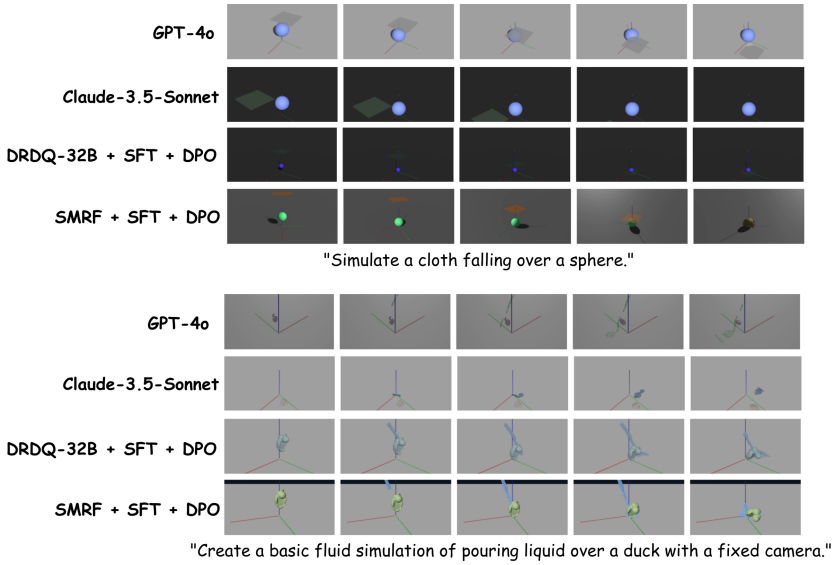

Fig. 6: Additional qualitative comparisons. Top ("cloth falling over a sphere"): baselines leave the sphere bare (cloth absent or rigid), while SMRF drapes the cloth with contact-consistent folds. Bottom ("pouring liquid over a duck"): baselines emit sparse disconnected particles, while SMRF produces a continuous stream interacting with the surface.

analytic post-impact velocities, and the SR tuned restitution so the kinetic-energy retention check passed at 0.97.

Fluid. For a rain-droplet scenario, the draft's surface-tension setting produced droplet merging without splashes. The EC identified the parameter miscalibration, and the SR's preference-aligned refinement produced ripple propagation satisfying the "$\geq$3 ripples in 0.3 s" assertion.

Soft body. For cloth draping over a sphere, the draft's stiffness caused unbounded stretching. The EC corrected the elasticity range, and the terminal-state stability check then passed with bounded strain.

E.7 Additional Qualitative Comparisons

Figure 6 shows two further test instructions with frame sequences for all four systems.

E.8 Contamination Audit Details

Semantic: each of the 200 test instructions is embedded (text-embedding-3-large) and compared to all 1,000 training instructions. Nearest-neighbor cosine

similarity has mean 0.41 and max 0.63, with zero pairs above the 0.85 near-duplicate threshold. Lexical: BM25 retrieval of the closest training instruction gives max token-level Jaccard 0.27. Code: five zero-shot samples per test prompt from each of the strongest baselines never exceed CodeBLEU [27] 0.7 against the corresponding test reference implementations, and expert rewriting during curation leaves mean CodeBLEU 0.46 between final references and original LLM drafts. Prompt-style control: on the hard subset, SMRF scores 52.1 on held-out-seed expansions vs. 53.9 on expert-authored instructions (1.8-point gap), so performance does not depend on the LLM-expansion style shared with training data. Statistical robustness: 1,000-resample bootstrap 95% CIs by domain do not overlap between SMRF and any baseline, and seed-grouped 5-fold cross-validation over the 1,120 pool-derived examples gives a total-score std of 1.4.

F Limitations

PhysCodeBench targets symbolic simulation through high-level engine APIs, not the implementation of solvers or numerical methods. Mastering the former is a prerequisite for the latter, but our scores say nothing about a model's ability to write, e.g., a stable MPM solver. Domain coverage excludes electromagnetism, thermodynamics beyond simple dissipation, and relativistic or quantum effects. The dissipative-scenario checks are necessary conditions rather than sufficient ones: a simulation can pass them while still deviating from the described phenomenon in ways the assertions do not capture. We also report conservation residuals as an aggregate only for the 102 rigid-body and mechanics cases. For the dissipative regimes the necessary conditions are checked per scenario and surface through the assertion set, but we do not summarize them as a standalone rate, since their admissible bands are scenario-specific and not comparable across scenarios. The Error Corrector is triggered only by execution failure, so semantically incorrect but crash-free programs bypass correction at inference time. The state-based checks detect (but do not yet repair) such cases, and closing this loop is future work. Cross-engine evaluation currently covers MuJoCo for 120 of 200 test cases, and fluid coverage there is approximate.

Two limits concern coverage of our own metrics. Executable assertions exist for the 200 test examples (436 assertions) and conservation residuals for the 102 rigid-body and mechanics test cases, so S_{phys} and the conservation diagnostic characterize the evaluation split rather than all 1,200 examples. Extending assertion coverage to the training split is the main outstanding curation cost.

A protocol caveat also deserves stating. All systems share a 4,096-token generation budget, which binds hardest on the zero-shot reasoning-style open models (DeepSeek-R1, QwQ-32B, and DRDQ-32B), whose chains of thought precede any code, and correspondingly on SMRF (base), which runs unspecialized DRDQ-32B agents. Those rows should be read as scores under a fixed budget rather than as ceilings for those models. The two headline comparisons are not affected: Claude-3.5-Sonnet is not a long-chain reasoning model and comfortably fits the budget, and the strongest single agent (DRDQ-32B+SFT+DPO) is

fine-tuned on the same reference implementations as the SMRF agents, so both sides of that comparison use the budget the same way.

Finally, SMRF triples inference cost relative to a single agent, which matters for interactive use.

G LLM Usage Statement

LLMs were used in three declared roles. (1) *Dataset construction*: GPT-4o, Claude-3.5-Sonnet, Gemini-2.0-Pro, and GitHub Copilot expanded the 50 expert seed prompts into 2,000 candidates and drafted candidate implementations. Every retained example passed two-stage validation and expert rewriting (Section A.2), and 80 test instructions were authored entirely without LLMs. (2) *Evaluation subjects*: three of the four construction models (GPT-4o, Claude-3.5-Sonnet, Gemini-2.0-Pro) are also evaluated as baselines, alongside open-source models. (3) *Writing*: language polishing only. All technical content, experiments, and analyses are by the authors.