CARPHY: A CARRIER PHYSICS AWARE POSE CORRECTION AGENT FOR INTERACTIVE WORLD MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Interactive world models (IWMs) generate virtual environments that evolve in response to user actions, enabling users to explore and interact with generated worlds. However, they can overlook physical interactions between the camera carrier and the surrounding environment, producing implausible motion such as passing through solid walls. To address this issue, we propose **CarPhy**, a pose correction agent that explicitly models **Car**rier **Phy**sics to calibrate IWMs to generate plausible videos. Since different scenes require different physical interpretations and modeling steps, CarPhy equips the agent with multimodal scene and carrier context, along with tools for semantic analysis, world editing, and motion rollout. Guided by this context, the agent adaptively selects and combines tools to determine how the environment affects the carrier and simulate the resulting motion. We further construct a test set spanning diverse terrain and obstacle configurations to evaluate physical event recognition and pitch-response timing. Experiments demonstrate improved physical event recognition and more accurate pitch-response timing across multiple VLM and LLM backbones.

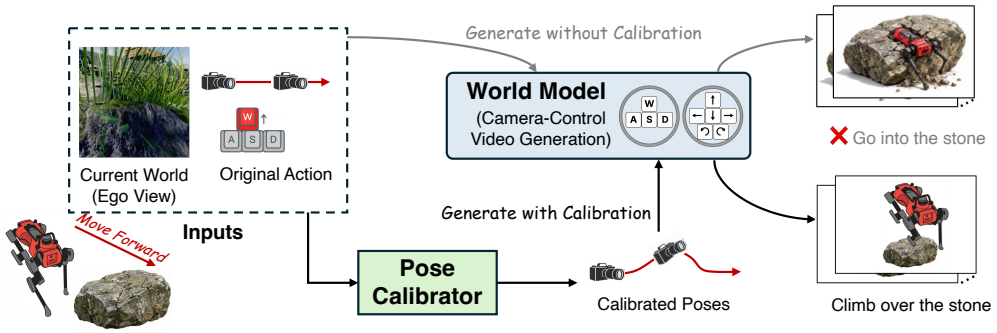


Figure 1: **Carrier Physics Aware Camera Pose Correction.** Given the current ego view and an original camera action (left), generation without calibration ignores carrier physics and drives the camera into the stone (top). Our method predicts calibrated camera poses that enable physically grounded camera control for the world model. This physically grounded correction allows the generated motion to follow the carrier response and climb over the stone (bottom).

1 INTRODUCTION

Interactive world models (IWMs) (Bruce et al., 2024; Shen et al., 2026; **?**) generate videos in response to users' control instructions, such as 6DoF camera poses (Sun et al., 2026), enabling video generation with controllable viewpoints. This ability to control viewpoints turns video generation into an interactive experience, supporting applications ranging from virtual exploration and entertainment (Valevski et al., 2025) to interactive simulation and embodied AI (Li et al., 2026b; Jiang et al., 2026).

However, current IWMs often overlook carrier physics, *i.e.*, how interactions with the IWMs affect the position and physical state of invisible or partially visible carriers. As illustrated in Figure 2,

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.

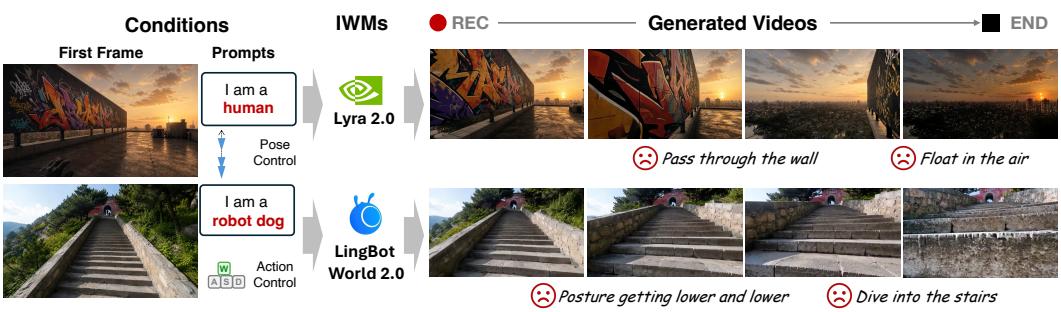


Figure 2: **Failure cases: IWMs ignore the physical plausibility of the carrier.** Even with explicit carrier prompts, current IWMs can generate camera motion that violates the physical response of the carrier. Lyra 2.0 passes through the wall and floats in the air, and LingBot World 2.0 lowers the viewpoint and dives into the stairs.

the viewpoint passes through a solid wall like a floating ghost instead of stopping, even when the prompt explicitly specifies a robot dog as the carrier. Such failures limit the use of generated videos in downstream applications such as embodied AI, where the visual consequences of actions must be physically plausible.

A key reason is that current IWMs lean more on visual rather than carrier dynamics (Hu et al., 2023; **?**; Wayve, 2025). As discussed in CWM (Chen et al., 2026), videos capture the observable outcomes of interactions, but do not directly specify the mechanisms that produce them (Alonso et al., 2024; Valevski et al., 2025). This distinction is particularly important for invisible or partially visible carriers, whose geometry and contact with the environment are not fully captured in the visual observations. Therefore, we aim to explicitly model the carrier's physical response and feed the corrected pose into the IWM.

Explicit modeling is challenging because scenes contain varied terrain, obstacles, and partially observed geometry (Miki et al., 2022). A fixed workflow may struggle when different scenes require different checks and modeling steps. For example, a low wall and a patch of grass may produce similar coarse depth profiles, yet require different semantic checks and collision models. An agent combines an LLM or VLM with a harness that provides context and tools (Yao et al., 2023), allowing it to plan and execute these steps according to the scene. This motivates us to achieve flexible explicit modeling of carrier physics through an agentic approach.

We therefore propose **CarPhy**, a pose correction agent that explicitly models **Car**rier **Phy**sics. Since pose correction requires both environment modeling and motion simulation, at the context level, CarPhy exposes the 2D and 3D data needed for environment modeling as observations to the agent. These observations include RGB images, depth maps, and reconstructed point clouds, together with carrier specifications and motion history. At the tool level, CarPhy provides environment modeling and motion simulation tools as the agent's action space. These tools support multiple tools, such as semantic segmentation, 3D scene construction, terrain completion and physics simulation, to estimate the carrier's pose response.

Moreover, we construct an evaluation set and corresponding metrics to assess physical event recognition and carrier pose accuracy. Experiments across multiple VLMs and LLMs demonstrate the broad applicability of our CarPhy. Moreover, the results also show that our design improves pose regression accuracy across diverse physical interactions.

Our main contributions are:

- We formulate carrier-aware pose correction for IWMs to account for the effects of carrier–environment interactions on camera motion.
- We propose **CarPhy**, which uses multimodal context to adaptively select and combine tools for explicit carrier-physics modeling.
- Experiments on a curated evaluation set show improved physical event recognition and pitch-response timing across multiple model backbones.

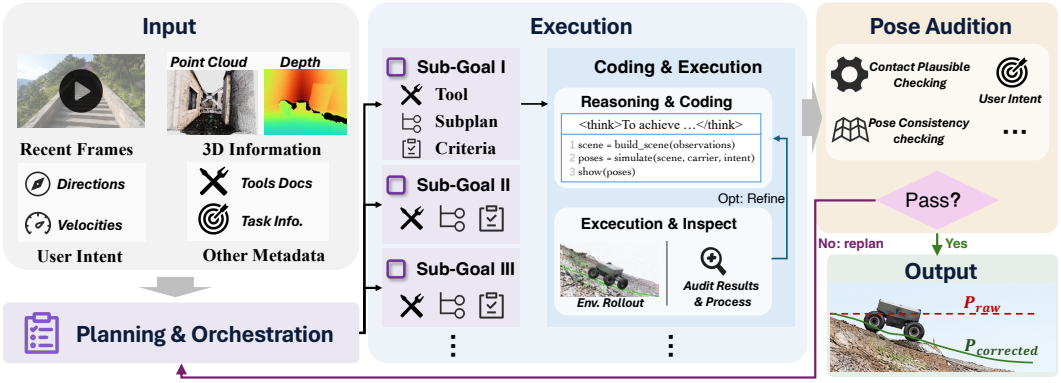


Figure 3: **Overview of the CarPhy agent harness.** Given recent visual observations, 3D information, user intent, and task metadata, the agent divides the task into smaller goals. It then iteratively reasons, executes code, and invokes tools to refine its world hypothesis. The resulting trajectory is audited for contact plausibility, pose consistency, and alignment with user intent. If the trajectory fails the audit, the agent replans and continues the execution loop. Otherwise, it returns the validated trajectory as the final output.

2 Related Work

Physically Grounded Generative World Models. Although video generative models achieve strong visual realism, they often fail to preserve physical consistency and plausible object dynamics (Yuan et al., 2024; Zhang et al., 2025; Wang et al., 2025). Recent work addresses this limitation through explicit simulation, learned physical dynamics, retrieved physical priors, and physics-aware post-training (Liu et al., 2024a; Montanaro et al., 2024; Yuan et al., 2026; Cheng et al., 2026a; Zhang et al., 2026; Cai et al., 2026). Simulator-in-the-loop methods further use reconstructed physical scenes and trajectories to guide video synthesis (Foo et al., 2026). These approaches primarily model scene and object evolution. Camera control remains largely decoupled from the evolving physical state. (Liu et al., 2024a; Foo et al., 2026; Zhang et al., 2025). Our method uses simulation to evolve the world and provide physical feedback for subsequent camera control.

Camera-Controllable Generative World Models. Recent methods enable explicit camera control through camera trajectories and motion conditions, improving trajectory fidelity, viewpoint consistency, and disentanglement between camera and scene motion (Cheng et al., 2026b; Lee et al., 2026; Zhao et al., 2026; Shin et al., 2026; Wang et al., 2026c; Fortier-Chouinard et al., 2026). Other work reasons over 3D geometry to plan feasible and collision-aware camera trajectories (Li et al., 2026c; Bian et al., 2026). Existing approaches typically condition generation on predefined camera motion or plan trajectories from a fixed scene representation. They do not link camera decisions to the evolving physical state. Our method evaluates camera transitions through simulator rollouts and adapts control based on the resulting physical states.

Agentic Planning and Control Grounded in Simulation. Recent work uses agents to translate high-level instructions into executable representations for controllable world generation and simulation (Wang et al., 2026b; Li et al., 2026a; Wang et al., 2026a; Li et al., 2026c). These methods use LLMs or coding agents to generate camera plans, coordinated motion controls, executable scene programs, or drafts produced through simulation that can be iteratively refined. These methods use executable simulation for grounded reasoning. Simulator feedback mainly supports scene construction or planning before execution. Its use for continuous camera adaptation remains limited. Our method places a coding agent in the loop between camera control and simulation. The agent uses visual evidence and simulator rollouts to revise scene assets and rerun the simulation. The process ends when the trajectory passes physical verification or the system abstains.

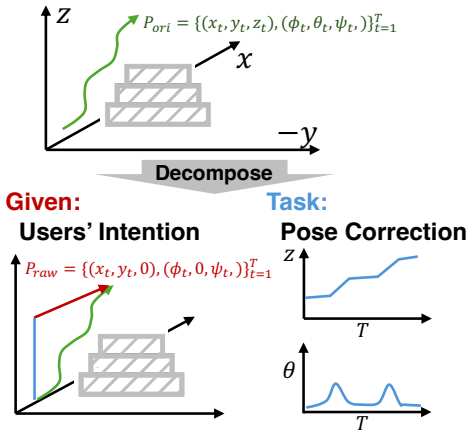
3 Method

CarPhy is an agentic pose correction framework for action-conditioned world models. As illustrated in Figure 3, it preserves the user-specified horizontal motion intent while predicting the physical response of the carrier through a task-specific coding-agent harness (Zechner, 2025). We first formulate the pose correction task and then describe the detailed agent loop.

3.1 PRELIMINARIES

Problem Formulation. As illustrated in Figure 4, the horizontal trajectory specified by P_{raw} represents the motion intent of the user. CarPhy preserves this intent while correcting the physical response of the carrier induced by interactions with the environment. In this work, the correction primarily addresses terrain-induced pitch changes and motion blocked by physical constraints. The complete pose definition and formal task formulation are provided in Appendix A.

Agent Harness. Recent work (Yao et al., 2023) characterizes an agent as the composition of a foundation model and an external *harness*, which provides the runtime that connects model reasoning to execution in the external environment. In CarPhy, the harness presents the current observations and scene state to the agent. It controls access to the available tools. After each rollout, feedback from the physical verifier is returned to guide the next refinement step.



(x, y, z): *FLU (forward, left, up).*
(ϕ, θ, ψ): *Euler angles (roll, pitch, yaw).*

Figure 4: **Task formulation.** CarPhy predicts the physical response by correcting pose changes caused by terrain while preserving the intended motion.

3.2 CARPHY AGENT

CarPhy adopts a coding agent harness (Zechner, 2025) for camera control under physical constraints, as illustrated in Figure 3. The workflow begins with 3D geometry reconstruction from visual observations and the construction of physical scene assets. The user's motion intent is projected onto the horizontal plane to form the initial trajectory P_{raw}, from which control inputs are derived for carrier simulation in PyBullet. A physical verifier checks the resulting trajectory and returns its findings to the agent. Based on the verification feedback, the agent inspects the scene assets and repairs them when necessary. After each repair, simulation and verification are repeated using the revised assets. The execution context stores intermediate results for subsequent decisions. The loop ends when the agent submits a corrected trajectory that passes verification or abstains because the evidence is insufficient.

This design assigns precise numerical operations to executable geometry and physics tools, allowing the VLM to focus on interpreting the scene and deciding which computations are needed. Tool results inform subsequent reasoning, so the agent can select further checks according to the remaining uncertainty. As new evidence becomes available, the agent can revise its choice of tools and the order of execution to address the needs of the current scene.

3.3 CONTEXT MANAGEMENT AND TOOL DESIGN

Figure 5 illustrates how CarPhy organizes agent interaction around a physical world hypothesis. The agent system is governed by two design questions: *what information should be exposed to the agent*, and *what capabilities should be externalized as tools*.

Context Management. CarPhy exposes complementary visual and geometric evidence, together with information about embodiment and motion intent. A key distinction is between semantic and metric reasoning: 3D geometry locates physical structures and provides metric quantities such as support height, slope, clearance, and spatial layout, whereas 2D visual observations help determine what these structures represent physically. Both are necessary. Geometric evidence alone may not distinguish traversable vegetation from rigid obstacles or non-supporting surfaces from valid terrain. Visual semantics alone cannot provide sufficiently precise metric information for pose correction.

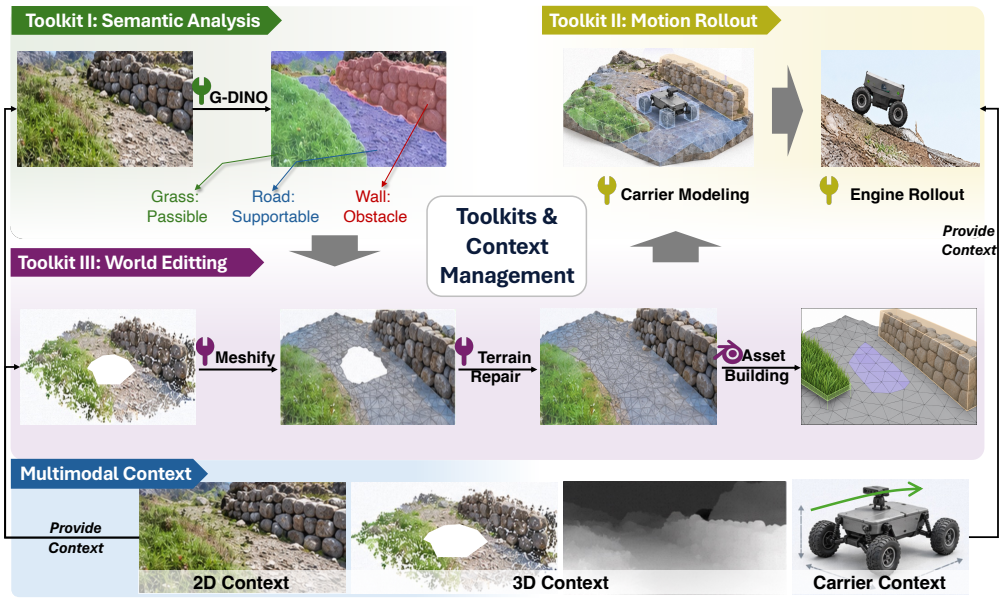


Figure 5: **Context management and toolkits in CarPhy**. CarPhy combines multimodal context with three toolkits, *e.g.*, Grounding DINO (Liu et al., 2024b). The context integrates 2D observations, 3D geometry, carrier information, and intended motion. The agent uses *semantic analysis* to infer physical roles, *world editing* to revise incomplete scene, and *motion rollout* to simulate physical outcomes. Together, they support iterative refinement of the world hypothesis.

Carrier information specifies the embodiment under which the scene is interpreted, and P_{raw} defines the intended motion whose physical outcome must be predicted.

Tool Design. CarPhy exposes three broad classes of tools to the agent: *semanticanalysis*, *world editing*, and *motion rollout*. The agent selects tools according to the uncertainty revealed by the current world hypothesis and verifier feedback. For a simple slope, the initial scene assets may support a physical rollout without further revision. If visual evidence conflicts with the reconstructed world, the agent can inspect the discrepancy, revise assets in uncertain regions, and rerun the rollout. Tool implementations, APIs, and simulation parameters are provided in Appendix B.

4 Experiments

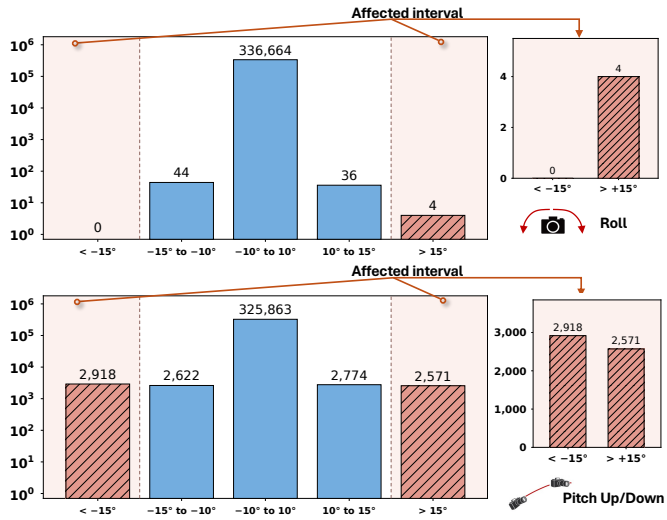
In this section, we evaluate CarPhy's ability to recognize physical events and predict the timing of carrier responses across diverse scenes and model backbones. We first introduce the evaluation set, metrics, and baselines. We then compare CarPhy with direct inference and general coding agents, and conduct ablation studies to examine the contribution of our agent. Finally, qualitative examples illustrate the corrected trajectories and effects on generation with different IWMs.

4.1 Benchmark Construction

Data Sources. We collect video segments with visual observations and camera-motion annotations from TartanGround (Patel et al., 2025), with more details provided in Appendix C.

Rare Event Coverage. The source trajectories are dominated by smooth motion. In TartanGround & TartanAir, 96.77% of pitch-angle deviations fall within $[-10°, 10°]$, whereas deviations above $15°$ and below $-15°$ account for only 0.76% and 0.87%, respectively (as shown in Figure 6). This pronounced long-tailed distribution motivates the targeted inclusion of rare physical events. Complete pitch and roll statistics are reported in Tables 1 and 2.

Sample Contruction. Each sample contains a 2-second video clip comprising 20 frames as the visual input. The camera poses of these 20 observed frames, together with those of the subsequent 20 frames and the physical event category, are provided as annotations. Given approximately 2



Distributions of roll and pitch changes relative to the camera pose 0.5 s earlier.

Table 1: Roll changes over 0.5 s.

Range (°)	Samples	%
< -15	0	0.00
-15–-10	44	0.01
-10–10	336,664	99.98
10–15	36	0.01
> 15	4	0.00

Table 2: Pitch changes over 0.5 s.

Range (°)	Samples	%
< -15	2,918	0.87
-15–-10	2,622	0.78
-10–10	325,863	96.77
10–15	2,774	0.82
> 15	2,571	0.76

Figure 6: **Distributions of roll (top) and pitch (bottom) changes relative to the camera pose** 0.5 s **earlier.** The main plots use logarithmic vertical axes. Shaded regions indicate angular changes with magnitudes greater than 15°; the smaller plots show the corresponding tail counts.

seconds of observed history, the model is required to produce a corrected camera trajectory for the following 2 seconds. For collision samples, where reliable future camera poses are unavailable, only the corresponding event labels are provided.

Throughout the benchmark, the user instruction is fixed to forward motion (the `W` command).

4.2 Evaluation Metrics

We evaluate each method on two aspects: **event recognition** and **pitch timing error**. Event recognition measures whether the method correctly identifies the physical response associated with the raw trajectory. Pitch timing error evaluates whether the predicted trajectory exhibits the expected pitch response at the correct time.

To support both metrics across physical events, we divide the samples into four classes: pitch-down (**PD**), pitch-up (**PU**), blocked (**BL**), and not-affected (**NA**). Event recognition is evaluated on all four event classes. Pitch timing error is evaluated only on PU and PD samples.

Event Recognition. For each class c, we report class-wise precision and recall:

$$\text{Precision}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c}, \qquad \text{Recall}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c}. \tag{1}$$

We additionally report mean class accuracy (mAcc), Macro-F1, and Answer Rate. mAcc is the average recall over the four classes, and Macro-F1 averages the class-wise F1 scores. Answer Rate denotes the proportion of samples for which the method returns a valid prediction.

Pitch Timing Error. We evaluate the timing accuracy of pitch responses on PU and PD samples. For a pitch threshold θ, we measure the difference between the first timestamps at which the predicted and reference trajectories reach θ:

$$e_i(\theta) = \left| t_i^{\text{pred}}(\theta) - t_i^{\text{gt}}(\theta) \right|, \quad E(\theta) = \frac{1}{N} \sum_i e_i(\theta). \tag{2}$$

If the predicted trajectory does not reach θ within the prediction horizon, we assign an error of 2 s. We evaluate θ from 1° to 6° with a step of 0.1° and report the average error.

4.3 Baselines

We evaluate direct prediction baselines and coding-agent baselines across multiple backbone models. Direct prediction utilizes the backbone model to predict the corrected future trajectory from

Table 3: **Comparison of direct inference and CarPhy agent on coarse physical response prediction.** We report recall and precision for each class (PD, PU, BL, and NA), mean class accuracy (mAcc), and Macro-F1. All metrics are shown in percentage (%). The three highest distinct values in each column are highlighted as **best**, second, and third, with ties sharing the same color. Red superscripts indicate absolute gains in percentage points over the corresponding backbone without CarPhy.

Model / Agent	PD		PU		BL		NA		mAcc	Macro-F1
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.		
Direct inference baselines										
Qwen3-VL-8B-Instruct	5.4	66.7	42.9	41.9	25.0	33.3	84.0	62.2	39.3	38.1
Qwen3-VL-32B-Instruct	0.0	0.0	31.0	68.4	0.0	0.0	97.0	59.1	32.0	29.0
Qwen3.8-27B (no-think)	8.1	**100.0**	26.2	**91.7**	62.5	45.5	**99.0**	61.5	48.9	46.1
Qwen3.8-27B	10.8	66.7	28.6	63.2	75.0	27.3	88.0	62.9	50.6	42.8
Qwen3.6-Flash (no-think)	13.5	50.0	35.7	78.9	25.0	66.7	94.0	60.6	42.1	45.1
Grok-4.7	43.2	84.2	16.7	53.8	75.0	21.4	88.0	69.8	55.7	48.5
GLM-5.2	16.2	54.5	19.0	72.7	0.0	0.0	85.0	56.7	30.1	30.8
Qwen3.6-Plus	18.9	63.6	19.0	50.0	**100.0**	17.8	76.0	66.7	53.5	39.5
Claude-Opus-4.6	5.4	40.0	16.7	53.8	37.5	25.0	93.0	59.6	38.1	34.4
GPT-5.5	21.6	66.7	28.6	52.2	25.0	11.1	88.0	65.7	40.8	40.0
Qwen3.7-Plus	18.9	70.0	16.7	43.8	87.5	12.7	72.0	67.9	48.8	36.5
GLM-5.3	16.2	54.5	21.4	56.2	0.0	0.0	94.0	58.8	32.9	32.1
GPT-5.6-Sol	29.7	57.9	28.6	60.0	50.0	13.3	80.0	67.8	47.1	43.1
Qwen3.8-Flash	18.9	63.6	23.8	71.4	75.0	26.1	89.0	64.0	51.7	44.5
Claude-Opus-4.8	5.4	40.0	16.7	58.3	37.5	25.0	94.0	59.5	38.4	34.6
Gemini-3-Pro	25.0	76.9	22.0	90.0	60.0	50.0	94.1	64.2	50.3	51.0
Kimi-K3	21.6	66.7	28.6	75.0	**100.0**	40.0	89.0	69.0	59.8	52.2
Kimi-K2.7-Agent	16.7	66.7	18.2	36.4	75.0	13.0	82.0	67.6	48.0	36.8
+ CarPhy	54.1	83.3	**66.7**	75.7	50.0	66.7	75.5	80.4	61.6 $^{\uparrow 13.6}$	67.9 $^{\uparrow 31.1}$
DeepSeek-V4-Flash	21.6	**100.0**	28.6	70.6	75.0	27.3	85.0	64.9	52.5	47.5
+ CarPhy	**59.5**	88.0	64.3	75.0	75.0	**100.0**	78.0	**81.3**	**69.2** $^{\uparrow 16.7}$	**76.4** $^{\uparrow 28.9}$
DeepSeek-V4-Pro	13.5	62.5	11.9	50.0	0.0	–.–	96.0	56.8	30.4	28.2
+ CarPhy	54.1	83.3	**66.7**	75.7	50.0	66.7	74.0	80.4	61.2 $^{\uparrow 30.8}$	67.7 $^{\uparrow 39.5}$
Gemini-3.8-Flash	47.5	86.4	31.7	86.7	16.7	12.5	97.0	69.0	48.2	50.7
+ CarPhy	56.8	87.5	**66.7**	75.7	75.0	**100.0**	78.0	**81.3**	69.1 $^{\uparrow 20.9}$	76.3 $^{\uparrow 25.6}$

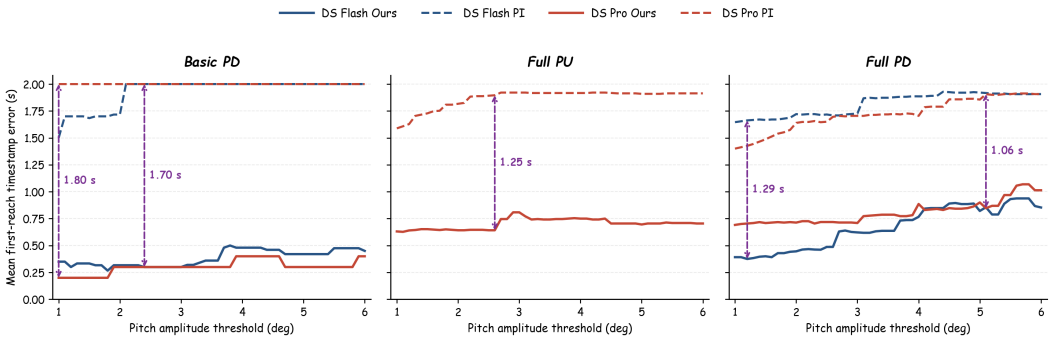


Figure 7: **Pitch timing error.** Mean absolute error between predicted and reference times to first reach pitch thresholds from 1° to 6°. Unreached thresholds incur a 2.0 s penalty. Purple arrows indicate the largest gap between our method and Pi. Pi Basic returns valid PD predictions for only six Flash samples and one Pro sample. Neither Pi Basic nor Pi with full tool access on Flash returns valid PU predictions.

historical observations and the raw trajectory. For coding-agent baselines, we use the Pi harness (Zechner, 2025) with a basic tool set and a full tool set. To make a fair comparison, all methods are evaluated using the same protocol and the same input.

Table 4: **Agent ablation with a basic coding agent.** Comparison of direct inference, Pi Basic, and our harness. Direct yields no valid timing prediction for any sample, and Pi Basic yields none for PU. All metrics are shown in percentage (%).

Metric		DeepSeek-V4-Flash			DeepSeek-V4-Pro		
		Direct	Pi Basic	**Ours**	Direct	Pi Basic	**Ours**
Answer Rate		**95.2**	13.0	87.2	**100.0**	40.1	85.0
Coarse Metric							
PD	Recall	21.6	0.0	**59.5**	13.5	0.0	**54.1**
	Precision	**100.0**	–.–	88.0	62.5	–.–	**83.3**
PU	Recall	28.6	0.0	**64.3**	11.9	0.0	**66.7**
	Precision	70.6	–.–	**75.0**	50.0	–.–	**75.7**
BL	Recall	**75.0**	0.0	**75.0**	0.0	0.0	**50.0**
	Precision	27.3	–.–	**100.0**	–.–	–.–	**66.7**
NA	Recall	**85.0**	23.0	78.0	**96.0**	55.0	74.0
	Precision	64.9	**92.0**	81.3	56.8	73.3	**80.4**
mAcc		52.5	5.8	**69.2**	30.4	13.8	**61.2**
Macro-F1		47.5	9.2	**76.4**	28.2	15.7	**67.7**
Fine Metric: Mean First-Reach Timestamp Error ↓ (s)							
PU		–.–	–.–	**0.58**	–.–	–.–	**0.64**
PD		–.–	1.93	**0.68**	–.–	2.00	**0.78**
Overall		–.–	–.–	**0.63**	–.–	–.–	**0.71**

Table 5: **Agent ablation with full tool access.** Comparison of direct inference, Pi Full Tools, and our harness. Direct yields no valid timing prediction for any sample, and Pi Full with DeepSeek-V4-Flash yields none for PU. All metrics are shown in percentage (%).

Metric		DeepSeek-V4-Flash			DeepSeek-V4-Pro		
		Direct	Pi Full	**Ours**	Direct	Pi Full	**Ours**
Answer Rate		**95.2**	84.5	87.2	**100.0**	69.0	85.0
Coarse Metric							
PD	Recall	21.6	8.1	**59.5**	13.5	8.1	**54.1**
	Precision	**100.0**	75.0	88.0	62.5	75.0	**83.3**
PU	Recall	28.6	4.9	**64.3**	11.9	2.4	**66.7**
	Precision	70.6	**100.0**	75.0	50.0	**100.0**	75.7
BL	Recall	**75.0**	0.0	**75.0**	0.0	0.0	**50.0**
	Precision	27.3	–.–	**100.0**	–.–	–.–	**66.7**
NA	Recall	85.0	**93.0**	78.0	**96.0**	90.0	74.0
	Precision	64.9	61.2	**81.3**	56.8	72.6	**80.4**
mAcc		52.5	26.5	**69.2**	30.4	25.1	**61.2**
Macro-F1		47.5	24.4	**76.4**	28.2	24.9	**67.7**
Fine Metric: Mean First-Reach Timestamp Error ↓ (s)							
PU		–.–	–.–	**0.58**	–.–	1.87	**0.64**
PD		–.–	1.81	**0.68**	–.–	1.72	**0.78**
Overall		–.–	–.–	**0.63**	–.–	1.79	**0.71**

4.4 Results and Analysis

As shown in Table 3, CarPhy consistently improves coarse physical response prediction for all four backbones. Compared with direct inference, it improves mAcc by 13.6–30.8 points and Macro-F1 by 25.6–39.5 points. Among the evaluated settings, DeepSeek-V4-Flash with CarPhy achieves the highest mAcc of 69.2 and Macro-F1 of 76.4, exceeding the strongest direct-inference baseline by 9.4 and 24.2 points, respectively. Direct inference yields mAcc of 30.1–59.8 and Macro-F1 of 28.2–52.2, indicating limited accuracy on this task. As shown in Figure 7, CarPhy also improves the timing accuracy of response prediction. Across Basic PD, Full PU, and Full PD, it achieves lower pitch timing error than the corresponding Pi agents over most pitch thresholds, with maximum reductions of 1.06–1.80 s. The improvement is especially clear in Basic PD and Full PU, where several Pi predictions reach the 2.0 s penalty, indicating less reliable timing of the physical response.

4.5 Ablation Study

Agent & Toolkit Ablation. Tables 4 and 5 show that CarPhy consistently improves physical response prediction over general coding agents. Pi Basic achieves only 5.8/13.8 mAcc and 9.2/15.7 Macro-F1 on the two backbones. Full tool access improves Pi to 26.5/25.1 mAcc and 24.4/24.9 Macro-F1. CarPhy reaches 69.2/61.2 mAcc and 76.4/67.7 Macro-F1, and also gives lower timing error on both PU and PD. Pi Basic produces no valid PU timing prediction, and Pi Full also fails on PU with DeepSeek-V4-Flash. These results show that a general coding agent cannot reliably solve this task, and that CarPhy is needed for accurate physical response prediction.

4.6 Qualitive Analysis

Pose Correction Visualization. As shown on the left of Figure 8, CarPhy produces corrected trajectories that follow the overall ground-truth trends in both PU and PD cases. The predicted pitch responses capture the upward and downward changes associated with terrain transitions, bringing the original motion closer to the expected carrier response.

IWM Generation with CarPhy. The videos in Figure 8 show how pose corrections translate into more plausible motion when integrated with IWMs. In the PU case, Lyra 2.0 without CarPhy misses the expected curb ascent, whereas CarPhy produces an upward pitch response climbing onto the sidewalk. In the PD case, LingBot World 2.0 without correction produces a viewpoint that dive

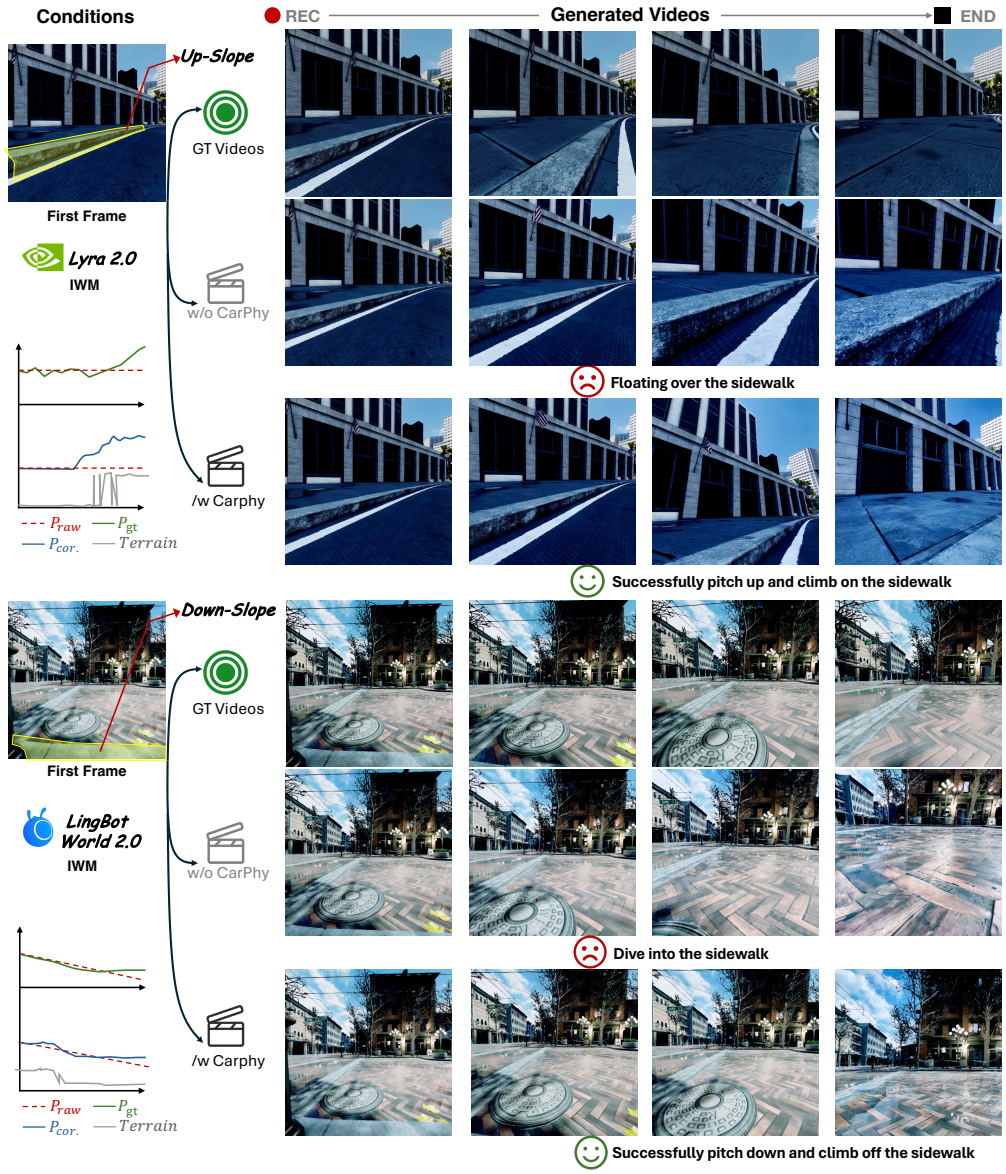


Figure 8: **Qualitative comparison of pose correction and IWM generation.** CarPhy corrects PU and PD trajectories to better align with the ground-truth motion, avoiding physically implausible behaviors such as floating above or intersecting the sidewalk. The generated videos exhibit more physically consistent motion across different IWMs.

into the sidewalk, while CarPhy guides the camera downward the curb. Both cases illustrate that CarPhy's corrections carry through to video generation across different IWM backbones.

5 Conclusion

We present CarPhy, a camera trajectory correction framework that explicitly models carrier physics for embodied world models. CarPhy reconstructs a physical scene from visual observations and uses physics simulation to estimate the response of the carrier to the intended motion. The coding agent supports this process by refining uncertain scene information based on visual evidence and verifier feedback. Experiments on our benchmark demonstrate consistent improvements in physical event recognition and trajectory correction across multiple model backbones. These results highlight the importance of explicitly modeling carrier physics for generating physically plausible camera trajectories in embodied world models.

AI USE STATEMENT

Generative AI tools were used primarily to refine the writing and presentation of the manuscript, with limited assistance in code editing and testing. All AI-assisted revisions and code changes were reviewed and verified by the authors. The authors take responsibility for the content, results, and conclusions of this work.

REPRODUCIBILITY STATEMENT

We document the CarPhy framework in Section 3 and provide tool implementations and simulation settings in Appendix B. Benchmark construction is described in Appendix C, and the direct VLM baseline protocol is specified in Appendix D. Additional tool interfaces, execution procedures, agent prompts, and physical verification criteria are provided in Appendices E, F, G, and H. Reproducing experiments with proprietary models requires access to the corresponding model versions and services.

ETHICS STATEMENT

CarPhy is intended to improve the physical plausibility of camera motion in interactive world models. Our evaluation uses sequences from TartanGround and additional collision scenarios generated using simulators and generative models. Any redistribution of source data or derived assets must respect the applicable licenses and terms of use. Although CarPhy improves physical consistency in the evaluated settings, its predictions depend on reconstructed scenes, estimated carrier properties, and simulation assumptions. They do not establish the safety or reliability of real-world robotic systems and should not be treated as a substitute for validation before deployment.

REFERENCES

Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in Atari. In *NeurIPS*, 2024.

Jiaming Bian, Bingliang Li, Yuehao Wu, Pichao Wang, Zhi Wang, Hailan Ma, Huadong Mo, and Zhenhong Sun. Look-before-move: Narrative-grounded world visual attention in dynamic 3d story worlds. *arXiv preprint arXiv:2606.26964*, 2026.

Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *ICML*, 2024.

Yuanhao Cai, Kunpeng Li, Menglin Jia, Jialiang Wang, Junzhe Sun, Feng Liang, Weifeng Chen, Felix Juefei-Xu, Chu Wang, Ali Thabet, Xiaoliang Dai, Xuan Ju, Alan Yuille, and Ji Hou. Phygdpo: Physics-aware groupwise direct preference optimization for physically consistent text-to-video generation. In *ECCV*, 2026.

Yiwen Chen, Guosheng Lin, and Chi Zhang. Code world model: Coding agent as world brain. *arXiv preprint arXiv:2608.25927*, 2026.

Kexu Cheng, Zicheng Liu, Mingju Gao, Chunhe Song, and Hao Tang. Physrag: Enhancing physics-awareness in video generation via retrieval-augmented generation. In *ECCV*, 2026a.

Zhijing Cheng, Xuancheng Zhang, Donglin Di, Chen Wei, Hao Li, and Xun Yang. Moca: Modeling object consistency for 3d camera control in video generation. In *ICLR*, 2026b.

Erwin Coumans and Yunfei Bai. PyBullet: A python module for physics simulation for games, robotics and machine learning. `https://pybullet.org`, 2016–2021.

Lin Geng Foo, Mark He Huang, Alexandros Lattas, Stylianos Moschoglou, Thabo Beeler, and Christian Theobalt. Physical simulator in-the-loop video generation. In *CVPR*, 2026.

Frédéric Fortier-Chouinard, Yannick Hold-Geoffroy, Valentin Deschaintre, Matheus Gadelha, and Jean-François Lalonde. Gimbaldiffusion: Gravity-aware camera control for video generation. In *ECCV*, 2026.

Anthony Hu, Lloyd Russell, Hudson Yeo, Zak Murez, George Fedoseev, Alex Kendall, Jamie Shotton, and Gianluca Corrado. Gaia-1: A generative world model for autonomous driving. *arXiv preprint arXiv:2309.17080*, 2023.

Fan Jiang, Zhaoxu Sun, Mengchao Wang, Ziyu Zhu, Chiyu Wang, Yunpeng Zhang, Wenlin Liu, Yun Wang, Xue Zheng, Rui Sun, et al. Abot-world-0: Infinite interactive world rollout on a single desktop gpu. *arXiv preprint arXiv:2607.19191*, 2026.

JoungBin Lee, Jaewoo Jung, Jisang Han, Takuya Narihira, Kazumi Fukuda, Junyoung Seo, Sunghwan Hong, Yuki Mitsufuji, and Seungryong Kim. 3d scene prompting for scene-consistent camera-controllable video generation. In *ICLR*, 2026.

Haodong Li, Tianfei Ren, Xiaoxiao Ma, Chunmei Qing, Zhen Fang, Sipeng He, Ziyu Guo, Haoyu Wu, Juanxi Tian, Yihang Zou, et al. Videococo: Code-as-cot for physically-consistent video generation via an agentic dual-engine system. *arXiv preprint arXiv:2607.27380*, 2026a.

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Luyao Zhang, Mingrui Yu, Zelin Gao, Nan Xue, Boyu Zhou, Xing Zhu, Mingyu Ding, Yujun Shen, and Yinghao Xu. Causal world modeling for robot control. In *RSS*, 2026b.

Qianru Li, Xuyang Chen, Erkin Türköz, Xuqin Wang, Tao Wu, Lu Liu, and Yanfeng Zhang. Cinematraj: Composing atomic camera trajectories for 3d scenes with llm agents. In *ACMMM*, 2026c.

Shaowei Liu, Zhongzheng Ren, Saurabh Gupta, and Shenlong Wang. Physgen: Rigid-body physics-grounded image-to-video generation. In *ECCV*, 2024a.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In *ECCV*, 2024b.

Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science robotics*, 2022.

Antonio Montanaro, Luca Savant Aira, Emanuele Aiello, Diego Valsesia, and Enrico Magli. Motioncraft: Physics-based zero-shot video generation. In *NeurIPS*, 2024.

Manthan Patel, Fan Yang, Yuheng Qiu, Cesar Cadena, Sebastian Scherer, Marco Hutter, and Wenshan Wang. Tartanground: A large-scale dataset for ground robot perception and navigation. In *IROS*, 2025.

Tianchang Shen, Sherwin Bahmani, Kai He, Sangeetha Grama Srinivasan, Tianshi Cao, Jiawei Ren, Ruilong Li, Zian Wang, Nicholas Sharp, Zan Gojcic, et al. Lyra 2.0: Explorable generative 3d worlds. In *SIGGRAPH Asia*, 2026.

Seunghyun Shin, Jifei Song, Wooseok Jeon, Hae-Gon Jeon, and Jiankang Deng. Trimotion: Modality-agnostic camera control for video generation. In *ECCV*, 2026.

Wenqiang Sun, Haiyu Zhang, Haoyuan Wang, Junta Wu, Zehan Wang, Zhenwei Wang, Yunhong Wang, Jun Zhang, Tengfei Wang, and Chunchao Guo. Worldplay: Towards long-term geometric consistency for real-time interactive world modeling. In *ICML*, 2026.

Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion models are real-time game engines. In *ICLR*, 2025.

Hanlin Wang, Hao Ouyang, Qiuyu Wang, Wen Wang, Qingyan Bai, Ka Leong Cheng, Yue Yu, Yixuan Li, Yihao Meng, Zichen Liu, et al. Worlddirector: Building controllable world simulators with persistent dynamic memory. *arXiv preprint arXiv:2607.02517*, 2026a.

Hongyu Wang, Jingquan Wang, Bocheng Zou, Radu Serban, and Dan Negrut. Coding agent is good as world simulator. *arXiv preprint arXiv:2605.14398*, 2026b.

Jing Wang, Ao Ma, Ke Cao, Jun Zheng, Jiasong Feng, Zhanjie Zhang, Wanyuan Pang, and Xiaodan Liang. Wisa: World simulator assistant for physics-aware text-to-video generation. In *NeurIPS*, 2025.

Yukun Wang, Ruihuang Li, Jiale Tao, Shiyuan Yang, Liyi Chen, Zhantao Yang, Handz, Yulan Guo, Shuai Shao, and Qinglin Lu. OmniCamera: A unified framework for multi-task video generation with arbitrary camera control. In *ECCV*, 2026c.

Wayve. GAIA-3: Scaling world models to power safety and evaluation. `https://wayve.ai/thinking/gaia-3/`, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

Shenghai Yuan, Jinfa Huang, Yongqi Xu, Yaoyang Liu, Shaofeng Zhang, Yujun Shi, Ruijie Zhu, Xinhua Cheng, Jiebo Luo, and Li Yuan. Chronomagic-bench: A benchmark for metamorphic evaluation of text-to-time-lapse video generation. In *NeurIPS*, 2024.

Yu Yuan, Xijun Wang, Tharindu Wickremasinghe, Zeeshan Nadir, Bole Ma, and Stanley Chan. Newtongen: Physics-consistent and controllable text-to-video generation via neural newtonian dynamics. In *ICLR*, 2026.

Mario Zechner. Pi Coding Agent. `https://github.com/earendil-works/pi`, 2025.

Qiyuan Zhang, Biao Gong, Shuai Tan, Zheng Zhang, Yujun Shen, Xing Zhu, Yuyuan Li, Kelu Yao, Chunhua Shen, and Changqing Zou. Physrvg: Physics-aware unified reinforcement learning for video generative models. In *ECCV*, 2026.

Xiangdong Zhang, Jiaqi Liao, Shaofeng Zhang, Fanqing Meng, Xiangpeng Wan, Junchi Yan, and Yu Cheng. Videorepa: Learning physics for video generation through relational alignment with foundation models. In *NeurIPS*, 2025.

Haoyu Zhao, Jiaxi Gu, Haoran Chen, Qingping Zheng, Yeying Jin, Hongyi Yang, Junqi Cheng, Yuang Zhang, Zenghui Lu, Huan Yu, et al. Cameranoise: Enabling faithful camera control in video diffusion through geometry-flow-guided noise warping. In *ICML*, 2026.

A Formal Task Formulation

Pose Representation. We represent the camera pose at time t as

$$p_t = \big((x_t, y_t, z_t), (\phi_t, \theta_t, \psi_t)\big), \tag{3}$$

where (x_t, y_t, z_t) gives the camera position in the world coordinate system. The angles $(\phi_t, \theta_t, \psi_t)$ describe its orientation. A camera trajectory of length T is written as

$$\mathcal{P} = \{p_t\}_{t=1}^{T}. \tag{4}$$

Motion Intent. As illustrated in Figure 4, the raw trajectory specifies the motion requested by the user. We construct this trajectory from the user input and project it onto the horizontal plane of the user coordinate frame, with pitch set to zero:

$$\mathcal{P}_{\text{raw}} = \big\{\big((x_t, y_t, 0), (\phi_t, 0, \psi_t)\big)\big\}_{t=1}^{T}. \tag{5}$$

Here, zero height corresponds to the camera height at the start of prediction, and zero pitch specifies a level orientation. This construction requires no future ground truth poses and applies to all samples, including collision scenarios. Thus, $\mathcal{P}_{\text{raw}}$ describes the intended motion before the physical response of the carrier is considered.

Pose Correction Task. CarPhy receives historical visual observations $\mathcal{O}_{t-H+1:t}$, their camera poses $\mathcal{P}_{\text{hist}}$, carrier information c, and the raw future trajectory $\mathcal{P}_{\text{raw}}$. The task is written as

$$\mathcal{F}\left(\mathcal{O}_{t-H+1:t}, \mathcal{P}_{\text{hist}}, c, \mathcal{P}_{\text{raw}}\right) \rightarrow \mathcal{P}_{\text{corr}}, \tag{6}$$

where

$$\mathcal{P}_{\text{corr}} = \left\{\big((\hat{x}_t, \hat{y}_t, \hat{z}_t), (\hat{\phi}_t, \hat{\theta}_t, \hat{\psi}_t)\big)\right\}_{t=1}^{T}. \tag{7}$$

The corrected trajectory follows the motion intent specified by $\mathcal{P}_{\text{raw}}$ under the physical constraints of the carrier and the environment. For terrain interaction, the physical response is mainly reflected by changes in height and pitch. A blocking event limits the forward motion according to the physical interaction. For an unaffected sample, the resulting trajectory remains consistent with the original motion. CarPhy can return ABSTAIN when the available evidence is insufficient to determine a reliable physical response.

B Tool Interfaces and Implementation Details

Scene inspection. The host constructs the initial scene asset and runs a default physics rollout before invoking the agent. The agent receives the historical observations, the raw trajectory, the default candidate, and its verification result. If the images suggest an error in the scene asset, the agent calls `inspect_semantic_asset`. This tool creates a fixed scene index and returns a short list of object groups near the intended path. The summary includes stable group identifiers, heights, and distances to the path. A scene hash identifies the asset version inspected by the agent.

Asset repair. The agent proposes asset changes through a structured plan submitted to `apply_semantic_plan`. Each operation specifies a target group, a semantic class, an allowed action, and supporting visual evidence. The impplementation defines five classes: `SUPPORT`, `RIGID_OBSTACLE`, `SOFT_TRAVERSABLE`, `VISUAL_ONLY`, and `UNKNOWN`. For example, the agent may disable collision for vegetation supported by the images or remove a labeled canopy from the support surface. Before applying a plan, the executor checks the scene hash, target group, evidence, and permitted class–action pair. It preserves the source meshes and records the revised asset and repair result.

Rollout and verification. Through `run_python`, the agent can call the exposed geometry and physics functions.The `rollout_intent` interface extracts speed and turning commands at each time step from the raw trajectory, runs the carrier in PyBullet (Coumans & Bai, 2016–2021).

The verifier checks the simulated trajectory and returns any violations to the agent. For uncertain terrain, each available scene hypothesis is simulated separately. A consistency check compares the resulting physical events. A world-consistency check compares the resulting physical events.

The agent can accept the default candidate through `accept_baseline` or submit a revised asset through `submit_final`. If no trajectory file is submitted, the system runs a new rollout using that asset.

Simulation settings.

Table 6 summarizes the settings used in our experiment.

Table 6: Default simulation and agent settings.

Parameter	Default value
Future horizon	$2\,\mathrm{s}$
Simulation step	$1/240\,\mathrm{s}$
Settling steps	480
Trajectory output rate	$24\,\mathrm{Hz}$
Lateral friction coefficient	1.1
Gravitational acceleration	$9.8\,\mathrm{m/s^2}$
Maximum agent steps	10
Python call timeout	$180\,\mathrm{s}$
Model temperature	0

For evaluation, we resample the ground truth poses provided by the benchmark over the future interval of $2\,\mathrm{s}$ to match the output trajectory timestamps at $24\,\mathrm{Hz}$.

C Benchmark Construction Details

We select continuous sequences containing both visual observations and camera-motion annotations from TartanGround (Patel et al., 2025). We further supplement the source data with collision scenarios generated using simulators and generative models, since these events are rare in the original datasets. All samples are manually curated. The final benchmark contains 187 samples covering diverse carriers, environments, and physical events. Table 7 summarizes its detailed composition.

Each sample contains a 2-second sequence of 20 historical visual observations and a 4-second sequence of 40 camera poses spanning both the historical and prediction intervals. Given the historical observations, the model is required to correct the 20-frame trajectory over the subsequent 2 seconds. Each camera pose is represented as a 6-DoF seven-dimensional tuple consisting of a three-dimensional translation and a unit quaternion. All poses are stored in a global world coordinate frame, where the x- and y-axes represent fixed horizontal directions in the environment. During experiments, all poses are transformed into a fixed user coordinate frame whose origin is the camera position at the start of prediction and whose axes point horizontally forward, left, and vertically upward.

For collision samples, camera motion after the critical event does not admit a unique and reliable reference trajectory. We therefore provide only event labels for these samples and use them solely for event recognition evaluation. The remaining 179 samples contain reference future poses and can be used for both event recognition and pitch timing error evaluation.

Table 7: Composition of the CarPhy benchmark. Proportions for *Event label* and *Environment* are computed over all 187 samples, whereas proportions for *Affected events* are computed over the 87 affected samples. Percentages may not sum to exactly 100% due to rounding.

Dimension	Category	Samples	Proportion
All	Total	187	100.00%
Event label	Affected	87	46.52%
	Unaffected (NA)	100	53.48%
Environment	Urban outdoor	79	42.25%
	Indoor	65	34.76%
	Wild	43	22.99%
Affected events	PU	42	48.28%
	PD	37	42.53%
	BL	8	9.20%

The benchmark is approximately balanced between affected and unaffected samples, preventing overall accuracy from being dominated by class priors. The affected subset covers slopes, curbs, and collisions. Collision cases account for 9.20% of the affected subset and evaluate whether a method can recognize critical events without reliable future reference trajectories.

D DIRECT VLM PREDICTION PROTOCOL

Task and inputs. The direct classification baseline predicts one physical-event label for the next two seconds without invoking tools, reconstructing a scene, running physics simulation, or receiving verifier feedback. The implementation uses four labels: `PITCH_DOWN`, `PITCH_UP`, `BLOCKED`, and `NOT_AFFECTED`, corresponding to PD, PU, BL, and NA, respectively. Figures 9 and 10 provide English translations of the Chinese prompts used by the implementation; output labels and delimiters are unchanged.

Visual and motion context. For a history containing M images, the implementation selects $n = \min(5, M)$ images with indices

$$i_j = \mathrm{round}\left(\frac{j(M-1)}{\max(n-1,1)}\right), \qquad j = 0, \ldots, n-1. \tag{8}$$

Duplicate indices are removed, and the current image is appended if it is not already selected. If no history images are available, the current image alone is used when available. The selected images are ordered chronologically and paired with historical/current-frame labels. These labels indicate order in the selected sequence, rather than elapsed seconds. The user message additionally contains a textual summary of current speed, historical path shape and distance, current pitch, future motion intent, and a wheel-size description. The classification prompt does not include numerical pose arrays or reconstructed geometry.

The motion summary uses the configured carrier speed when available. Otherwise, it estimates speed from historical horizontal path length and duration. A path is described as straight when its net horizontal displacement divided by its path length exceeds 0.98; otherwise it is described as turning. The current attitude is described as approximately level when the absolute pitch is below 2°; otherwise the summary reports nose-up or nose-down motion according to the implementation's pitch convention. Sample-dependent fields and conditional lines are indicated in Figure 9.

Native vision and visual bridge. For a vision-capable backbone, images, frame labels, and the motion summary are supplied directly in the user message. When a separate visual encoder model is configured, it first summarizes the images using Figure 10; the classification backbone receives the motion summary followed by this visual summary as text. The visual-summary call uses temperature 0 and a maximum of 768 output tokens. In the inspected DeepSeek Flash and Pro `direct-class` run configurations, the classifier uses temperature 0 and a maximum of 8192 output tokens. Flash

receives images directly; Pro uses a Flash visual-summary bridge. There is one classification request per example, without an iterative tool-feedback loop.

Output parsing. The requested response is a single label enclosed in `<answer>` and `</answer>` tags. Parsing is case-insensitive: the parser first collects labels from matching answer tags; if no such tag is found, it searches for complete occurrences of the four allowed label names in the response. A response is valid only when the resulting set contains exactly one distinct allowed label. Repeated occurrences of the same label are accepted; synonyms are not mapped to labels. Empty responses, responses without an allowed label, and responses with multiple distinct extracted labels are assigned `NO_ANSWER`. Thus, the requested output format is stricter than the parser's fallback behavior. This classification arm returns an event label rather than a corrected trajectory.

E Tool Interfaces and Structured Inputs and Outputs

CarPhy exposes two levels of interfaces. The first consists of model-facing tools with structured argument schemas. The second consists of geometry and physics functions that the agent can invoke through Python execution. This distinction separates the agent's communication interface from the numerical operations performed by the underlying toolkit.

E.1 Model-Facing Tools

The native tool interface includes six functions: `inspect_semantic_asset`, `apply_semantic_plan`, `run_python`, `accept_baseline`, `submit_final`, and `abstain`. The `accept_baseline` interface is exposed only when the host-baseline mode is enabled. All six argument schemas reject additional top-level properties. Nested objects such as the semantic plan and uncertainty record may contain additional fields, subject to subsequent execution-time checks.

Semantic asset inspection. The tool `inspect_semantic_asset` requires a `reason` string describing the discrepancy that motivates inspection. It stages the default asset into a frozen semantic scene index. If an index has already been staged, it is reused.

The returned object contains two fields: `stage`, which identifies the staged scene and its digest, and `summary`, which describes the candidate groups. The summary includes `scene_digest`, `terrain_mesh`, `n_components`, `n_groups`, and `candidate_groups`. Candidate groups expose stable identifiers and geometric attributes without returning the full mesh. Under the default inspection settings, groups within $2.0\,\mathrm{m}$ of the intended-motion corridor are considered, and at most 40 groups are returned. Groups are ordered by corridor distance, height, and identifier. Component membership is summarized through a component count and digest.

Declarative semantic repair. The tool `apply_semantic_plan` requires a `plan` object. Its nested structure is validated against the previously staged scene. The plan contains:

- `schema`: the protocol identifier `carphy-semantic-plan/1`;
- `scene_index_hash`: the digest of the scene being edited;
- `operations`: a list of semantic editing operations.

Each operation specifies a `target`, `semantic_class`, `action`, `reason`, and `evidence`. An optional `labels` list provides segmentation labels; this list is required when removing a terrain support region. The executor checks that the scene digest matches, the target is valid, the reason and evidence are nonempty, and the semantic class permits the requested action.

Support targets accept only `VISUAL_ONLY`/`remove_from_support`, with segmentation labels identifying the affected surface. Collision groups cannot directly remove terrain support. The source meshes remain unchanged; the executor produces a revised asset and a repair report.

After constructing the revised asset, the host runs `rollout_intent` and checks the resulting trajectory. The tool returns:

Direct VLM Classification — System Prompt

```
You are a wheeled carrier equipped with a front-facing camera, moving according to the
given motion intent.  You will see historical and current frames in chronological order
(the last image is the current frame), together with a textual description of the current
motion and the motion intent for the next 2 seconds.

Determine the carrier's traversability and pitch attitude over the next 2 seconds.  Output
only one label, enclosed in <answer></answer>:

BLOCKED  An excessively close, impassable obstacle lies directly ahead along the path
over the next 2 seconds; the carrier will be blocked and can hardly continue forward.
Impassable obstacles include walls, solid barriers, large rigid bodies that cannot be
driven over or bypassed.  Grass, bushes, fallen leaves, uneven ground, and low vegetation
that can be driven over do not count as impassable.

PITCH_UP  Passage is unobstructed, but the environment will raise the front of the carrier
(going uphill or ascending a slope or step).

PITCH_DOWN  Passage is unobstructed, but the environment will lower the front of the
carrier (going downhill or descending a slope or step).

NOT_AFFECTED  Passage is unobstructed, and the environment will not noticeably change the
carrier's pitch.

BLOCKED takes priority over the other three labels.  Do not use Markdown.  Do not explain.
Do not output any numbers.
```

Direct VLM Classification — User-Message Template

```
[Historical image]  Historical frame (t-{selected-frame offset})
[Additional selected historical images and their labels]
[Current image]  Current frame (t)

Current forward speed is approximately {speed, 1 decimal} meters per second.
The historical trajectory is {straight / turning}, with a total distance of approximately
{distance, 1 decimal} meters.
[One attitude sentence:]
The carrier is currently approximately level.
[or:]  The front is currently {raised / lowered} by approximately {absolute pitch, 0
decimals} degrees.
Motion intent for the next {horizon, 0 decimals} seconds:  {straight / turning / forward},
covering approximately {distance, 1 decimal} meters.
The carrier is a small wheeled cart with a wheel diameter of approximately {wheel-size
value, 2 decimals} meters.
Now answer directly in the required format.
```

Figure 9: **Direct VLM classification prompts.** English translations of the implemented system prompt and user-message template. Braced fields and bracketed image/branch descriptions are explanatory placeholders, not literal input. Speed and trajectory-summary lines are included only when the required data are available.

Optional Visual Bridge — Summary Prompt

```
Summarize the information in these chronologically ordered frames (the last image is the
current frame) that is relevant to the vehicle's pitch attitude, slopes, and whether an
impassable obstacle lies directly ahead over the next 2 seconds.

[Selected images, each followed by its historical/current-frame label]
```

Classification Backbone — User Message with Visual Summary

```
[Motion-summary text from the direct classification user-message template, including the
final instruction to answer]

Visual model summary:
{visual summary returned by the bridge model}
```

Figure 10: **Visual-summary bridge for a text-based classification backbone.** The bridge receives the selected images and their temporal-order labels. The classifier retains the same system prompt as in Figure 9, but receives the motion summary and the bridge's textual output instead of images.

Table 8: Allowed semantic class–action pairs. These pairs constrain the interpretation of reconstructed geometry.

Semantic class	Allowed action
SUPPORT	keep_support
RIGID_OBSTACLE	keep_collision
SOFT_TRAVERSABLE	disable_collision
VISUAL_ONLY	remove_from_support
UNKNOWN	no_change

- `manifest`: the revised asset-manifest path;
- `registered_for_final`: whether the sanity check passed;
- `trajectory` and `trajectory_file`: the simulated candidate and its saved location;
- `check`: the verification result;
- `repair_report`: the semantic repair record;
- `semantic_assetization`: assetization metadata.

A repaired manifest is added to the approved-manifest set only when the sanity check passes. Consequently, producing an edited asset does not by itself establish that the asset supports a valid physical prediction.

Python execution. The tool `run_python` requires a single string field, `code`. The host saves the submitted source and executes it in a subprocess whose working directory is the case workspace. Its response contains the process exit code and captured standard output and standard error. The implementation retains the last 4,000 characters of standard output and the last 1,500 characters of standard error. Timeouts, process-launch failures, and unavailable sandbox execution are returned as error messages.

Each call starts a separate process. Therefore, files persist across calls, whereas Python variables and in-memory objects do not. Intermediate assets and trajectories must be written to the workspace when they are needed in later iterations.

Baseline acceptance. The tool `accept_baseline` requires:

- `reason`: a string explaining acceptance;
- `clearance_obstacle_class`: one of `rigid_wall`, `vegetation`, `none`, or `unknown`.

This action selects the host-precomputed candidate. The tool description instructs the agent to use it when the baseline has passed its physical checks, and no concrete visual or geometric evidence contradicts the asset. It is an acceptance action rather than an additional scene-construction or simulation tool.

Final submission. The tool `submit_final` requires `asset_manifest_file`. Its optional fields are `corrected_pose_trajectory_file`, `analysis`, `correction_reason`, `uncertainty`, `semantic_check`, and `clearance_obstacle_class`. The first four optional textual fields are strings where applicable; `uncertainty` and `semantic_check` are objects. The obstacle-class field uses the same four values as `accept_baseline`.

When the trajectory file is omitted, the host generates a trajectory from the submitted asset through the canonical `rollout_intent` procedure. When a trajectory is supplied, it is read from the specified file. Submission initiates host-side evaluation; it is not equivalent to successful verification.

Abstention. The tool `abstain` requires a `reason` string and an `evidence` array of strings. It allows the agent to return no physical candidate when the available evidence is insufficient or when support validity remains unresolved. Abstention is recorded separately from a verified trajectory.

E.2 Executable Geometry and Physics Interfaces

The agent accesses numerical functions through `carphy.tools.terrain3d`, conventionally imported as `T`. The workspace documentation provides the detailed function signatures, argument names, and return fields. These functions are used inside `run_python`; they are not additional native model-facing tool names.

Observation access. `T.load_observations` loads reconstructed depth, confidence, camera extrinsics, intrinsics, RGB observations, and coordinate and scale information. `T.colored_cloud` returns points and associated colors for joint visual and geometric inspection. `T.camera_up` estimates an up direction from camera poses and reports its angular spread.

Scene construction. The toolkit supports TSDF fusion, terrain–obstacle separation, terrain completion, and collision-asset construction. `T.fuse_tsdf` consumes multi-view observations and reconstruction parameters and produces a fused mesh. `T.separate_terrain` separates terrain and obstacle meshes. `T.complete_terrain` constructs an elevation representation and records observation coverage. `T.build_collision_asset` converts the terrain representation and obstacle mesh into a simulator-compatible manifest.

The convenience interface `T.build_terrain` combines these operations and returns the asset manifest together with processing records. Terrain and obstacles are represented separately so that vertical obstacles are not collapsed into a single-valued elevation surface.

Physical queries and rollout. `T.probe_support` queries the support height at a planar location. A return value of `None` indicates the absence of support geometry in the queried asset; it does not alone establish that the real scene contains a cliff. `T.contact_query` evaluates contact and penetration for a specified carrier pose.

`T.rollout` simulates carrier motion under the specified asset, initial pose, control intent, and carrier configuration. Its result contains the trajectory and physical outcome diagnostics. The trajectory height follows the camera-height convention.

`T.rollout_intent` provides the canonical task-level interface: It receives an asset and observation record, derives the controls from the raw trajectory, simulates the carrier, aligns the trajectory with the requested time window, and returns the trajectory and verification result. An optional output path saves the candidate for subsequent submission.

Verification interface. The Python-level `T.check_poses` interface returns a check object with `passed`, `violations`, and `metrics` attributes. Physical verification requires a scene asset. A kinematic check without an asset is not treated as evidence that the trajectory is physically supported by the reconstructed scene.

F Harness Execution and Feedback

F.1 Initial Context and Persistent Workspace

The initial task message combines the observation record with host-generated context. Its principal fields are: `task`, `observation`, `baseline_rollout`, `visual_safety_summary`, `visual_clearance_obstacle_class`, `semantic_scene`, `workspace_root`, `workspace_files`, and `budget`. The budget specifies the maximum number of agent steps and the Python execution timeout.

The workspace stores the initial state, observation record, executed code, scene assets, candidate trajectories, and run logs. This file-based state allows subsequent calls to reuse intermediate results without assuming a persistent Python interpreter.

For a vision-capable agent, selected historical images and the current image are attached to the initial message. For a text-only backbone, the harness provides visual-model feedback. The agent can request image inspection by printing an image declaration of the form `CARPHY_IMAGE:`

<absolute path> | <question>. The host processes the declared image and returns the visual feedback to the agent.

F.2 Baseline Acceptance and Asset Revision

In the host-baseline configuration, the host can supply a default physics candidate and its verification result before the agent acts. When the candidate passes and no visual–geometric conflict is identified, the task message instructs the agent to accept the baseline without repeating the rollout or verifier.

When the visual interpretation conflicts with the physical asset, the agent can inspect the staged scene and propose a declarative repair. For example, vegetation represented as a rigid collision obstacle can motivate a supported SOFT_TRAVERSABLE/disable_collision operation. The repair must reference the staged scene and a valid target, and must include supporting evidence. The host constructs the revised asset and tests it through a new rollout before registering it for final use.

If the default asset fails support or physical sanity checks, the agent may attempt an evidence-supported semantic repair. If reliable support cannot be established, the protocol permits abstention.

F.3 Executable Feedback and Completion

Python errors and tool exceptions are returned as observations that the agent can use to revise its next action. For candidate trajectories, the host checks the submitted motion against the raw intent, carrier configuration, current reference pose, and selected physical asset. Verification feedback separates the pass indicator, violation descriptions, and numerical metrics.

Asset errors are not silently interpreted as successful physical verification. If an asset-dependent check fails to execute, the implementation may retain kinematic diagnostics while explicitly reporting that the asset was unusable and only kinematics and intent were checked.

The harness also records physical outcomes across the supplied world hypotheses. In the result record, completion is defined by the conjunction of physical passage and world consistency. Semantic forecasts are retained as diagnostics rather than being sufficient evidence of physical validity. Thus, a successful tool execution, a submitted candidate, and a completed physical prediction are distinct events.

G Harness Prompt Specification

The harness combines a system instruction with a structured, case-dependent task message. The system instruction describes the correction objective, observation semantics, available computational interfaces, and requirements on trajectory provenance. The task message supplies the current observation, workspace state, execution budget, and configuration-dependent execution guidance.

The following figures provide English translations of selected implementation passages. They are organized by function for readability and do not reproduce the complete system prompt. Identifiers, field names, and tool names retain their implemented form. Braced fields denote sample-dependent content.

H Physical Verification and Acceptance Criteria

Our verifier evaluates a candidate trajectory under an explicit scene hypothesis and a specified carrier model. It implements numerical checks rather than a VLM-based judgment. A passing result indicates that the candidate satisfies the implemented checks under the supplied scene; it does not establish that the scene reconstruction or the predicted future response matches the real environment.

Inputs and outputs. The verifier receives a candidate camera trajectory $\hat{P} = \{(t_i, \mathbf{x}_i, \boldsymbol{\theta}_i)\}_{i=1}^{N}$, a scene asset $\mathcal{S}$, a carrier specification, the current reference pose, and the raw trajectory P_{raw} encoding the requested motion. Here, $\mathbf{x}_i = (x_i, y_i, z_i)$ denotes camera position, and $\boldsymbol{\theta}_i$ contains its orientation. The reproduction configuration additionally specifies the commanded speed, yaw

rate, and motion intent used for an independent simulator rollout. The output contains a Boolean `passed` flag, a list of `violations`, and numerical `metrics`. The flag is true only when the violation list is empty. An empty trajectory or the absence of an asset-backed physical check does not constitute a successful verification.

Temporal coverage and trajectory regularity. Timestamps must be strictly increasing. When a raw trajectory is provided, the candidate must contain at least $\max(4, \lceil N_{\mathrm{raw}}/2 \rceil)$ samples, and its start and end timestamps must match those of P_{raw} within 10^{-6} s. Consecutive positions must satisfy

$$\|\mathbf{x}_{i+1} - \mathbf{x}_i\|_2 \leq 0.60\,\mathrm{m}.$$

For trajectories with at least three samples, the verifier also bounds the absolute vertical acceleration by $90\,\mathrm{m/s^2}$ and the absolute pitch rate by $700\,\mathrm{deg/s}$. These quantities are computed by finite differences, with time intervals clamped below at 10^{-3} s. These thresholds serve as trajectory sanity checks; they are not tolerances on prediction error against ground-truth future poses.

Preservation of motion intent. The verifier constrains lateral deviation from the requested trajectory. Let $\mathbf{u}$ be the normalized horizontal displacement between the first and last raw poses, and let $\mathbf{u}_\perp$ be its perpendicular direction. For each candidate sample, the raw pose with the closest timestamp is selected, and the verifier requires

$$\left| \mathbf{u}_\perp^\mathsf{T} \left(\hat{\mathbf{x}}_{i,xy} - \mathbf{x}^{\mathrm{raw}}_{j(i),xy} \right) \right| \leq 0.35\,\mathrm{m}.$$

For negligible raw horizontal displacement, the implementation uses the positive x direction as $\mathbf{u}$. Longitudinal lag is recorded but is not itself a rejection criterion, allowing a physical obstacle to stop the carrier before it reaches the requested position. This check preserves a lateral motion corridor rather than requiring exact pointwise agreement with the raw trajectory.

Contact and penetration. At each eligible sample, the camera height is converted to the chassis height using the carrier-specific camera offset. The carrier is placed in the scene using the supplied pitch and roll; the instantaneous contact query uses zero yaw. The collision engine then reports contact points and signed contact distances. A sample is considered in contact when at least one contact point is returned. Its penetration depth is

$$d_i = \max\left(0, -\min_k d_{ik}^{\mathrm{contact}}\right),$$

with zero penetration when no penetrating contact is reported. To accommodate penetration already present in the reference placement, the verifier uses its penetration depth d_{ref} as a bias and requires

$$\max\left(0, \max_i d_i - d_{\mathrm{ref}}\right) \leq 0.10\,\mathrm{m}.$$

Contact is marked indeterminate when the queried ground height is unavailable, or when the chassis is more than $0.15\,\mathrm{m}$ below the queried surface while descending by more than $0.02\,\mathrm{m}$ between samples. Such samples are excluded from the direct contact queries rather than being treated as confirmed valid contacts.

Independent rollout reproduction. When a reproduction configuration is supplied, the verifier reruns the simulator from the reference pose using the same scene, carrier, motion command, and requested duration. The replayed height is linearly interpolated at the candidate's elapsed timestamps, and the verifier requires

$$\max_i |\hat{z}_i - z_{\mathrm{replay}}(t_i - t_1)| \leq 0.20\,\mathrm{m}.$$

This reproduction test compares vertical position only. It does not impose a replay-agreement threshold on pitch, roll, or the full pose, and the verifier does not contain a separate absolute rollover-angle threshold.

Without a successful height-reproduction check, the determinate contact sequence is subject to three additional conditions: its first sample must be in contact, any non-contact samples must form a

suffix, and non-contact samples must account for no more than 60% of the determinate samples. These three conditions are bypassed when height reproduction succeeds.

A separate heuristic checks unsupported descent, where unsupported samples include both indeterminate and non-contact samples. For the last contiguous unsupported run containing at least four samples, a height drop greater than $0.15\,\mathrm{m}$ together with a maximum absolute vertical acceleration below $3\,\mathrm{m/s^2}$ is rejected. This detects approximately constant-speed unsupported descent; it is not a complete test of ballistic motion.

Scene coverage. When the asset provides spatial bounds, the verifier checks that the candidate trajectory has sufficient surrounding scene coverage. Let L be the candidate's horizontal path length. The required margins on both sides of the trajectory's bounding box are

$$m_x = \min(2.0,\, 0.5 + 0.5L), \qquad m_y = \min(1.0,\, 0.3 + 0.5L),$$

in meters. The x-margin requirement is waived when the asset contains an `obstacle_mesh` entry; the y-margin requirement remains active. This exception is based on asset metadata rather than an additional proof that an obstacle blocks the requested motion.

Acceptance, rejection, and budget exhaustion. A verifier pass is a local decision for the submitted trajectory and scene. Harness completion additionally requires the host's world-consistency check:

$$\texttt{complete} = \texttt{passed} \wedge \texttt{world_consistent}.$$

The agent's textual claim of success cannot override these checks. A rejected submission receives violation feedback and may be revised while interaction budget remains. The implementation allows one additional corrective turn when a submission is rejected at the nominal step limit. If no acceptable submission is obtained, a previously verified host baseline may be used when available; it remains subject to the world-consistency requirement and is explicitly recorded as a fallback. Otherwise, the run remains incomplete rather than accepting an unverified trajectory. Explicit abstention is also available when a reliable scene hypothesis cannot be established.

Constraints on scene edits and verification scope. The constrained semantic-edit interface ties each edit plan to a frozen scene index through its digest and requires edits to reference existing targets. It permits only the following class–action pairs:

```
SUPPORT           keep_support,
RIGID_OBSTACLE    keep_collision,
SOFT_TRAVERSABLE  disable_collision,
VISUAL_ONLY       remove_from_support,
UNKNOWN           no_change.
```

Each operation must include a nonempty reason and an evidence field. Removal from the support representation additionally requires explicit target labels. Original source meshes are preserved, and the edited asset is registered for final use only after its resulting rollout passes verification.

These restrictions provide edit provenance and constrain the available operations, but they do not independently establish the correctness of the agent's visual interpretation. In particular, the verifier does not certify that the supplied evidence justifies removing or disabling an obstacle. An incorrectly edited scene can therefore admit a trajectory that passes the numerical checks. World consistency provides an additional check across the evaluated hypotheses, but cannot rule out errors shared by those hypotheses. Accordingly, we interpret verification as conditional consistency with the supplied scene and carrier model, rather than a guarantee of physical correctness in the real environment.

Harness System Prompt — Task and Motion Intent

```
You are CarPhy's pose-correction coding agent.

Input:  historical poses, the current frame's 3D and 2D observations, carrier physical
parameters, and the raw pose trajectory P_raw over the requested window.  P_raw
specifies the horizontal trajectory of the user's WASD motion intent; do not rewrite
its t/x/y/heading.

Output:  a corrected pose trajectory covering the same requested window.  Correct the
z/roll/pitch/motion_state response induced by terrain contact.  Read the window duration
from observation['protocol']['future_horizon_s']; The current pose_future_v4 protocol uses
2.0 seconds.

Preserve the original motion intent.  You may correct height, pitch, roll, and whether
motion is blocked.  Do not change the intended forward direction, turning command, speed
intent, or path shape.

If terrain blocks the carrier, it should stop rather than detour.  If the carrier ascends
a slope, it should exhibit the corresponding pitch response rather than remain level.
Changing direction to avoid an obstacle is a deviation from the task.

Intent preservation is assessed through lateral deviation, not total planar distance.
Lateral deviation exceeding 0.35 meters violates the intent.  Along-path lag caused by
climbing or being blocked is a physically valid response.
```

Harness System Prompt — Execution and Interfaces

```
The main action is run_python:  execute your own Python code in the workspace.

Read TOOLS_API.txt in the workspace root for exact function signatures, field names,
and return keys.  You can also print T.describe().  Do not guess interface details from
memory.

Import the executable toolkit using:
from carphy.tools import terrain3d as T

These are importable library functions rather than black-box buttons.  You may use a
subset, change parameters, or implement an alternative computation.

Each execution step runs in a separate subprocess.  Only files written to disk persist.
Check that a file exists before reading it.

To expose a generated diagnostic image to the next interaction, print a separate line:
CARPHY_IMAGE: /absolute/path/to/figure.png

The host can attach declared images as feedback.  Images that are not declared are not
automatically visible to the agent.
```

Figure 11: **Harness system-prompt excerpts.** English translations of the task, intent-preservation, and execution instructions. The excerpts describe the distinction between high-level reasoning and executable numerical operations.

Harness User Message — Initial Task Context

```
The Task, Observation, and execution budget for this case are provided below as JSON:

{
  "task": "Correct the physical response of P_raw into a pose trajectory consistent with
the current 3D observations and carrier physics",
  "observation": {observation record},
  "baseline_rollout": {baseline result},
  "visual_safety_summary": {visual summary},
  "visual_clearance_obstacle_class": {obstacle class},
  "semantic_scene": {staged scene summary},
  "workspace_root": {workspace path},
  "workspace_files": {workspace file list},
  "budget": { "max_steps": {step budget}, "code_timeout_s": {timeout} }
}

initial_state.json and observation.json are available in the workspace root and can be
read through run_python.

Historical frames and the current frame are attached below. The last image is the current
frame, which is the starting point for correction. Decide whether to accept the default
candidate; use additional tools only when there is a concrete evidence conflict.
```

Conditional Harness Instruction — Verified Baseline

```
FAST PATH: The host has generated and verified default_physics_candidate. If there is no
explicit RGB or geometry conflict, immediately call accept_baseline. Do not read files,
repeat the rollout, or repeat the verifier.
```

Conditional Harness Instruction — Failed Asset Sanity

```
DEFAULT ASSET FAILED SUPPORT/PHYSICS SANITY. If visual evidence supports a canopy or
vegetation interpretation, call inspect_semantic_asset and submit a constrained semantic
plan. If reliable support remains unavailable after repair, call abstain.
```

Figure 12: **Initial task context and conditional execution guidance.** The structured context is populated separately for each case. The two conditional instructions shown here correspond to the host-baseline configuration and are appended according to the baseline verification result.

Harness Instruction — Vegetation and Collision Conflict

```
Vegetation has been detected, but the physical result reports contact_blocked. The host
has frozen semantic_scene.

Call apply_semantic_plan directly. The plan must reference semantic_scene.scene_digest and
a logical group in candidate_groups. Use SOFT_TRAVERSABLE/disable_collision and include
RGB or preview evidence.

Do not read the API, manually delete mesh geometry, or call run_python.
```

Harness System Prompt — Final Asset and Trajectory

```
asset_manifest_file must explicitly identify the final asset. If the trajectory file is
omitted, the host automatically runs rollout_intent using P_raw and verifies the result.
Do not manually reconstruct the fixed time axis or control parameters.

By default, submit the manifest and let the host generate the trajectory. If a custom
candidate must be submitted, use a file rather than embedding the trajectory inline.

The answer must cover the requested window. Its duration, time axis, and sampling density
must align with P_raw. Answering only part of the window does not constitute a complete
answer.

The trajectory must come from physical rollout. Do not hand-author a trajectory or
present the unchanged P_raw as a physical simulation result.
```

Figure 13: **Evidence-conditioned repair and final-submission instructions.** The vegetation instruction is a conditional task-message passage. The submission instructions are translated excerpts from the system prompt. These instructions guide the agent; the executable checks described in Appendix F determine the recorded outcome.