WORLDSOLVER: CAN LLM AGENTS SIMULATE THE PHYSICAL DYNAMICS VIA SOLVER GENERATION?

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based agents are increasingly advancing scientific and engineering problem solving, with physics simulation emerging as a challenging yet practical testbed for reproducing complex physical phenomena with application in embodied AI, games and films. As the workhorse of such simulation, a solver computes how the state of a dynamic system evolves over time. Building such solvers requires physical understanding to identify appropriate models, mathematical reasoning to formulate the underlying dynamics, and software engineering to implement them as executable code, yet this capability of LLM agents remains underexplored. To this end, we introduce **WorldSolver**, a benchmark of 168 simulation tasks derived from physical phenomena in 61 classic computer graphics papers, spanning 7 physical domains. Each task contains a code scaffold that provides a fixed simulation environment for the scene, with the solver implementation left for the agent to complete. Specifically, we evaluate them along three dimensions: **Execution Checks** for successful execution, **Visual Fidelity** for reproducing the intended dynamic behavior in the rendered simulation, and **Physical Plausibility** for physics-grounded verification of the generated dynamics. Experiments on frontier agents reveal that producing executable solvers is difficult itself, and satisfying visual and physical correctness is even harder. GPT-5.6-Sol and Claude-Opus-5 perform comparatively better than the other evaluated agents, yet achieve overall scores of only 48.7% and 46.7%, respectively. WorldSolver is an early step toward agentic solver generation, and we hope it helps drive progress toward agents that can faithfully simulate the dynamic physical world.

1 INTRODUCTION

Large language model (LLM) agents are increasingly capable of tackling complex real-world tasks (Huang et al., 2024b; Guo et al., 2024), spanning from software engineering (Jimenez et al., 2024; Yang et al., 2024; Chi et al., 2026; Luo et al., 2026) to scientific discovery (Huang et al., 2024a; Majumder et al., 2024; Somasekharan et al., 2026). Recent studies have shown growing interest in physically realistic simulation of 3D worlds, with broad applications in embodied AI, games, and films (Liu et al., 2026; Wang et al., 2025). A line of work focuses on static scene construction through asset composition and spatial layout (Hu et al., 2024; Xia et al., 2025). Another explores dynamic scene generation to reproduce physical phenomena and their temporal evolution (Zhang et al., 2026; Lv et al., 2026; Xie et al., 2026; Wang et al., 2026).

As the workhorse of such simulation, *solver translates the governing equations of a physical model into a numerical procedure that evolves the simulated system over time* (Baraff & Witkin, 2023; Stam, 1999). Building them requires physical understanding to identify appropriate models, mathematical reasoning to formulate the underlying dynamics, and software engineering to implement them as executable code (Guendelman et al., 2003). For example, cloth solver handles stiff elastic dynamics and frictional contact at high degrees of freedom (Li et al., 2018), while fluid solver instead focuses on incompressibility enforcement and free-surface tracking (Enright et al., 2002).

However, existing works target dynamics scene generation either produce animations in a solver-free manner (Wang et al., 2026) or rely on existing simulation engine (Zhang et al., 2026; Xie et al., 2026; Liu et al., 2026). In contrast, scientific-computing benchmarks evaluate numerical solver generation for prescribed governing equations, where the physical model to be solved is given in

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.

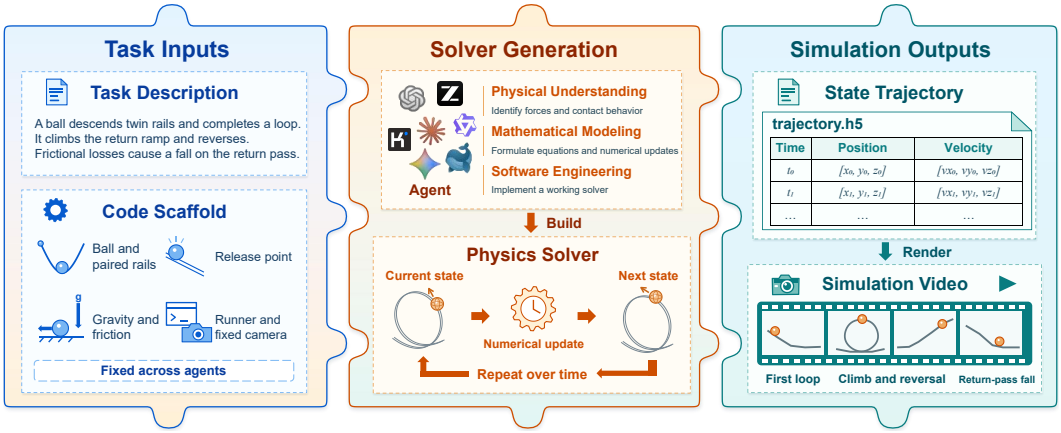


Figure 1: Example of a WorldSolver task. Given a task description and a fixed code scaffold, an agent constructs a physics solver leveraging physical understanding, mathematical modeling, and software engineering. The solver advances the system state over time, producing a state trajectory that is rendered into a simulation video. Scene geometry, physical inputs, and rendering settings remain fixed across agents.

advance (Li et al., 2026; Hang et al., 2026; Somasekharan et al., 2026). Neither setting evaluates the full capability required for physical dynamics simulation, spanning physical understanding for model selection, mathematical reasoning for numerical formulation, and software engineering for executable implementation. Together, they motivate our question:

Can LLM agents turn described physical phenomena into dynamics simulation solvers?

In this work, we present **WorldSolver**, a benchmark for evaluating LLM agents' ability to generate solvers for physical dynamics simulation tasks. As shown in Figure 1, WorldSolver isolates solver generation with a fixed code scaffold and evaluates the generated physical dynamics through the state trajectory and rendered video outputs. To ensure a challenging yet physically realistic setting, we curate physical phenomena from classic computer graphics literature and construct simulation tasks around them (Guendelman et al., 2003; Kaufman et al., 2005; Li et al., 2018). Given a description of the target phenomenon, the agent implements a solver within a fixed code scaffold that specifies the simulation environment, including scene geometry, physical parameters, external inputs, and rendering configuration. Once executed, *the solver produces not only a rendered video but also a state trajectory that exposes the underlying physical dynamics through evolving quantities over time*, such as position, velocity, and forces.

Therefore, we evaluate each solver through three protocols: Execution Checks for successful and complete execution, Visual Fidelity for reproducing the intended dynamic behavior in the rendered video, and Physical Plausibility for physically-grounded verification of the generated trajectory. Table 1 summarizes how WorldSolver differs from existing benchmarks in task formulation and evaluation. Evaluation on the WorldSolver benchmark shows that generating executable solvers from described physical phenomena remains challenging even for frontier coding agents. GPT-5.6-Sol and Claude-Opus-5 perform best, yet reach overall scores of only 48.7% and 46.7%, while most other agents hover around 20%. Our contributions are threefold:

- To the best of our knowledge, we are the first to formulate physical dynamics solver generation as an agentic coding task, where agents' capabilities in physical understanding, mathematical formalization and software engineering are put to the test.
- We introduce **WorldSolver**, comprising 168 tasks derived from physical phenomena in 61 classic computer graphics papers across seven domains, with fixed task-specific scaffolds that isolate solver implementation.
- We propose an evaluation protocol combining execution checks, rendered-simulation assessment, and physics-grounded trajectory verification, benchmarking frontier coding agents under this unified protocol.

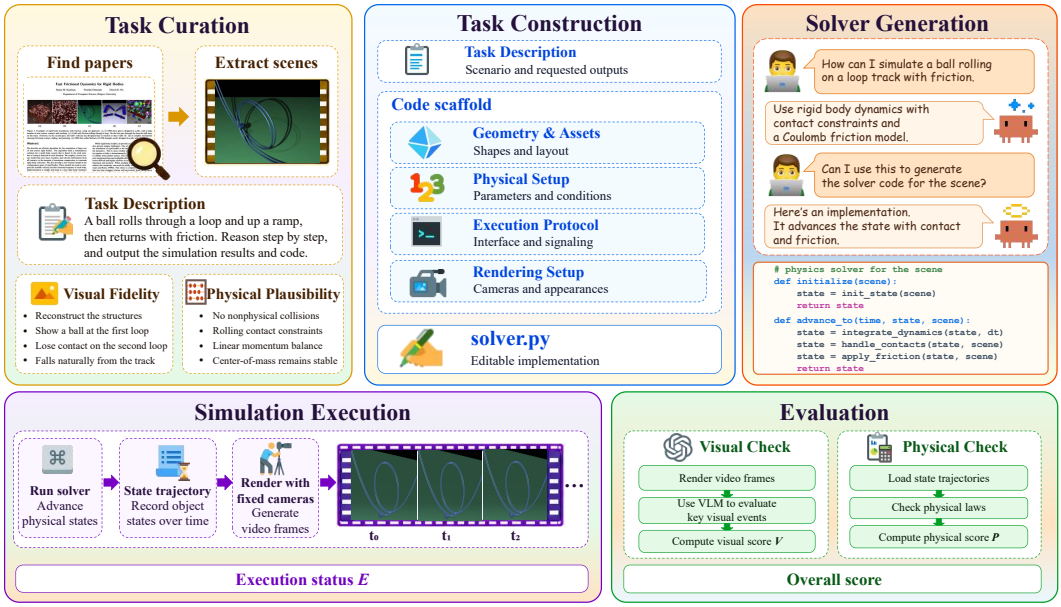


Figure 2: Overview of the WorldSolver workflow. We curate physical phenomena from classic computer graphics papers and build a simulation task around each one. Every task supplies a phenomenon description and a code scaffold, and the agent implements the solver. Running the solver produces state trajectories and rendered videos, which are then assessed by execution, visual, and physical checks.

2 WorldSolver

WorldSolver is a benchmark that evaluates how well coding agents can generate solvers for physical simulation. Each task is derived from a challenging yet physically realistic phenomenon, curated from classic computer graphics papers spanning seven physical domains. Overview of the WorldSolver workflow is provided in Figure 2.

2.1 Task Formulation

Each task provides a description d of a target physical phenomenon and a fixed code scaffold

$$\mathcal{S} = (\mathcal{G}, \mathcal{P}, \mathcal{E}, \mathcal{R}), \tag{1}$$

where $\mathcal{G}$ specifies scene geometry and assets, $\mathcal{P}$ specifies physical parameters, external inputs, and boundary conditions, $\mathcal{E}$ defines the execution protocol and solver interface, and $\mathcal{R}$ specifies the rendering configuration. The scaffold provides the infrastructure for executing and rendering a simulation. We then formulate physics solver generation as a task in which an agent A develops a solver Φ from a description d of a target phenomenon and a fixed code scaffold $\mathcal{S}$:

$$\Phi = \mathcal{A}(d, \mathcal{S}). \tag{2}$$

A solver Φ defines how the state of a simulated system evolves over time. Let $\mathbf{s}_t$ denote the physical state at step t, and let $\mathbf{s}_0$ be the initial state defined by the scaffold. Executed under the scaffold's protocol, Φ advances each state to the next one over a horizon of T steps:

$$\mathbf{s}_{t+1} = \Phi(\mathbf{s}_t), \qquad t = 0, \ldots, T-1. \tag{3}$$

Each state $\mathbf{s}_t$ exports necessary variables for rendering and quantities required by physical checks, such as velocities and contact forces. Finally, each run produces two outputs for evaluation. Iterating state $\mathbf{s}_T$ collects a state trajectory τ, and passing the trajectory through the fixed renderer yields a video v.

$$\tau = (\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_T), \; v = \mathcal{R}(\tau). \tag{4}$$

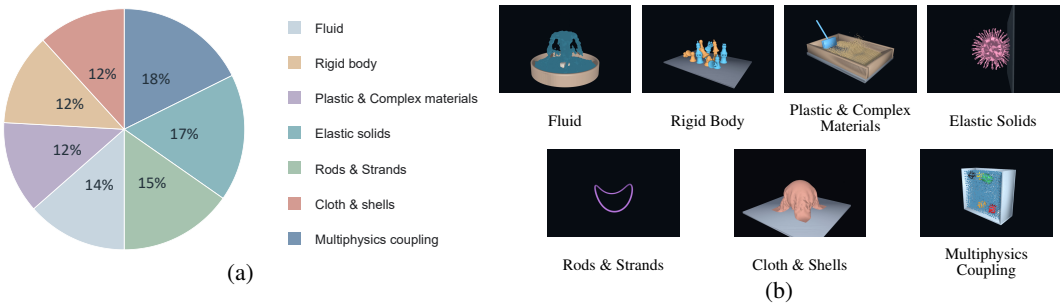


Figure 3: Benchmark coverage across seven physics domains. (a) Distribution of the 168 tasks across domains. (b) Representative simulation cases from each domain.

Table 1: Comparison with related benchmarks. WorldSolver covers seven physics domains and supports both visual and numerical evaluation of generated solvers. Notably, WorldSolver is the first to use a fixed code scaffold to isolate solver implementation under controlled conditions.

Benchmark	Domain Coverage	Code Scaffold	Solver Generation	Visual Evaluation	Numerical Evaluation
SimuScene (Wang et al., 2026)	Physics (5 domains)	×	×	✓	×
PhysCodeBench (Xie et al., 2026)	Physics (3 domains)	×	×	✓	×
SimulCost (Cao et al., 2026)	Physics (3 domains)	×	×	×	✓
CFDLLMBench (Somasekharan et al., 2026)	CFD (3 tasks)	×	✓	×	✓
PDEAgent-Bench (Hang et al., 2026)	PDEs (11 families)	×	✓	×	✓
WorldSolver	**Physics (7 domains)**	✓	✓	✓	✓

2.2 Task Construction

Task Curation. We curate our tasks from challenging yet physically realistic phenomena presented in classic computer graphics papers (Guendelman et al., 2003; Li et al., 2018; Enright et al., 2002). For each phenomenon, we write a task description together with a set of scoring criteria, the latter withheld from the agent during solver generation.

Domain Coverage. Our benchmark covers seven domains, including fluid (Jiang et al., 2015; De Goes et al., 2015), rigid body (Kaufman et al., 2005; Ferguson et al., 2021), plastic and complex materials (Klár et al., 2016; Stomakhin et al., 2013), elastic solids (Li et al., 2020; Irving et al., 2007), rods and strands (Bergou et al., 2008; Bertails et al., 2006), cloth and shells (Li et al., 2021; Bridson et al., 2002), and multiphysics coupling (Batty et al., 2007; Guendelman et al., 2005). Figure 3 shows how the tasks distribute across these domains, along with a representative case from each.

Task Specification. We turn each description into a fixed code scaffold $\mathcal{S} = (\mathcal{G}, \mathcal{P}, \mathcal{E}, \mathcal{R})$ automatically with manual checks. The geometry $\mathcal{G}$ and the physical setup $\mathcal{P}$ define what is simulated, from object shapes and arrangements to physical parameters and boundary conditions. The execution protocol $\mathcal{E}$ and the rendering setup $\mathcal{R}$ define how it is run and observed, fixing the solver interface and trajectory format on one side and the cameras and appearance on the other. Each agent receives the task prompt d and the same scaffold $\mathcal{S}$ for a given task, with modifications restricted to solver code so that modeling and numerical choices and code engineering are evaluated under controlled conditions.

2.3 Evaluation Protocol

We evaluate submitted solvers along three axes: Execution Checks, Visual Fidelity, and Physical Plausibility. Together, they assess whether a solver runs successfully, reproduces the expected visible phenomena, and respects the relevant physical laws and constraints, with the visual and physical evidence serving as complementary views of simulation quality.

Table 2: Four dimensions of visual fidelity, evaluated using task-specific criteria. The visual fidelity score is the mean score across the four dimensions.

Dimension	Evaluation Focus
Event Sequence	Occurrence and ordering of the key events specified in the task
Interaction Response	Visible response of objects and materials to interactions
Signature Phenomenon	Reproduction of the characteristic physical phenomenon described in the task
Temporal Coherence	Continuity and consistency of motion throughout the simulation

Execution Checks. We first run each solver under the code scaffold $\mathcal{S}$, setting the execution score $E = 1$ if it completes successfully and produces valid output and $E = 0$ otherwise. For successful executions, the resulting state trajectory τ and rendered video v are passed to the physical and visual evaluations described below.

Visual Fidelity. Visual Fidelity V measures how well the rendered simulation reproduces the task's target phenomena. For each task, we specify a set of visual criteria $\mathcal{C} = \{c_i\}_{i=1}^{4}$ across the four shared dimensions shown in Table 2. We use a Vision-Language Model (VLM) to score each criterion c_i from sampled key frames of the rendered video v, and the Visual Fidelity score is their average:

$$V_i = \text{VLM}(v, c_i) \in [0, 1], \qquad i = 1, \ldots, 4, \tag{5}$$

$$V(v; \mathcal{C}) = \frac{1}{4} \sum_{i=1}^{4} V_i. \tag{6}$$

To validate the VLM judge, we evaluate its stability and compare its assessments with those of human experts, as further discussed in Appendix D.

Physical Plausibility. A solver outputs the physical state at every time step, enabling continuous verification beyond the discrete observations used for visual evaluation. Specifically, Physical Plausibility P verifies whether the generated trajectory τ respects relevant physical laws and constraints. For each physical domain, we predefine a set of violation functions $\mathcal{E}_i$ for laws, *such as mass conservation, momentum–impulse consistency, and contact nonpenetration.* For each case, we define a task-specific rule set $\mathcal{R} = \{r_i\}_{i=1}^{M}$ based on the relevant physical properties of the simulated phenomenon, with M typically ranging from 3 to 5. Required output properties are specified in the solver-generation prompt, while the physical laws used for evaluation remain hidden from the agent. Each rule r_i is evaluated by a calculator $\mathcal{E}_i$ that computes a nonnegative, dimensionless violation error e_i:

$$e_i = \mathcal{E}_i(\tau) \geq 0, \qquad i = 1, \ldots, M. \tag{7}$$

The calculator normalizes the raw physical residual and aggregates it over time. Thus, $e_i = 0$ indicates no measured violation, while larger values indicate stronger violations. Each violation error e_i is then mapped to a criterion score $p_i \in [0, 1]$ using the calculator-specific upper bound, thus higher scores indicate better physical plausibility. The overall Physical Plausibility score $P \in [0, 1]$ is defined as:

$$P(\tau; \mathcal{R}) = \frac{1}{M} \sum_{i=1}^{M} p_i. \tag{8}$$

Final Score. Visual Fidelity can expose observable failures, but its reliance on discrete key frames may miss errors in the continuous evolution of the system. Physical Plausibility reveals hidden dynamics from continuous trajectories, but no finite set of metrics can capture every relevant physical rule. The two therefore serve as complementary evaluation protocols, and we combine their scores for a more complete assessment. A representative case study illustrating this complementarity is provided in the Appendix C. For case k, let $E_k \in \{0, 1\}$ denote execution success, and let V_k and P_k denote the Visual Fidelity and Physical Plausibility scores. The case score s_k and final

Table 3: Performance across seven physics domains. Overall denotes the mean score across all cases. Best and second-best results are highlighted in each column. All scores are reported on a percentage scale for clarity.

Model	Fluid	Rigid Body	Plastic & Complex Materials	Elastic Solids	Rods & Strands	Cloth & Shells	Multiphysics Coupling	Overall
GPT-5.6 Sol (OpenAI, 2026)	**49.6**	**54.8**	31.2	52.6	**59.5**	35.8	**51.5**	**48.7**
Claude-Opus-5 (Anthropic, 2026)	48.1	44.4	**47.9**	**61.9**	33.5	**60.1**	34.9	46.7
Gemini-3.7-Flash (Google DeepMind, 2026)	35.8	38.9	26.5	29.5	34.0	9.5	28.4	29.3
DeepSeek-V4.1-Flash (DeepSeek-AI, 2026)	19.9	31.1	32.1	20.7	31.0	30.0	13.0	24.6
GLM-5.3 (Zhipu AI, 2026)	16.9	30.7	33.9	23.2	31.9	27.9	7.0	23.6
Qwen-3.8-Max (Qwen Team, 2026)	19.8	32.0	18.3	18.6	23.9	30.1	19.7	22.7
Kimi-K2.7-Code (Moonshot AI, 2026)	9.7	34.0	20.1	15.4	19.0	12.8	12.3	17.1

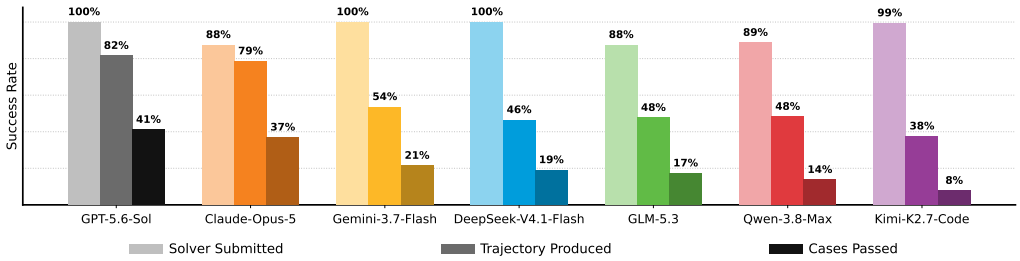


Figure 4: Success rates across three stages of solver generation. Solver Submitted measures successful solver delivery, Trajectory Produced measures successful execution with valid trajectory outputs, and Cases Passed measures the fraction of tasks achieving a final score of at least 60 points.

benchmark score are defined as:

$$s_k = \begin{cases} \dfrac{V_k + P_k}{2}, & E_k = 1, \\ 0, & E_k = 0, \end{cases} \qquad k = 1, \ldots, K, \tag{9}$$

$$\text{Score} = \frac{1}{K} \sum_{k=1}^{K} s_k, \qquad K \geq 1. \tag{10}$$

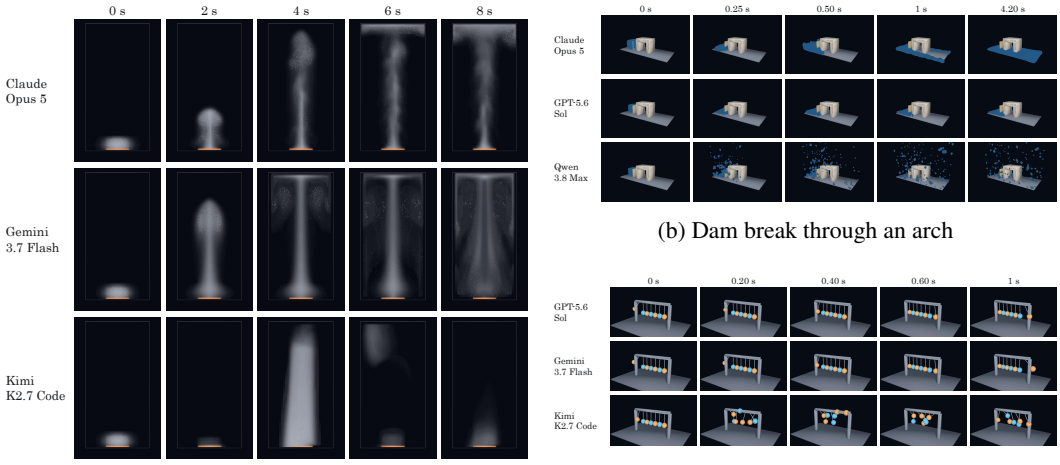
3 Experiment

3.1 Experimental Setup

We evaluate seven frontier models on the WorldSolver benchmark: GPT-5.6-Sol (OpenAI, 2026), Claude-Opus-5 (Anthropic, 2026), Gemini-3.7-Flash (Google DeepMind, 2026), DeepSeek-V4.1-Flash (DeepSeek-AI, 2026), GLM-5.3 (Zhipu AI, 2026), Qwen-3.8-Max (Qwen Team, 2026), and Kimi-K2.7-Code (Moonshot AI, 2026). We use Codex CLI as the agent scaffold for GPT-5.6-Sol and Kimi Code for Kimi-K2.7-Code, while all other models are evaluated with Claude Code. For each model, we use the reasoning-effort setting of "high" when available, models without explicit levels are evaluated with thinking mode "enabled". We set a generation time limit of 1,800 seconds for each agent and a runtime limit of 600 seconds for each generated solver. We use GPT-5.6-Sol (OpenAI, 2026) as the VLM judge and sample 20 key frames from each rendered video for evaluation. For clarity, all benchmark scores are reported on a 0–100 scale.

3.2 Main Results

Overall performance. Table 3 shows that physical dynamics solver generation remains challenging for the evaluated agents. GPT-5.6-Sol achieves the highest overall score of 48.7%, followed



(a) Buoyant smoke

(b) Dam break through an arch

(c) Newton cradle breakaway

Figure 5: Temporal comparison of selected generated solvers on three physical tasks. Within each subfigure, rows identify models and columns show matched video timestamps. The sequences illustrate differences in plume development, flow around an obstacle, and motion following impact. Each case uses a fixed crop across models and timestamps.

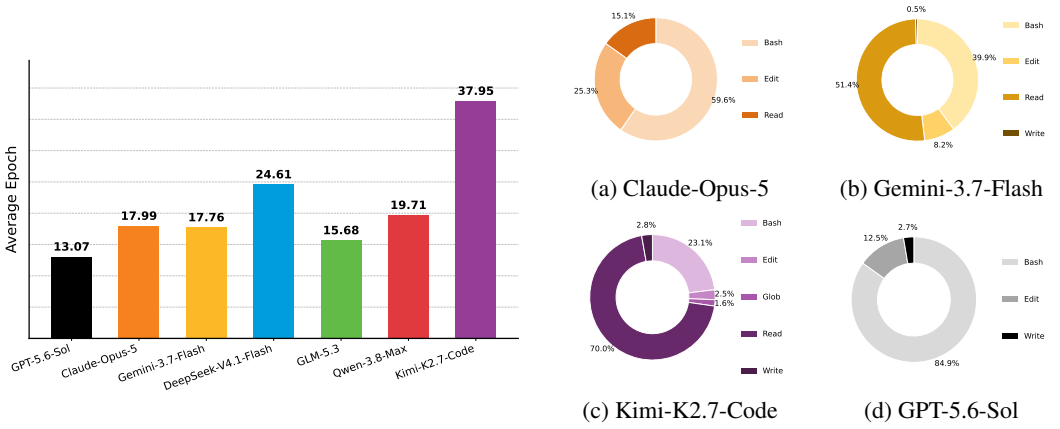
closely by Claude-Opus-5 at 46.7%. Gemini-3.7-Flash scores 29.3%, while DeepSeek-V4.1-Flash, GLM-5.3, Qwen-3.8-Max, and Kimi-K2.7-Code obtain 24.6%, 23.6%, 22.7%, and 17.1%, respectively. The performance gaps across models are substantial, yet none of the evaluated agents demonstrates consistently strong solver-generation performance. We also provide detailed Visual Fidelity and Physical Plausibility results for each agent across all domains in the Appendix D.

Domain-level performance. Each agent shows substantial performance variation across different physical domains. GPT-5.6-Sol performs best on Fluid, Rigid Body, Rods and Strands, and Multiphysics Coupling, whereas Claude-Opus-5 leads on Plastic and Complex Materials, Elastic Solids, and Cloth and Shells. Although their overall scores differ by only 2 points, their domain-level gaps are much larger: GPT-5.6-Sol outperforms Claude-Opus-5 by 26% on Rods and Strands, while Claude-Opus-5 leads by 24.3% on Cloth and Shells. Several agents also show weaknesses in specific domains, for example, Gemini-3.7-Flash scores only 9.5% on Cloth and Shells, and GLM-5.3 scores 7.0% on Multiphysics Coupling.

3.3 More Analysis

Success rate across different stages. Generating a reliable solver for physical dynamics simulation remains challenging: most agents return a submission, fewer produce valid simulation outputs, and fewer still pass the benchmark threshold. As shown in Figure 4, *Solver Submitted* denotes the fraction of tasks with a solver returned in time, *Trajectory Produced* the fraction whose solver successfully exports both a trajectory and a rendered video, and *Cases Passed* the fraction achieving a final score of at least 60%. We use 60 as a preliminary threshold to illustrate the stage-wise trend, rather than as an established criterion for solver success. All models submit solvers for most tasks, with rates ranging from 88% to 100%. However, substantially fewer solvers produce valid outputs, from 82% for GPT-5.6-Sol to 38% for Kimi-K2.7-Code. Passing the benchmark threshold is harder still. GPT-5.6-Sol and Claude-Opus-5 pass 41% and 37% of cases, compared with 14% for Qwen-3.8-Max and 8% for Kimi-K2.7-Code. The largest separation appears at the execution stage, where GPT-5.6-Sol and Claude-Opus-5 reach 82% and 79%, while the remaining models range from 38% to 54%. This gap is most striking for Kimi-K2.7-Code, which submits a solver for 99% of tasks but produces valid simulation outputs for only 38%.

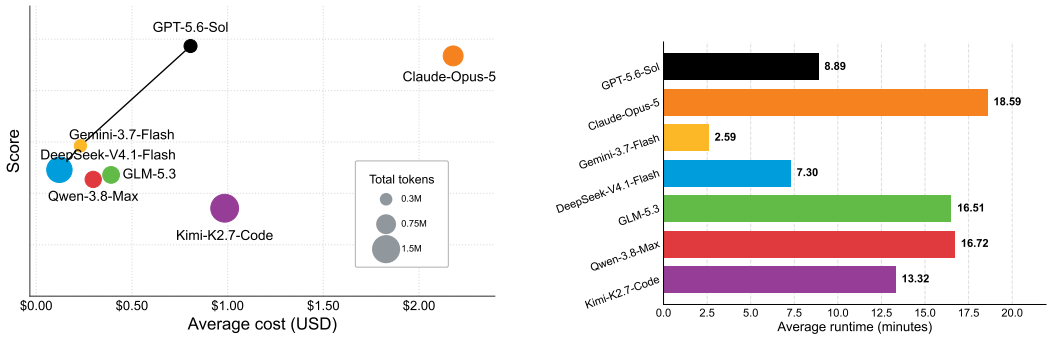
Visualization of generated solver dynamics. Figure 5 shows the temporal evolution of representative solver across three physical dynamics simulation tasks at matched video timestamps. In the buoyant smoke example (a), Claude-Opus-5 produces a visually natural rising plume with irregular billowing structures that spread beneath the upper boundary. Gemini-3.7-Flash develops a



(a) Claude-Opus-5 (b) Gemini-3.7-Flash

(c) Kimi-K2.7-Code (d) GPT-5.6-Sol

Figure 6: Mean epochs and tool-use distributions across the evaluated models. ***Left:*** Mean model-call epochs, GPT-5.6-Sol averages only 13.07 epochs per task while achieving the highest score. ***Right:*** Tool-use distributions across 4 models, Claude-Opus-5 and GPT-5.6-Sol devote more tool-use to editing.



(a) Score versus average API cost. (b) Average wall time.

Figure 7: Comparison of model score against runtime and API cost. (a) Score versus average API cost, the line represents best cost effectiveness. GPT-5.6-Sol, Gemini-3.7-Flash and DeepSeek-V4.1-Flash demonstrate highest cost effectivenesses. (b) Average agent wall time by model. Gemini-3.7-Flash reports the shortest wall time (2.59 minutes per task on average) while achieving third highest score.

more symmetric column with visibly separated particle structures, while Kimi-K2.7-Code shows a transient rise followed by substantial fading, rather than sustained plume development. In the dam-break example (b), Claude-Opus-5's fluid spreads around the arch and into the downstream region, whereas GPT-5.6-Sol's fluid remains largely upstream over the displayed interval. Qwen-3.8-Max instead exhibits extensive airborne particle dispersion, without a comparably continuous body of flowing liquid. In the Newton cradle example (c), GPT-5.6-Sol and Gemini-3.7-Flash show the initially displaced end ball approaching the row and the opposite end ball subsequently swinging outward, with relatively little interior motion. Kimi-K2.7-Code exhibits large displacements of multiple interior balls, departing from the intended impulse-transfer behavior.

Agent behavior patterns in solver generation. Figure 6 shows differences in average epoch counts and tool usage. GPT-5.6-Sol uses the fewest model-call epochs on average of 13.07, while achieving the highest benchmark score. However, Kimi-K2.7-Code requires 37.95 epochs, nearly three times as many, yet performs substantially worse. These results show that more model calls do not consistently yield better solvers. Figure 6 (a) to (d) shows that Claude-Opus-5 and GPT-5.6 Sol allocate 25.3% and 12.5% of their tool calls to Edit, compared with 8.2% for Gemini-3.7-Flash and 2.5% for Kimi-K2.7-Code. This pattern suggests a potential association between a greater emphasis on code revision and better solver performance.

Cost effectiveness. Figure 7 compares benchmark performance with average API cost and wall time. GPT-5.6-Sol achieves the highest score at a lower API cost than Claude-Opus-5, while requiring less than half its average wall time (8.89 versus 18.59 minutes), which is consistent with the mean model-call epochs data. Gemini-3.7-Flash offers a cheaper and much faster alternative, with the shortest average wall time of 2.59 minutes, although its score is lower. DeepSeek-V4.1-Flash achieves the best cost-effectiveness, but accompanied by high token usage and longer wall time. Conversely, Kimi-K2.7-Code incurs a higher API cost and longer wall time than GPT-5.6-Sol despite weaker performance, which is related to its overthinking tendency.

4 Related Work

Code Generation Benchmarks. Code generation aims to translate natural-language specifications into executable programs (Chen et al., 2021; Austin et al., 2021; Hendrycks et al., 2021; Li et al., 2022). As an early milestone, HumanEval (Chen et al., 2021) focuses on generating standalone Python functions and evaluates their correctness through execution. With the increasing capability of LLM-based agents, recent work has moved beyond function-level generation toward more complex repository-level tasks. One line of work extends code generation to complex software development tasks, including software engineering (Jimenez et al., 2024; Miserendino et al., 2025; Yang et al., 2024) and game development (Jiang et al., 2026; Chi et al., 2026; Luo et al., 2026). Another line extends code generation to scientific tasks, including implementing scientific ideas (Huang et al., 2024a; Majumder et al., 2024; Chen et al., 2025) and developing computational tools for scientific computing (Li et al., 2026; Hang et al., 2026; Somasekharan et al., 2026). WorldSolver bridges these two directions by generating physics simulation solvers for described phenomena. On the one hand, it requires agents to develop and modify physics simulation systems at the repository level. On the other hand, agents must use physical understanding to select appropriate models and mathematical reasoning to formulate numerical methods.

Autonomous Physics Simulation. Constructing a physical scene begins with specifying its static configuration, including object geometry, material properties, and initial and boundary conditions. Thanks to the development of LLM models, works in autonomous physics simulation based on LLM agents have emerged recently. SimuScene evaluates code-generated animations through visible behavior (Wang et al., 2026), while PhysCodeBench, GS-Agent, and PhysAgent construct dynamic simulations using pre-built physics engines (Xie et al., 2026; Zhang et al., 2026; Lv et al., 2026). Neither setting directly evaluates the full process of selecting physical models and implementing solvers for diverse graphics scenes under fixed experimental conditions. CodePDE, CFD-CodeBench, and PDEAgent-Bench study solver generation for prescribed equations, evaluating solution accuracy, convergence, or computational efficiency (Li et al., 2026; Somasekharan et al., 2026; Hang et al., 2026). These settings primarily focus on scientific computing, without specific physical scenarios or rendering environments, leaving agents' physical understanding and mathematical modeling abilities largely unevaluated. WorldSolver instead fixes the scene conditions and requires agents to both select physical models and implement numerical solvers without a prebuilt physics engine. Its evaluation combines visible behavior with trajectory-based physical checks across seven domains.

5 Conclusion

We introduced WorldSolver, a benchmark for evaluating whether LLM agents can translate physical phenomena description into executable physics solvers. It comprises 168 tasks drawn from 61 computer graphics papers across seven physical domains. Under fixed code scaffolds, agents implement solvers whose outputs are assessed through execution checks, visual fidelity, and physical plausibility. Our experiments show that reliable solver generation remains highly challenging: many agents struggle to submit a runnable solver within the time budget, and even fewer produce solvers that are both visually faithful and physically plausible. WorldSolver provides a testbed for studying physically grounded computation through agent-generated solvers.

Limitations. WorldSolver covers seven physical domains with 168 tasks, but still represents only a subset of physical phenomena. In the future, we can expand this coverage and explore training methods for physically grounded solver generation.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic datasets and polish our manuscript to improve the readability. We have not used generative AI tools for help develop theoretical models or conceptual frameworks, design or provide feedback on research methodology or experiments, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis, interpret results. Formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, implement methods are not applicable to this work.

We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The main paper describes the benchmark pipeline and evaluation protocol. The Appendix provides detailed descriptions of benchmark construction and evaluation. The supplementary material includes the task scaffolds, generated solvers, evaluation results, and code for reproducing the evaluation.

REFERENCES

Anthropic. Claude Opus 5 model. https://www.anthropic.com/, 2026. Model documentation and release page.

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

David Baraff and Andrew Witkin. *Large Steps in Cloth Simulation*. 1 edition, 2023. URL https://doi.org/10.1145/3596711.3596792.

Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. In *SIGGRAPH 2007*, pp. 100–es, 2007.

Miklós Bergou, Max Wardetzky, Stephen Robinson, Basile Audoly, and Eitan Grinspun. Discrete elastic rods. In *SIGGRAPH 2008*, pp. 1–12. 2008.

Florence Bertails, Basile Audoly, Marie-Paule Cani, Bernard Querleux, Frédéric Leroy, and Jean-Luc Lévêque. Super-helices for predicting the dynamics of natural hair. *TOG*, 25(3):1180–1187, 2006.

Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *SIGGRAPH*, pp. 594–603, 2002.

Yadi Cao, Sicheng Lai, Jiahe Huang, Yang Zhang, Zach Lawrence, Rohan Bhakta, Izzy F Thomas, Mingyun Cao, Chung-Hao Tsai, Zihao Zhou, et al. Simulcost: A cost-aware benchmark and toolkit for automating physics simulations with llms. In *ICML*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, et al. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *ICLR*, pp. 96934–96990, 2025.

Wayne Chi, Yixiong Fang, Arnav Yayavaram, Siddharth Yayavaram, Seth Karten, Qiuhong Anna Wei, Runkun Chen, Alexander Wang, Valerie Chen, Ameet Talwalkar, and Chris Donahue. Gamedevbench: Evaluating agentic capabilities through game development. In *ICLR*, 2026.

Fernando De Goes, Corentin Wallez, Jin Huang, Dmitry Pavlov, and Mathieu Desbrun. Power particles: an incompressible fluid solver based on power diagrams. *TOG*, 34(4):50–1, 2015.

DeepSeek-AI. DeepSeek-V4.1-Flash model. https://www.deepseek.com/, 2026. Model documentation and release page.

Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. In *SIGGRAPH*, pp. 736–744, 2002.

Zachary Ferguson, Minchen Li, Teseo Schneider, Francisca Gil-Ureta, Timothy Langlois, Chenfanfu Jiang, Denis Zorin, Danny M. Kaufman, and Daniele Panozzo. Intersection-free rigid body dynamics. *TOG*, 40(4), July 2021. URL https://doi.org/10.1145/3450626.3459802.

Google DeepMind. Gemini 3.7 Flash model. https://deepmind.google/models/gemini/, 2026. Model documentation and release page.

Eran Guendelman, Robert Bridson, and Ronald Fedkiw. Nonconvex rigid bodies with stacking. *TOG*, 22(3):871–878, 2003.

Eran Guendelman, Andrew Selle, Frank Losasso, and Ronald Fedkiw. Coupling water and smoke to thin deformable and rigid shells. *TOG*, 24(3):973–981, 2005.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. In *IJCAI*, pp. 8048–8057, 8 2024. URL https://doi.org/10.24963/ijcai.2024/890. Survey Track.

Zhen Hang, Yushan Yashengjiang, Junhui Li, Huanshuo Dong, Yang Wei, Zhezheng Hao, Jiangtao Ma, Songlin Bai, Haozhong Kai, Xihang Yue, Gangzong Si, Dongming Jiang, Chao Yao, Zhanhua Hu, Jiangqing Zhang, Pengwei Liu, Yaomin Shen, Xingyu Ren, Lei Liu, Zikang Xu, Han Li, Qingsong Yao, Hande Dong, and Hong Wang. Pdeagent-bench: A multi-metric, multi-library benchmark for pde solver generation, 2026. URL https://arxiv.org/abs/2605.09636.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. In *NeurIPS*, 2021.

Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A. Ross, Cordelia Schmid, and Alireza Fathi. Scenecraft: An llm agent for synthesizing 3d scene as blender code. In *ICML*, 2024.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: evaluating language agents on machine learning experimentation. In *ICML*. JMLR.org, 2024a.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey, 2024b. URL https://arxiv.org/abs/2402.02716.

Geoffrey Irving, Craig Schroeder, and Ronald Fedkiw. Volume conserving finite element simulations of deformable models. *TOG*, 26(3):13–es, 2007.

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *TOG*, 34(4):1–10, 2015.

Yilei Jiang, Jinyuan Hu, Qianyin Xiao, Yaozhi Zheng, Ruize Ma, Kaituo Feng, Jiaming Han, Tianshuo Peng, Kaixuan Fan, Manyuan Zhang, and Xiangyu Yue. Opengame: Open agentic coding for games, 2026. URL https://arxiv.org/abs/2604.18394.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *ICLR*, pp. 54107–54157, 2024.

Danny M. Kaufman, Timothy Edmunds, and Dinesh K. Pai. Fast frictional dynamics for rigid bodies. *ACM Trans. Graph.*, 24(3):946–956, July 2005. URL https://doi.org/10.1145/1073204.1073295.

Gergely Klár, Theodore Gast, Andre Pradhana, Chuyuan Fu, Craig Schroeder, Chenfanfu Jiang, and Joseph Teran. Drucker-prager elastoplasticity for sand animation. *ACM Trans. Graph.*, 35(4), July 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925906. URL https://doi.org/10.1145/2897824.2925906.

Jie Li, Gilles Daviet, Rahul Narain, Florence Bertails-Descoubes, Matthew Overby, George E. Brown, and Laurence Boissieux. An implicit frictional contact solver for adaptive cloth simulation. *ACM Trans. Graph.*, 37(4), July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201308. URL https://doi.org/10.1145/3197517.3201308.

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M Kaufman. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *TOG*, 2020.

Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. Codimensional incremental potential contact. *ACM Trans. Graph.*, 40(4), July 2021. doi: 10.1145/3450626.3459767. URL https://doi.org/10.1145/3450626.3459767.

Shanda Li, Tanya Marwah, Junhong Shen, Weiwei Sun, Andrej Risteski, Yiming Yang, and Ameet Talwalkar. Codepde: An inference framework for llm-driven pde solver generation. *TMLR*, 2026.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://www.science.org/doi/abs/10.1126/science.abq1158.

Chunjiang Liu, Xiaoyuan Wang, Haoyu Chen, Yizhou Zhao, Ming-Hsuan Yang, and László A Jeni. Simworlds: A multi-agent system for dynamic 3d scene creation, 2026. URL https://arxiv.org/abs/2607.01766.

Tongxu Luo, Rongsheng Wang, Jiaxi Bi, Chenming Xu, Zhengyang Tang, Jianlong Chen, Juhao Liang, Ke Ji, Shuqi Guo, Yuhao Du, Fan Bu, Wenyu Du, Xiaotong Zhang, Kyle Li, Shaobo Wang, Linfeng Zhang, Yuxuan Liu, Xin Lai, Chenxin Li, Yiduo Guo, Zhexin Zhang, Xinyuan Wang, Tianyi Bai, Ziniu Li, and Benyou Wang. Gamecraft-bench: Can agents build playable games end-to-end in a real game engine?, 2026. URL https://arxiv.org/abs/2606.17861.

Chunji Lv, Jiaxi Ye, Yuchen Jiang, Rexar Lin, and Changsheng Li. Physagent: Automating physics-based 4d synthesis via trajectory-grounded multi-agent feedback, 2026. URL https://arxiv.org/abs/2606.08688.

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. Discoverybench: Towards data-driven discovery with large language models, 2024. URL https://arxiv.org/abs/2407.01725.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. SWE-lancer: Can frontier LLMs earn $1 million from real-world freelance software engineering? In *ICML*, volume 267, pp. 44412–44450, 2025.

Moonshot AI. Kimi-K2.7-Code model. https://www.kimi.ai/, 2026. Model documentation and release page.

OpenAI. GPT-5.6 model. https://openai.com/, 2026. Model documentation and release page.

Qwen Team. Qwen3.8-Max model. https://qwen.ai/, 2026. Model documentation and release page.

Nithin Somasekharan, Ling Yue, Yadi Cao, Weichao Li, Patrick Emami, Pochinapeddi Sai Bhargav, Anurag Acharya, Xingyu Xie, and Shaowu Pan. CFDLLMBench: A benchmark suite for evaluating large language models in computational fluid dynamics. *JMLR*, 3(13):1–40, 2 2026. URL https://openreview.net/forum?id=kTcH1MnkjY.

Jos Stam. *Stable Fluids*. New York, NY, USA, 1 edition, 1999.

Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. A material point method for snow simulation. *TOG*, 32(4):1–10, 2013.

Xinjie Wang, Liu Liu, Yu Cao, Ruiqi Wu, Wenkang Qin, Dehui Wang, Wei Sui, and Zhizhong Su. Embodiedgen: Towards a generative 3d world engine for embodied intelligence, 2025. URL https://arxiv.org/abs/2506.10600.

Yanan Wang, Renxi Wang, Yongxin Wang, Xuezhi Liang, Fajri Koto, Timothy Baldwin, Xiaodan Liang, and Haonan Li. Simuscene: Training and benchmarking code generation to simulate physical scenarios, 2026. URL https://arxiv.org/abs/2602.10840.

Xiao Xia, Dan Zhang, Zibo Liao, Zhenyu Hou, Tianrui Sun, Jing Li, Ling Fu, and Yuxiao Dong. SceneGenAgent: Precise industrial scene generation with coding agent. In *ACL*, pp. 17847–17875, July 2025. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.873. URL https://aclanthology.org/2025.acl-long.873/.

Tianyidan Xie, Peiyu Wang, Yuyi Qian, Yuxuan Wang, Rui Ma, Ying Tai, Song Wu, Qian Wang, Lanjun Wang, and Zili Yi. Physcodebench: Benchmarking physics-aware symbolic simulation of 3d scenes via self-corrective multi-agent refinement, 2026. URL https://arxiv.org/abs/2604.23580v1.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *NeurIPS*, volume 37, pp. 50528–50652, 2024.

Hongxin Zhang, Chunru Lin, Junyan Li, Zhou Xian, Tsun-Hsuan Wang, and Chuang Gan. Gs-agent: Creating 4d physical worlds with generative simulation, 2026. URL https://arxiv.org/abs/2607.21522.

Zhipu AI. GLM-5.3 model. https://www.zhipuai.cn/, 2026. Model documentation and release page.

A BENCHMARK DETAILS

A.1 TASK CURATION

To curate phenomena that are challenging yet physically realistic, we survey computer graphics papers across seven domains and assemble a candidate pool. Then, we let human experts filter this pool to ensure task quality. A candidate is retained only if: (1) the phenomenon is visually identifiable; (2) a deterministic scene can be constructed; (3) the observable events can be stated explicitly; (4) the relevant physical quantities can be computed from raw solver outputs. To keep the benchmark diverse, the experts further merge candidates that share the same objects, geometry, initial state, and target event and differ only in parameter setting. For example, the two-meter-high, ten-body rigid stack settling under gravity appears in its source paper both with and without mass scaling; we keep it as a single task. The resulting benchmark consists of 168 tasks curated from 61 reference papers.

A.2 SCAFFOLD CONSTRUCTION

A code scaffold is the complete simulation setup for a task with the solver removed. It defines what the scene contains and how it is observed, but not how the physics is computed: the numerical method is left entirely to the solver. Every task therefore differs only in its solver, which is the variable we evaluate.

To ensure scaffold correctness, we construct each scene through an iterative human-AI refinement procedure. A scaffold agent first produces an initial draft from the task description according to a common template. Human experts then refine the draft over successive iterations, executing it with a test solver to expose and correct errors in the scene specification. The test solver serves a diagnostic purpose only: its trajectories are not treated as physical ground truth, and no solver under evaluation is compared against them. A scaffold is accepted once the physical and rendered geometry are consistent, the target event is visible, and the scene and render contracts pass validation. Each scaffold adopts the following structure:

Scaffold Template

```
ID_Name/
    config.yaml
    # Stores the task's public simulation configuration.
    scene.py
    # defines the initial state of the scene
    scene_geometry.yaml
    render_spec.yaml
    # object bindings in the scene.
    scaffold_manifest.json
    run.py
    render.py
    assets/
    solver/
```

A.3 SOLVER GENERATION

Each solver agent received the frozen scaffold and a task prompt, and worked in a private workspace. The prompt supplied four authoritative blocks: the task definition, the public configuration, the scene geometry with its StateFrame bindings, and a task-specific `Required raw trajectory outputs` block giving the path, shape, and unit of every required field. These requirements were derived from the physics-metric bindings and the platform's public trajectory-field vocabulary, so the agent exported raw physical state while the platform privately computed the physical plausibility scores. The agent could read the entire scaffold but modify only `solver/**`, with no network access, reference material, or files outside its workspace. Existing simulation engines were barred, so the physical model, discretization, and numerical scheme had to be derived from base scientific libraries. Specifically, the solver implemented three methods:

Solver.py

```python
class BenchmarkSolver:
    def initialize(self, scene, seed: int) -> None:
        raise NotImplementedError("Solver Agent must implement initialize()")

    def advance_to(self, time: float):
        raise NotImplementedError("Solver Agent must implement advance_to()")

    def diagnostics(self) -> dict[str, object]:
        return {}
```

The solver was required to perform physical time stepping and to respect the frozen configuration, geometry, collisions, and state bindings. Hard-coded animations, keyframes, precomputed trajectories, and scene-specific motion tables were prohibited. Left unconstrained, a solver generation agent tends to keep designing a full engine, or to iterate on parameters indefinitely, and may submit nothing at all. Our prompt therefore pushed it toward an early deliverable: save a complete physically-based baseline well before the allowance is spent, verify it with a compile check, and make at most two focused correction passes for concrete syntax, interface, required-field, or obvious implementation errors found by inspection. No token or tool-call limit was stated.

A.4 Task Outputs

After submission the solver source was frozen, and the platform executed it, recorded the trajectory, rendered the video, and evaluated the result with no further source changes.

trajectory.h5

```
/ metadata_json={"scene_id": "017_ten_body_stack", "seed": 17017}

/time  float64 (301,)
    [0.0, 0.02, 0.04, ...]

/frames/00000000/
    position  float64 (10, 3)
        [[ 0.015, 0.0, 0.150],
         [-0.012, 0.0, 0.355],
         [ 0.010, 0.0, 0.560], ...]
    velocity  float64 (10, 3)
        [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], ...]
    orientation  float64 (10, 4)
        [[0.0, 0.0, 0.0, 1.0], [0.0, 0.0, 0.0, 1.0], [0.0, 0.0, 0.0, 1.0], ...]
    angular_velocity  float64 (10, 3)
        [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], ...]
    mass  float64 (10,)
        [1.0, 1.0, 1.0, ...]
    body_id  int64 (10,)
        [1, 2, 3, ...]
    scalars/  attrs: contact_length_scale=0.2
    attributes/
    fields/
        contact_signed_distance  float64 (10,)
            [0.050, 0.005, 0.005, ...]
        external_force  float64 (10, 3)
            [[0.0, 0.0, -9.81], [0.0, 0.0, -9.81], [0.0, 0.0, -9.81], ...]
        interaction_force  float64 (10, 3)
            [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], ...]
```

```
/frames/00000001/  ...
/frames/00000300/  ...
```

A.5 Execution Environment

We separate the agent-generation environment from post-submission execution and evaluation. Agents run in disposable Debian Bookworm containers with only the assigned workspace mounted, holding NumPy, SciPy, Numba, h5py, PyYAML, and a bundled FFmpeg. The evaluation package is removed from these images, so the scoring metrics are hidden from the agents. Frozen submissions are then executed, rendered, and evaluated host-side by the platform. Rendering is performed by offscreen OpenGL rasterization through pyrender and PyOpenGL, trimesh for geometry processing, and ImageIO with FFmpeg for video encoding.

B Evaluation Details

B.1 Execution Checks

The execution check determines whether a submitted solver can run successfully and generate the required outputs. A case fails this check if it cannot export an evaluable trajectory and rendered video. Common failures include incomplete implementations, interface violations, runtime errors, numerical instability, and time exhaustion.

B.2 Visual Fidelity Evaluation

Our visual fidelity evaluation is divided into four parts: task-specific visual criteria construction, uniform frame sampling, VLM-based judging, and repeated score aggregation.

Part 1: Task-specific visual criteria. For every task, we construct four visual criteria from the target physical phenomenon and its required events. The criteria follow four shared dimensions: `event_sequence`, `interaction_response`, `signature_phenomenon`, and `temporal_coherence`. As an example, the Cube Drop Water task considers a rigid cube entering water, producing a splash and wake, and subsequently moving according to its buoyant response. Its task-specific visual criteria are shown below.

metric.yaml

```
visual:
  - id: visible_event_onset
    dimension: event_sequence
    criterion: The rigid cube visibly falls toward and enters
      the water surface.

  - id: visible_interaction_response
    dimension: interaction_response
    criterion: Water entry produces a visible splash and a
      depressed free surface around the cube.

  - id: visible_signature_outcome
    dimension: signature_phenomenon
    criterion: During subsequent motion, a wake forms and the
      cube continues to sink, rise, or float according to its
      buoyant response.

  - id: case_visual_coherence
    dimension: temporal_coherence
    criterion: The coupled motion remains continuous without
      cube-water interpenetration, surface sticking,
```

```
    teleportation, or a fluid response detached from the
    cube trajectory.
```

Part 2: Uniform frame sampling. We uniformly sample 20 frames over the full timeline of each rendered MP4. For a video containing N decodable frames, the sampled frame indices are

$$i_k = \text{round}\left(\frac{k(N-1)}{19}\right), \qquad k = 0, \ldots, 19.$$

This sampling includes the first and last frames. The sampled frames are kept in chronological order, resized so that each side is at most 768 pixels, and encoded as JPEG images. Each image is labeled with its temporal position, such as `(frame 1/20)`.

Part 3: VLM-based judging. We use `GPT-5.6-Sol` as the visual judge. The 20 ordered frames and the task-specific visual criteria are supplied in the same request. The judge assigns each criterion a score between 0 and 1 and provides a short evidence statement. The key prompt content is shown below.

VLM Judge Prompt

```
System:
You are a strict visual benchmark judge. Inspect the attached
frames in chronological reading order. Judge only visible
evidence. Return JSON only.

User:
The image blocks above are video frames in chronological order,
each labeled (frame i/N).

Evaluate every rubric item below.

For each item return criterion_id, score (number from 0 to 1,
where 1 is best and 0 is worst), and a concise evidence
statement. Penalize missing tank/subject visibility,
discontinuity, and absent required events.

Rubric:
[task-specific visual criteria]

Required JSON shape:
{"criteria": [
  {
    "criterion_id": "...",
    "score": 0.0,
    "evidence": "..."
  }
]}
```

Figure 8 shows an example of how VLM outputs score based on the evidence together. The GPT-5.6-Sol solver output shows a partially immersed cube with strong surface deformation but the splash is not clear enough.

Part 4: Repeated evaluation and score aggregation. We evaluate the same key frames five times using the same prompt. For each criterion, we average the five returned scores. The final visual score is the average of the four averaged criterion scores. The visual fidelity score for the Cube Drop Water task is shown below.

evaluation.json

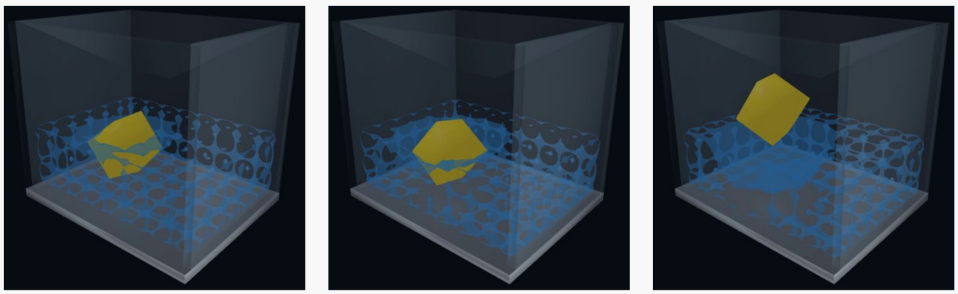
```
{
  "criteria": {
```

visible_interaction_response

score: 0.76

evidence:

Frames 3-6 show strong local surface deformation, with water rising around the impact area and a depressed region beneath/around the cube; however, there are no clearly separated splash droplets.



frame 3/20 frame 4/20 frame 6/20

Figure 8: An example of VLM output for the interaction-response criterion on case "Cube Drop Water".


```
    "visual": [
      {
        "criterion_id": "visible_event_onset",
        "score": 0.978,
        "details": {
          "attempt_count": 5,
          "frame_count": 20
        }
      },
      {
        "criterion_id": "visible_interaction_response",
        "score": 0.796,
        "details": {
          "attempt_count": 5,
          "frame_count": 20
        }
      },
      {
        "criterion_id": "visible_signature_outcome",
        "score": 0.846,
        "details": {
          "attempt_count": 5,
          "frame_count": 20
        }
      },
      {
        "criterion_id": "case_visual_coherence",
        "score": 0.772,
        "details": {
          "attempt_count": 5,
          "frame_count": 20
        }
      }
    ]
  },
  "visual_score": 0.848
```

```
}
```

Example interpretation. The cube entry is clearly visible and therefore receives the highest criterion score. The splash, subsequent buoyant motion, and overall temporal coherence are also captured, although the cube–water interface is less clear in some frames. The criterion-level results expose visual weaknesses that would be hidden by reporting only the final score.

B.3 Physical Plausibility Evaluation

The physical plausibility evaluation is divided into three parts: task-specific criterion binding, deterministic calculator execution, and result inspection.

Part 1: Task-specific physical criteria and calculator bindings. Physical criteria are selected according to the domain and target phenomenon of each task. Each criterion contains a `dimension` that groups related physical properties, while the `calculator` specifies the numerical check to execute. As an example, we use the `GPT-5.6-Sol` result for the Billiard Impulse Chain task. The task evaluates nonpenetrating contact, linear momentum balance, center-of-mass consistency, and internal interaction-force balance. We fully expand the first binding below and abbreviate the remaining entries.

metric.yaml

```yaml
physics:
  - id: nonpenetrating_contact
    dimension: contact_and_interface_conditions
    numeric_binding:
      calculator: contact.penetration
      inputs:
        signed_distance: fields.contact_signed_distance
        length_scale: scalars.contact_length_scale
      parameters: {}
      reduction:
        type: maximum

  # Remaining entries follow the same structure and are abbreviated.
  - id: linear_momentum_balance
  - id: center_of_mass_consistency
  - id: internal_interaction_force_balance
```

The names on the left side of the `inputs` block are calculator arguments, and the values on the right are paths to the corresponding trajectory fields. The `parameters` block provides additional fixed settings, while `reduction` specifies how the per-frame errors are combined.

Part 2: Deterministic calculator execution. The evaluator resolves the input paths, executes the selected calculator over the trajectory, and reduces its per-frame errors to one normalized error. The central execution logic is:

evaluator.py

```python
binding = criterion["numeric_binding"]
calculator = CALCULATORS[binding["calculator"]]

result = calculator.compute(
    frames,
    binding["inputs"],
    binding.get("parameters", {}),
)

normalized_error = reduce_values(
```

```
    result.normalized_errors,
    binding["reduction"],
)
```

The contact calculator measures the deepest ball–ball or ball–support penetration relative to the contact length scale. The momentum calculator compares the change in total system momentum with the exported external forces. The center-of-mass calculator compares each ball's observed displacement with the displacement obtained by integrating its velocity. The interaction-force calculator checks whether the exported internal contact forces cancel in aggregate.

Part 3: Example results. For the Billiard Impulse Chain example, we fully expand the first result. The remaining entries follow the same structure and retain only the fields needed for identification and comparison.

evaluation.json


```
{
  "task_id": "012_billiard_impulse_chain",
  "solver_model": "gpt-5.6-sol",
  "physics": [
    {
      "criterion_id": "nonpenetrating_contact",
      "calculator": "contact.penetration",
      "reduction": {
        "type": "maximum"
      },
      "normalized_error": 0.0,
      "details": {
        "sample_count": 181,
        "worst_frame_index": 0
      }
    },

    // Remaining entries follow the same structure and are abbreviated
    .
    {
      "criterion_id": "linear_momentum_balance",
      "normalized_error": 1.0
    },
    {
      "criterion_id": "center_of_mass_consistency",
      "normalized_error": 0.101274
    },
    {
      "criterion_id": "internal_interaction_force_balance",
      "normalized_error": 0.0
    }
  ],
  "physical_score": 73.734
}
```


The result contains no detected penetration, and the exported internal contact forces cancel correctly. The center-of-mass error is 0.101, with a maximum position residual of approximately 0.0149 meters. However, the total momentum change is inconsistent with the exported external-force record, producing a momentum-balance error of 1.0. The simulation therefore receives a physical score of 73.734, compared with a visual score of 99.5. It reproduces the expected motion convincingly in the rendered video, while the trajectory reveals a momentum inconsistency that is difficult to detect visually.

C CASE STUDIES

To examine the complementarity of Visual Fidelity and Physical Plausibility, we present a fine-grained case study that illustrates the behavior of our metric. Specifically, we group the results into three categories: Visually Correct but Physically Incorrect, Physically Correct but Visually Inaccurate, and Metrics in Agreement. Each category is illustrated with four pairs of cases, accompanied by a clarification that traces the judgment back to its root cause. Together, these examples show that neither score is sufficient in isolation: visual fidelity evaluates whether the expected phenomenon is visibly reproduced, whereas physical plausibility tests whether the generated dynamics satisfy the selected quantitative constraints.

C.1 RIGHT APPEARANCE, WRONG PHYSICS

Figure 9 simulates a square cloth sheet falling onto a sharp edge and an isolated point. Claude-Opus-5 and Qwen-3.8-Max obtain relatively similar visual scores, because both videos show the expected falling cloth, obstacle interaction, and folded resting configuration. However, the physical evaluation separates the two solutions much more clearly. The main differences arise from the `orientation_preserving_deformation` and `nonpenetrating_contact` criteria. Qwen-3.8-Max's simulation remains visually recognizable as draping cloth, but its physical trajectory contains extensive element inversion and obstacle interpenetration that are difficult to diagnose from rendered frames alone.

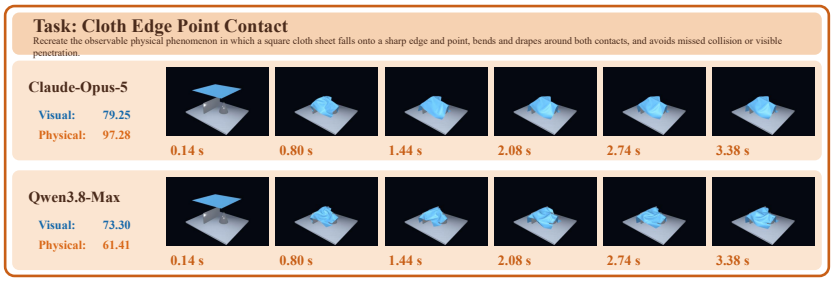


Figure 9: Case study results for the cloth edge–point contact task.

As shown in Figure 10, Gemini-3.7-Flash and Claude-Opus-5 both receive relatively high visual scores because both videos show the tube expanding outward while retaining its constrained ends. However, the physical evaluation clearly favors Gemini-3.7-Flash. The main difference arises from the `quasistatic_nodal_force_balance` criteria. Claude-Opus-5 produces a plausible inflated silhouette, but its unconstrained material nodes have large residual forces, indicating that the displayed configuration is not close to mechanical equilibrium.

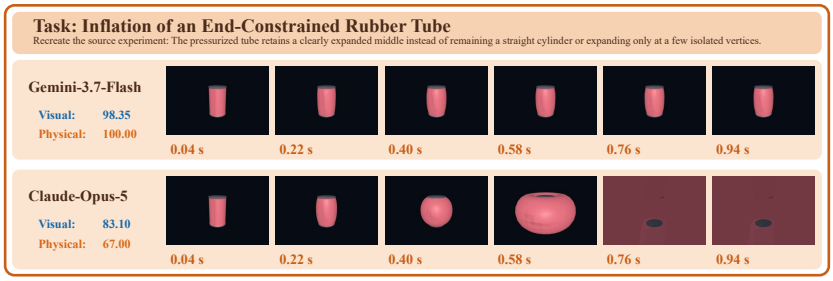


Figure 10: Case study results for the pressurized rubber tube task.

The task shown in Figure 11 simulates a rubber cube compressed vertically between a support and a loaded top face. The distinction of their physical plausibility score is primarily produced by the `quasistatic_nodal_force_balance`. Gemini-3.7-Flash reproduces the expected macro-

scopic shape, but its trajectory contains a much larger node-wise force residual, which cannot be determined reliably from the rendered surface alone.

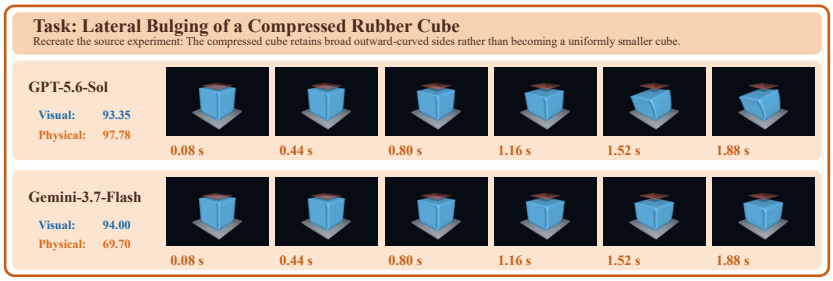


Figure 11: Case study results for the rubber cube compression task.

In figure 12, both agents simulate the characteristic axial banding pattern. While GPT-5.6-Sol produces convincing visible bands, it fails to respect the `particle_fluid_force_balance` criterion. Specifically, its exported particle-to-fluid and fluid-to-particle forces fail to cancel, revealing an inconsistent momentum exchange that is not observable in the rendered video.

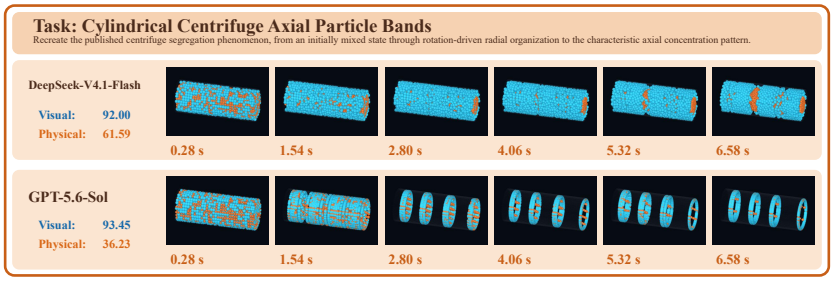


Figure 12: Case study results for the cylindrical centrifuge axial particle bands task.

C.2 Right Physics, Wrong Appearance

This task simulates a parabolic arch assembled from rigid blocks that should remain standing through frictional contact. As shown in Figure 13, Kimi-K2.7-Code preserves the arch, whereas GLM-5.3 rapidly loses the intended structure. However, their physical scores are nearly identical because none of the evaluated criteria directly verifies if the complete arch remains standing. The visual fidelity score complements this gap by directly recognizing the stable arch as the signature outcome and strongly penalizing the collapsed scene.

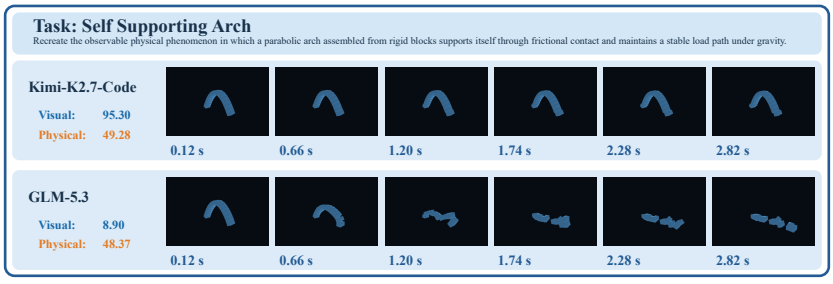


Figure 13: Case study results for the self-supporting arch task.

In Figure 14, we provide a simulation where a tendril-covered elastic ball impacting a wall, flattening, and quickly recovering its rounded shape. Claude-Opus-5 reproduces the impact and recovery,

whereas GPT-5.6-Sol remains almost static. Nevertheless, GPT-5.6-Sol receives a relatively high physical score because physical criteria such as `linear_momentum_impulse_balance` do not directly require an impact to occur. A nearly static trajectory can conserve mass, avoid element inversion, and maintain a small budget residual while omitting the target phenomenon entirely.

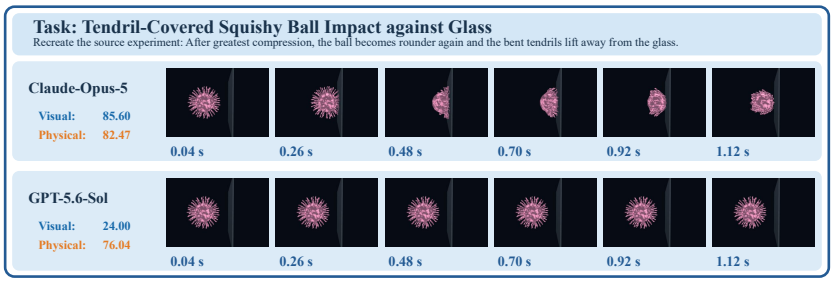


Figure 14: Case study results for the squishy ball glass impact task.

As shown in Figure 15, we provide a task in which incoming airflow should accelerate a dynamic rotor and produce a visible helical smoke wake. GPT-5.6-Sol barely responds, yet it still receives a perfect physical score: its nearly static state drives both sides of the `angular_momentum_torque_budget`, `solid_fluid_exchange_force_balance`, and `impermeable_interface_normal_velocity` balances to approximately zero. However, visual fidelity closes this loophole, checking rotor acceleration, smoke transport, and the helical wake directly against the scene.

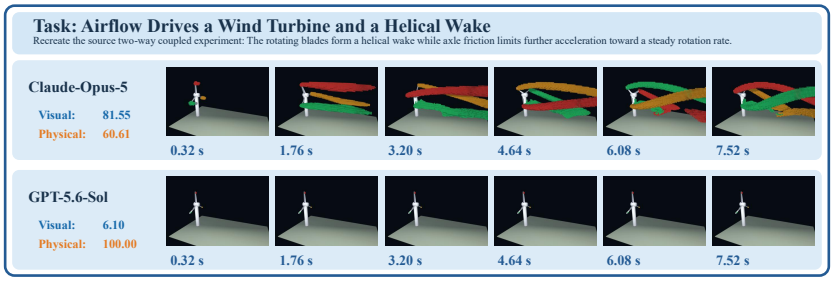


Figure 15: Case study results for the air-driven wind turbine task.

The task in Figure 16 simulates a knitted scarf sliding down an incline and accumulating into thick bunched folds. Their physical scores are almost identical because both conserve mass, remain within the allowed surface-strain range, and avoid measured contact penetration, while both have similarly poor momentum-balance residuals. However, only GLM-5.3 reproduces the expected cloth pile, while in Claude-Opus-5 the scarf flies off the edge.

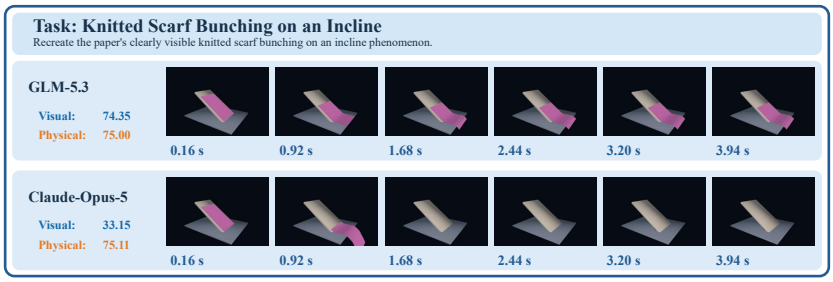


Figure 16: Case study results for the knit scarf incline bunching task.

C.3 Both Metrics Agree

As a complementary set of examples, these cases show agreement between the two metrics. Successful solvers reproduce the target phenomenon while satisfying the physical criteria, whereas unsuccessful solvers fail both visually and physically. Together, they illustrate that the two evaluations are consistent when solver quality isn't ambiguous.

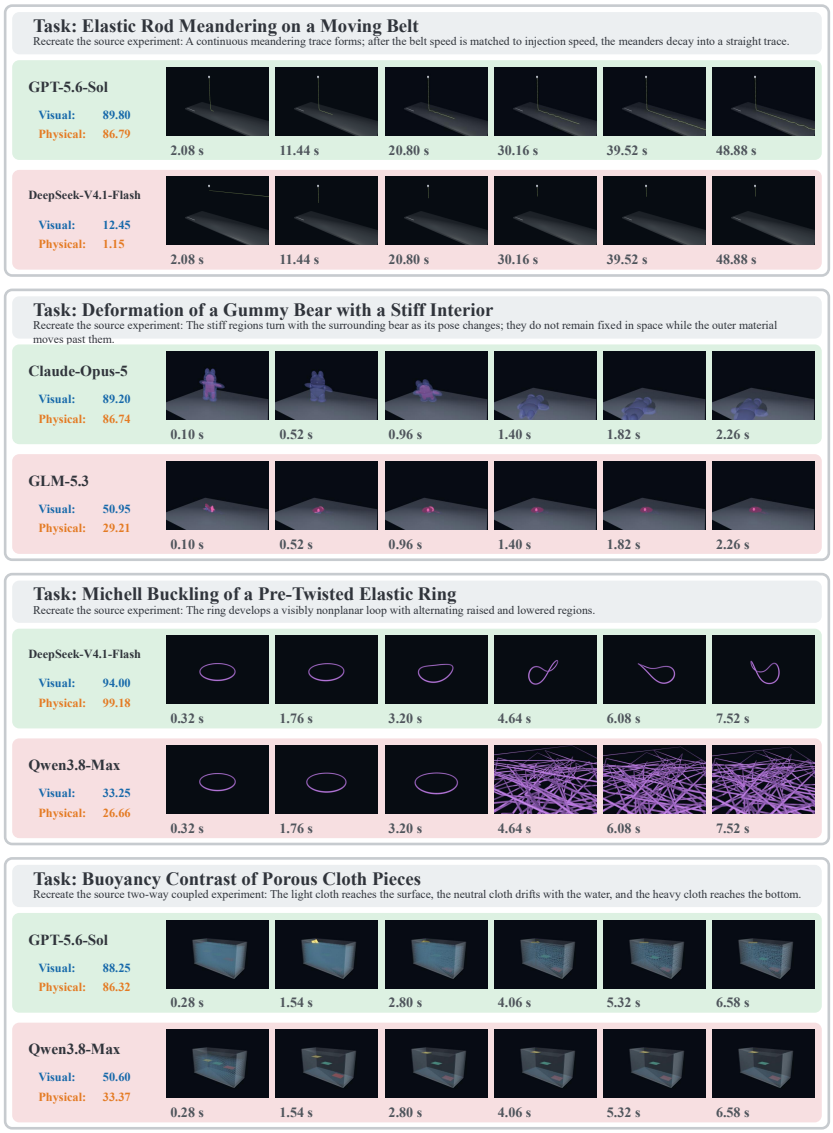


Figure 17: Case study results for unambiguous solvers.

D More Results

D.1 Complete Main Results

Figure 18 shows broadly consistent performance trends between visual and physics scores, together with systematic differences across domains. In rigid-body dynamics, every model scores higher visually than physically. For example, GPT-5.6-Sol obtains 65.5% versus 44.2%. This gap suggests that visually plausible motion can coexist with violations detected by the physical checks, demonstrating the necessity of physical evaluation. The pattern reverses for plastic and complex materials,

where physics scores consistently exceed visual scores, suggesting that, in certain domains, even minor violations of physical constraints by a solver can lead to poor visual results. The two criteria therefore provide complementary evidence: visual evaluation assesses observable fidelity, while physics evaluation checks constraints that plausible-looking outputs may nevertheless violate.

D.2 Visual-Judge Stability

To verify the stability of the VLM judge, we repeat the visual-fidelity evaluation of each task five times. Figure 19 reports the mean scores per domain, with error bars indicating the mean standard deviation within the task in the five evaluations. The error bars are generally small across agents and physical domains, showing that repeated evaluations produce consistent scores. Slightly larger variations appear in some cases, but the overall performance patterns remain stable. These results suggest that our VLM judge provides consistent visual-fidelity evaluations reliably.

D.3 Human Annotations

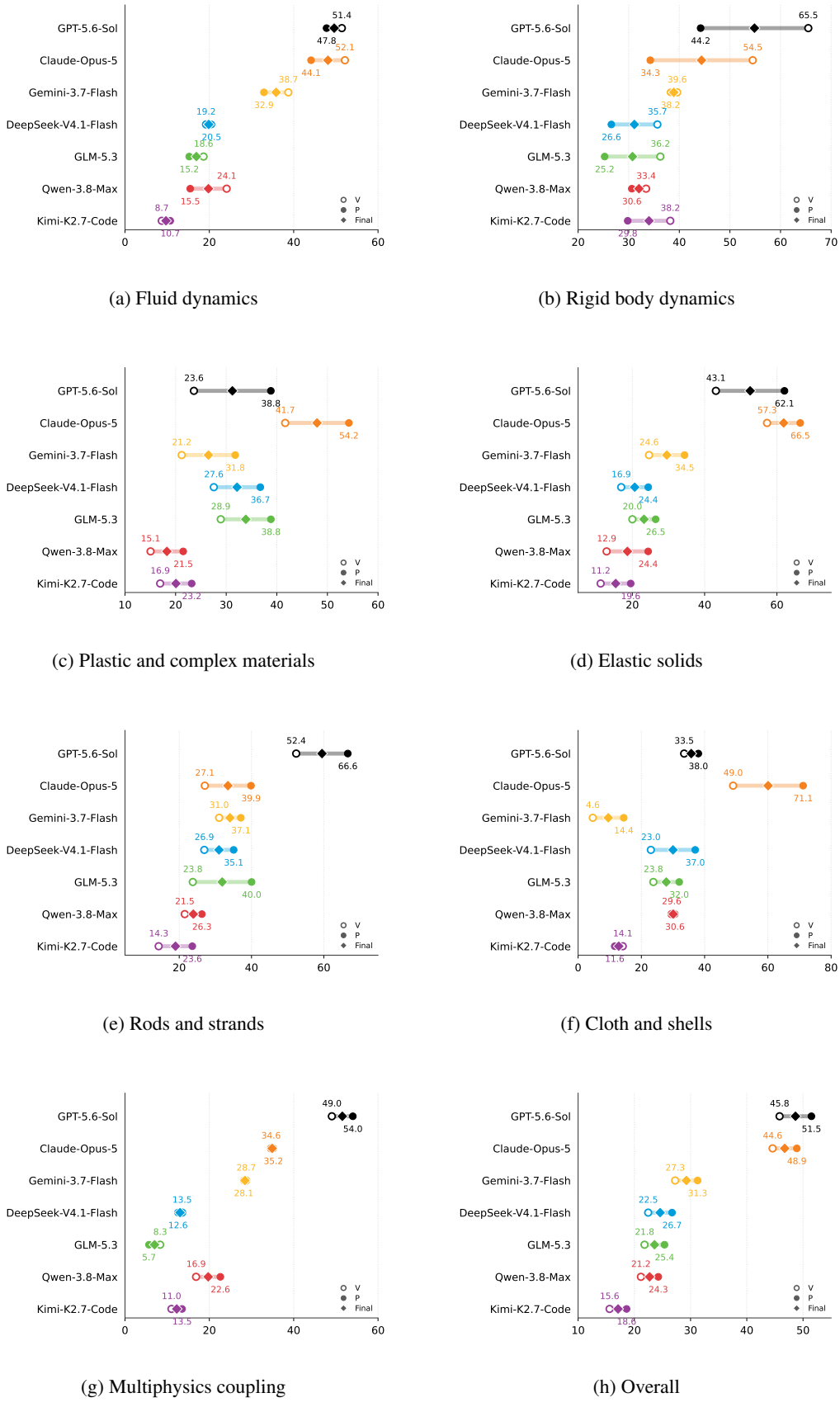
Our Visual Fidelity evaluation consists of four task-specific criteria for each case, which are assessed by a VLM using a shared set of key frames. To validate this evaluation protocol against human judgment, we asked expert annotators to assess solver quality using the same key frames and evaluation criteria. Among the 661 cases with successfully rendered videos, we sampled 231 cases for human evaluation, including 40 overlapping cases rated by multiple annotators to measure Human-Human consistency. As shown in Table 4, the VLM scores show strong agreement with human judgments in both absolute values (Pearson $r = 0.730$) and rank ordering (Spearman $\rho = 0.713$). For comparison, Human-Human correlations on the shared cases are lower, with Pearson $r = 0.624$ and Spearman $\rho = 0.607$. These results indicate that the VLM-based evaluation aligns well with expert judgments and achieves correlations, in this sample higher, than those observed between individual human annotators.

Table 4: Correlation of VLM and human judgments.

Comparison	Pearson r	Spearman ρ
Human–VLM	0.730	0.713
Human–Human	0.624	0.607

D.4 Solver Style

To grasp a view of each agent's output solver style, we further provide the size and runtime efficiency of the generated solvers in Table 5. Being the best performance model, GPT-5.6-Sol also produces the most compact implementations, averaging 234 lines and 1,028 words, while achieving the shortest execution time of 22.78 seconds. Though showing comparable performance, Claude-Opus-5 attains a comparable benchmark score, but its solvers are the longest on average and have the highest mean execution time (100.79 seconds). GLM-5.3 has the lowest mean memory consumption (235.81 MiB), but its solvers are longer than those of GPT-5.6-Sol in context. Overall, solver size and resource usage vary substantially across agents, even when their performance scores are similar.



(a) Fluid dynamics

(b) Rigid body dynamics

(c) Plastic and complex materials

(d) Elastic solids

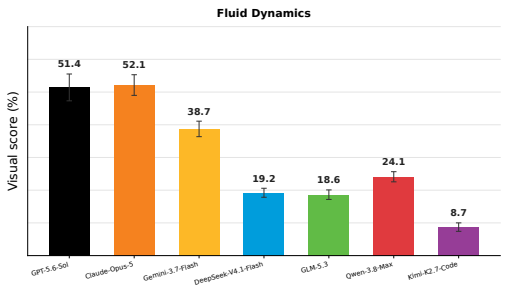
(e) Rods and strands

(f) Cloth and shells

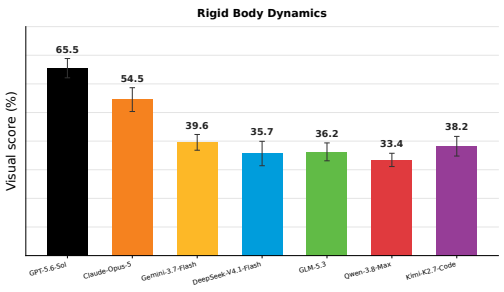
(g) Multiphysics coupling

(h) Overall

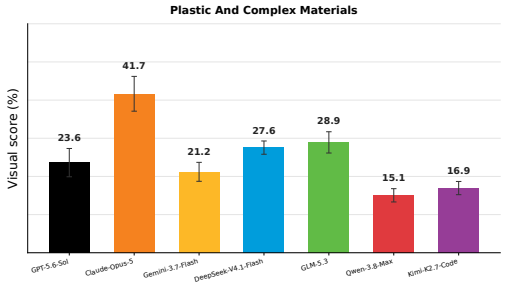
Figure 18: Visual Fidelity and Physical Plausibility scores across different physical domains.



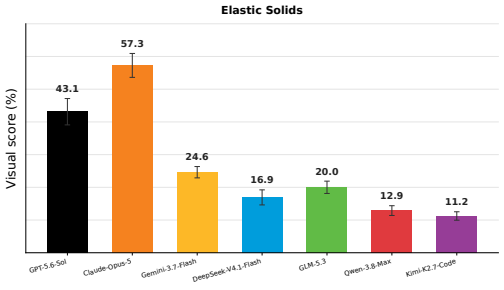
Fluid



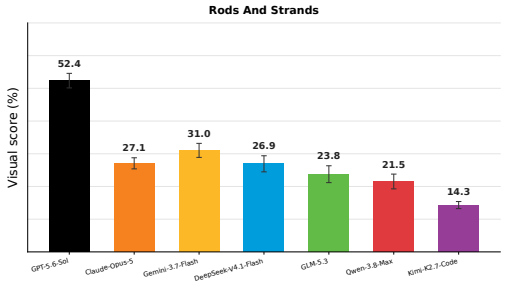
Rigid Body



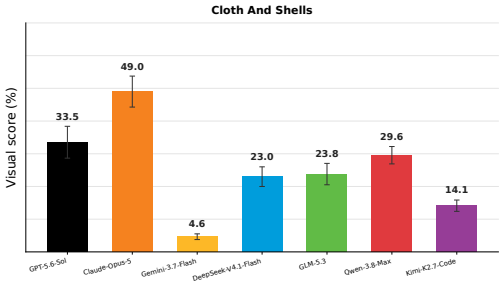
Plastic & Complex Materials



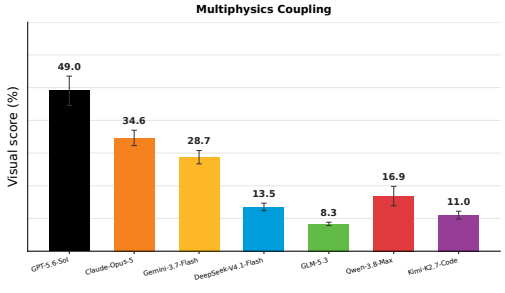
Elastic Solids



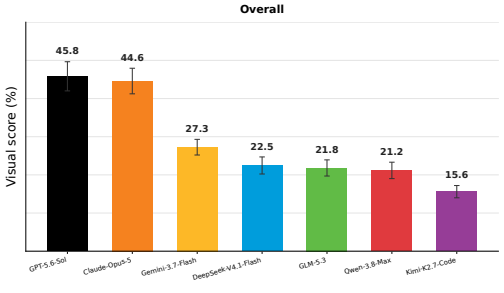
Rods & Strands



Cloth & Shells



Multiphysics Coupling



Overall

Figure 19: Performance across different physics domains.

Table 5: Statistics of generated solvers and their runtime resource usage. We report the mean and maximum solver size in lines and words, together with execution time and memory consumption. The minimum value in each column is highlighted yellow.

Model	Lines		Words		Runtime	
	Mean	Max	Mean	Max	Time (s)	Memory (MiB)
GPT-5.6 Sol (OpenAI, 2026)	234	606	1028	2765	22.78	388.21
Claude Opus 5 (Anthropic, 2026)	523	878	2382	4183	100.79	347.28
Gemini 3.7 Flash (Google DeepMind, 2026)	345	639	1495	3130	61.29	414.26
DeepSeek-V4.1-Flash (DeepSeek-AI, 2026)	512	1060	2293	4951	77.04	355.77
GLM-5.3 (Zhipu AI, 2026)	440	1014	2021	5148	59.42	235.81
Qwen-3.8-Max (Qwen Team, 2026)	371	806	1705	3522	68.09	258.59
Kimi-K2.7-Code (Moonshot AI, 2026)	372	707	1496	2868	60.94	295.42

D.5 Task-Level Results

Figures 20 and 21 report per-task agent wall time and token usage, respectively. The heatmaps reveal substantial variation across both agents and tasks: some agents consistently consume more time or tokens, while certain tasks require considerably greater resources across multiple agents. These task-level results suggest that the computational difficulty of solver generation is highly uneven and depends on both the physical phenomena and the agent's problem-solving ability.

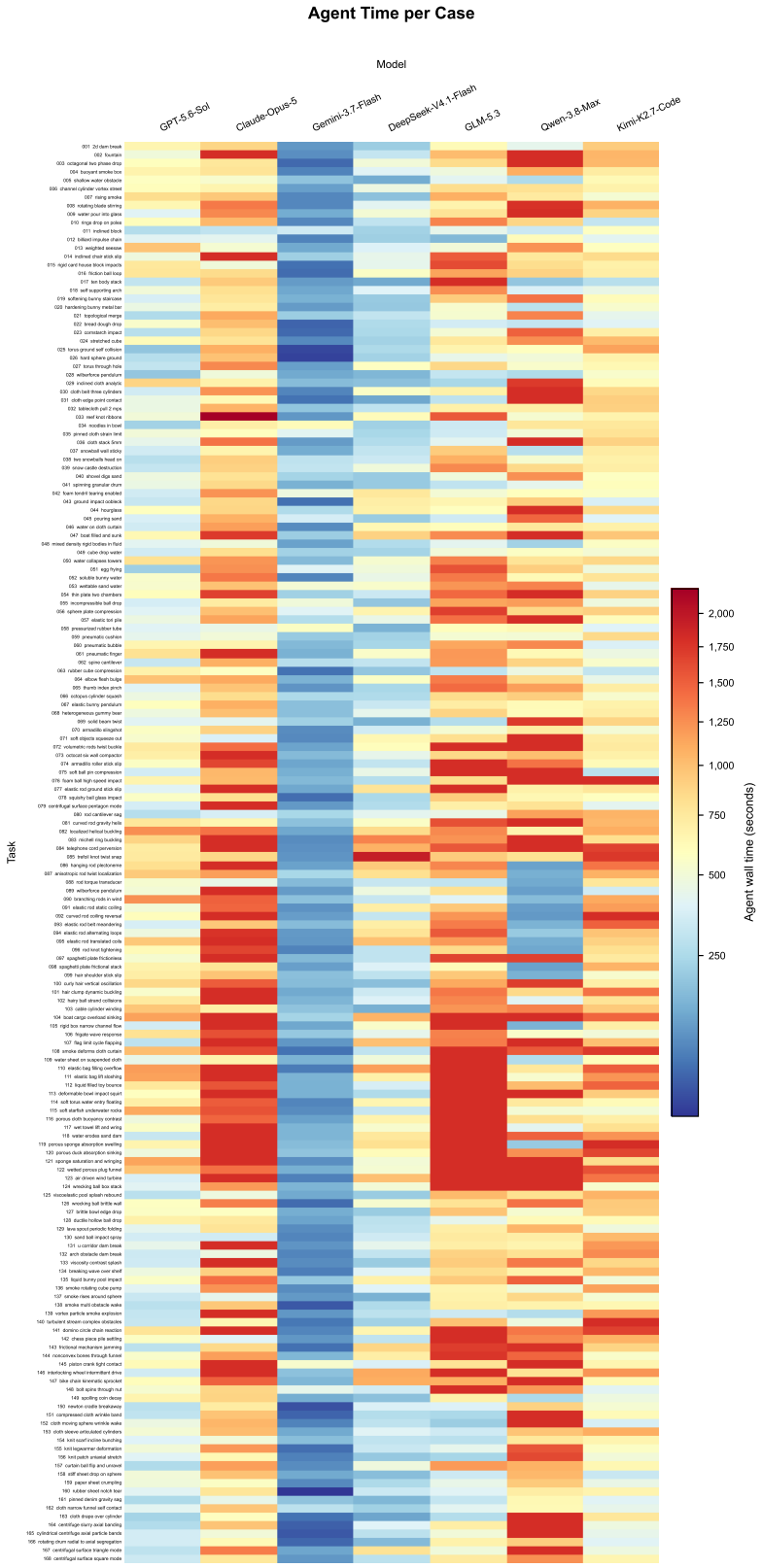


Figure 20: Per-case agent wall time across 168 benchmark tasks and seven models. Each cell represents the generation time consumed by the agent in a single run. Blue indicates shorter durations, while red indicates longer durations.

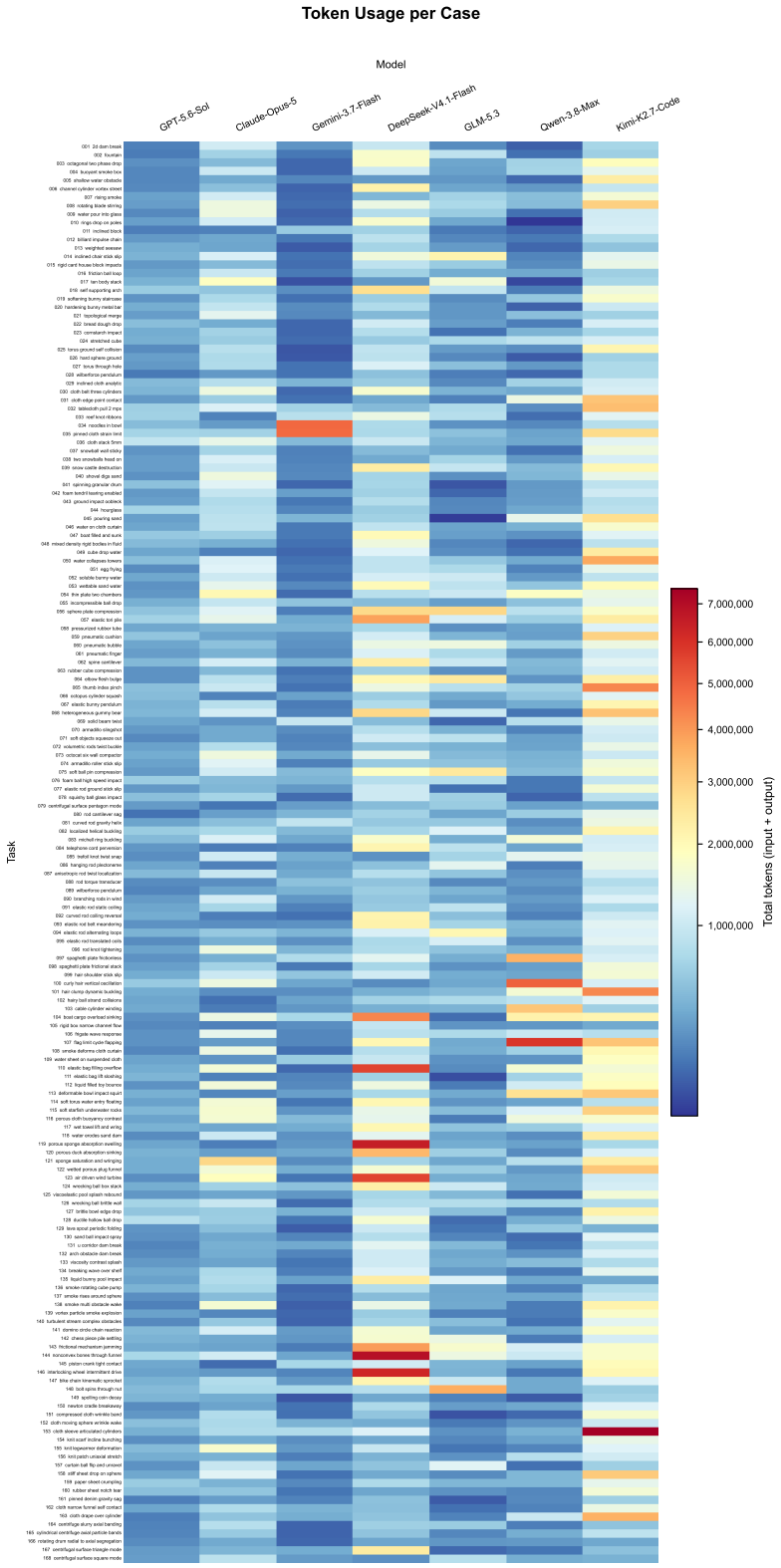


Figure 21: Per-case token usage across 168 benchmark tasks and seven models. Each cell represents the total input and output tokens consumed in a single run. Blue indicates lower usage, while red indicates higher usage.