

Δ YNAMICS: Language-Based Representation for Inferring Rigid-Body Dynamics From Videos

Supplementary Material

A. Method Details

A.1. YAML-to-XML Conversion and Initialization

Our dataset uses YAML as the canonical representation of the full scene configuration. The YAML file contains both static attributes (e.g., object geometries, masses, friction coefficients, camera pose) and dynamic initial states (linear and angular velocities). MuJoCo, however, accepts model definitions only in its XML-based scene description format, which encodes static model properties but does not support specifying initial velocities.

To run each simulation, we therefore proceed in two steps. First, we convert the static components of the YAML configuration into a MuJoCo XML file, defining the bodies, inertial properties, joints, geoms, and camera. Second, we load this XML into MuJoCo, initialize the engine and then set all remaining dynamic quantities (e.g., initial linear and angular velocities) directly in the simulator state before simulation rollout.

A.2. Dataset Preparation

MuJoCo Rendering Details. Each simulation is rendered for one second at 30 FPS, with eight physics steps per frame. The output resolution is 480×320 pixels. We introduce randomized lighting conditions with varying shadow configurations and apply diverse object textures, including checkerboard and gradient patterns, to improve robustness to visual appearance. During simulation, we record RGB frames, segmentation masks, contact information, and full state histories, which are later used to generate structured visual reasoning annotations.

Optical flow estimation is often unreliable on smooth or textureless surfaces. To mitigate this, we fill the background and floor with natural scene images, which introduce sufficient texture and yield substantially higher-quality flow fields. Optical flow is computed between consecutive frames (30 FPS), producing 29 flow maps per video; we then sample every third frame to obtain the inputs used for model training.

Structured Visual Reasoning. To ensure consistent and interpretable spatial reasoning, we discretize continuous object positions and motions into categorical linguistic descriptors along three spatial axes. For example, positions along the x -axis are quantized into seven categories: far left ($x < -2$), moderately left ($-2 \leq x < -1$), slightly left ($-1 \leq x < -0.5$), near center ($-0.5 \leq x < 0.5$), slightly right ($0.5 \leq x < 1$), moderately right ($1 \leq x < 2$), and far

right ($x \geq 2$).

Segmentation maps are used to determine whether each object is initially visible and to record when it leaves the camera’s field of view. During simulation, we log collision and contact histories from the physics engine to capture fine-grained interaction events such as ground contact and inter-object collisions. We also analyze object state trajectories to identify when each object comes to rest, enabling precise annotation of temporal dynamics such as motion duration and stopping time. To increase linguistic diversity, the transformation script converts these auxiliary signals into textual descriptions using multiple paraphrased sentence templates while preserving structural consistency.

Example of Structured Visual Reasoning. An example of synthetic data is shown in Table 8. The model analyzes object properties, motion patterns, and physical interactions in natural language, which serves as an intermediate step to improve parameter estimation accuracy.

A.3. Details on In-Context Example Preparation

To guide the models effectively, we construct three in-context examples that (i) include all primitive shapes present in our dataset and (ii) cover the full range of physical parameters by selecting configurations at the minimum and maximum values of the training distribution. Each example consists of ten video raw RGB frames paired with its YAML-based scene configuration.

For CLEVRER [59], we first select three target example videos and manually annotate three corresponding scene configurations. The examples are chosen to (i) include all three primitive shapes and (ii) span the minimum and maximum values of our physical-parameter ranges, ensuring that the model does not extrapolate beyond the provided examples. To maintain fidelity, we iteratively refine each configuration annotation so that the resulting simulated motions and object trajectories closely match those in the target videos.

A.4. Details on Preference Optimization

Preliminary: Preference Rank Optimization. Following the Bradley–Terry formulation [12], a reward model (RM) estimates pairwise preferences by contrasting two responses y^1 and y^2 for a given input x . Preference Ranking Optimization (PRO) [45] extends this idea by directly fine-tuning the policy π_θ , treating it as both the RM and the

policy network. The PRO loss is defined as:

$$\mathcal{L}_{\text{PRO}} = -\log \frac{e^{r_{\pi}(x, y^1)}}{e^{r_{\pi}(x, y^1)} + e^{r_{\pi}(x, y^2)}}, \quad (3)$$

where $r_{\pi}(x, y)$ denotes the implicit reward $r_{\pi_{\text{PRO}}}$ for a given input x and candidate response y^k . It is defined as the average token-level log-likelihood:

$$r_{\pi_{\text{PRO}}}(x, y^k) = \frac{1}{|y^k|} \sum_{t=1}^{|y^k|} \log P_{\pi}(y_t^k | x, y_{<t}^k). \quad (4)$$

Intuitively, $r_{\pi_{\text{PRO}}}(x, y^k)$ measures the normalized sequence log-likelihood (i.e., the mean per-token log-probability) and serves as a scalar proxy for how confidently the model assigns probability mass to the response y^k given x .

Soft Preference Weighting. However, Eq. 3 assumes a *one-hot* preference—one response is strictly preferred ($y^1 \succ y^2$) while the other is not. In our setting, this binary assumption is overly rigid: two simulated rollouts may each excel in distinct aspects (e.g., one accurately reproduces geometry while the other better matches damping or velocity). To capture such nuanced trade-offs, we introduce a *soft preference weighting* that transforms the simulator-derived rewards into continuous targets:

$$\tilde{r}(y^i) = \frac{e^{s(y^i)/\tau}}{\sum_{j \in \{1,2\}} e^{s(y^j)/\tau}},$$

where τ is a temperature and $s(\cdot)$ denotes the simulation-derived score function, i.e., segmentation IoU. Thus, the optimization objective then becomes:

$$\mathcal{L}_{\text{soft-PRO}} = - \sum_{i \in \{1,2\}} \tilde{r}(y^i) \log \frac{e^{r_{\pi}(x, y^i)}}{e^{r_{\pi}(x, y^1)} + e^{r_{\pi}(x, y^2)}} \quad (5)$$

which can be interpreted as a *reward-weighted cross-entropy*—analogous to replacing a binary BCE loss with a soft-label CE loss. This formulation better accommodates partially correct rollouts and encourages the policy to allocate probability mass in proportion to their normalized simulator rewards, leading to smoother and more stable test-time adaptation. This approach bears similarity with soft-preference concept in previous work [22, 44]

B. More Qualitative Results

For direct comparison of physical parameters, we show initial state error on synthetic eval set in Table 7.

C. Cross-Engine Generalization

C.1. Dataset Preparation

We subsample 400 one-second clips from the validation split of CLEVRER [59]. Object segmentation masks are

Table 7. Initial States Estimation (MAE ↓).

	Position	Velocity
InternVL3 / Qwen2.5 / Claude4	3.05 / 3.39 / 2.55	4.38 / 3.32 / 3.85
Ours / Ours+Analysis	2.20 / 2.16	3.14 / 2.94

obtained from the official release. We first clean the masks by checking whether each object’s segmentation changes within the subsampled clip—objects with static masks are treated as non-moving and removed from motion evaluation. This information is then used to refine the corresponding optical flow maps, ensuring that static regions are not misinterpreted as motion.

C.2. Baseline Evaluation Outcomes

The full quantitative evaluations, including optical flows, are shown in Table 9. Qualitative results are shown in Figure 6. These are the results without using CMA-ES.

Test-Time Optimization. We therefore explore preference optimization to learn from unlabeled videos as detailed previously in Section 4.2. Specifically, we use 1000 cases from CLEVRER training data, generate 32 sampled predictions per case, render rollouts with MuJoCo, and compute segmentation IoU to construct pairwise preferences. We then draw three paired data per case to construct the preference learning dataset and finetune our reasoning model. As shown in Table 10, preference optimization yields consistent improvements—modest IoU gains (+1%) and a notable reduction in first-frame EPE (2.22 to 1.85) under the Best@32 metric. This demonstrates that preference optimization provides a practical route to label-free adaptation, enabling models to refine directly through simulation feedback. However, the marginal gain in performance is less than CMA-ES.

D. Real-World Benchmark and Qualitative Results

D.1. Real-World Dataset Statistics.

Our real-world evaluation set comprises a total of 235 videos capturing diverse rigid-body motion scenarios. Among them, 155 videos were recorded with an iPhone 13 and 80 videos with a Canon camera. The iPhone videos were captured at 210 FPS or 240 FPS, while the Canon videos were recorded at 50 FPS. To create variations in temporal resolution, we downsampled the original recordings by uniformly sampling frames to obtain videos at 25, 30, 50, 60, and 70 FPS. The resulting frame-rate distribution is: 30 FPS (76 videos), 70 FPS (46), 25 FPS (40), 50 FPS (40), and 60 FPS (33). Each scene contains between one and five objects, with 139 single-object, 69 two-object, 17 three-object, 6 four-object, and 4 five-object configurations. The objects span a wide variety of everyday items, includ-

Table 8. **Example Synthetic Training Data Instance.** We illustrate the target output format for both the vanilla and reasoning-augmented variants of our model. The blue box shows the structured YAML configuration. The red box shows the corresponding natural-language reasoning describing object motions and interactions. In the vanilla setting, the model is trained to generate only the configuration text wrapped within the `<answer>` tag. In the reasoning-enhanced setting, the model first outputs the reasoning text enclosed by `<think>` tags, followed by the configuration text within `<answer>` tags.

configuration =

```
- type: box
  name: box_0
  size: [1.0, 0.5, 0.4]
  state:
    angular_velocity: [0, 0, 0]
    linear_velocity: [4.3, 2.5, 0.0]
    orientation: [0.97, 0.0, 0.0, 0.23]
    position: [-5.0, -0.3, 0.4]
  physics:
    friction: [1.1, 0.3]
    mass: 1.0
    damping: -4
- type: cylinder
  name: cylinder_0
  radius: 0.3
  height: 0.5
  state:
    angular_velocity: [0, 0, 0]
    linear_velocity: [-0.1, -0.5, 0.0]
    orientation: [0.69, 0.69, 0.15, 0.15]
    position: [-0.5, 0.8, 0.3]
  physics:
    friction: [0.5, 0.3]
    mass: 1.0
    damping: -4
- type: camera
  fovy: 45
  orientation: 45
  position: [0, -2, 3.5]
- type: gravity
  gravity: [0, 0, -7.0]
```

reasoning =

This physics simulation showcases objects interacting under realistic physics.

- Box_0: A cuboid with dimensions 1.0 x 0.5 x 0.4 m, mass 1.0 kg, positioned leftmost and in the foreground, close to the surface. It moves at 4.97 m/s (rightward and forward), stops at t=0.4s, and stays grounded from 0.00-0.90s. Collides with cylinder_0 at t=0.1s.
- Cylinder_0: A solid column (radius=0.3m, height=0.5m, mass=1.0kg) located at the X-axis origin, near the camera and base plane. Moves at 0.51 m/s, momentum 2.79 m/s until end, collides with box_0 at t=0.1s.
- Observation Data: Visible entities: box_0, cylinder_0. Both visible in 10/10 frames.
- Dynamic Interactions: Contact event between cylinder_0 and box_0 at t=0.1s.

Target Sequence (Vanilla Δ YNAMICS):

`<answer>` configuration `</answer>`

Target Sequence (Δ YNAMICS + Motion Reasoning):

`<think>` reasoning `</think>` \n\n `<answer>` configuration `</answer>`

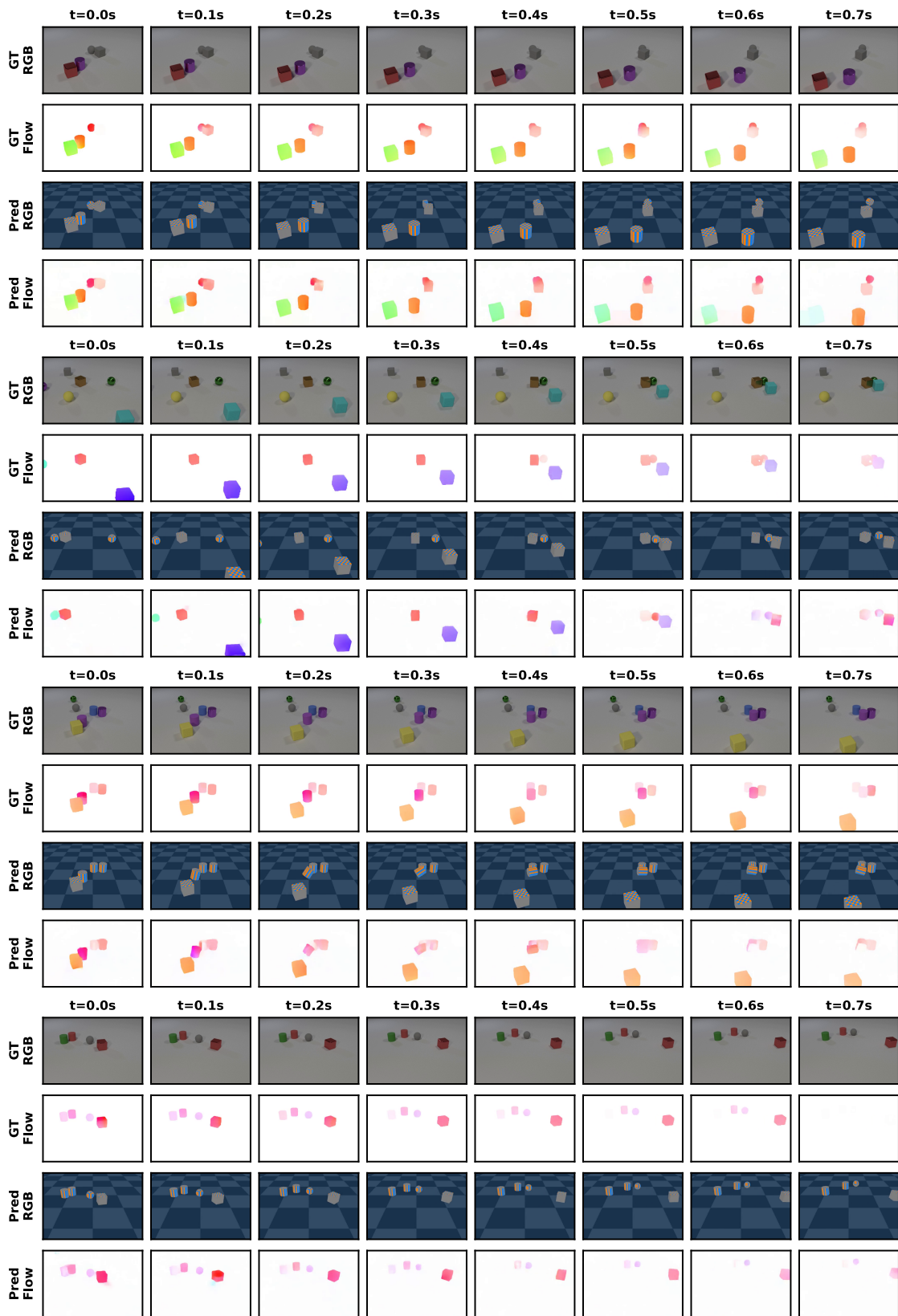


Figure 6. CLEVRER Dataset Results.

Table 9. **Generalization Across Simulation Engines.** Evaluating transfer from MuJoCo (training) to Blender (CLEVRER [59]) demonstrates that Δ YNAMICS maintains its performance in a zero-shot setting. Despite domain shifts in rendering and dynamics, incorporating structured motion description consistently improves segmentation IoU and optical flow EPE. Note that some off-the-shelf models achieve low optical flow EPE by generating fewer objects than present in the scene, which artificially reduces motion and lowers EPE compared to models attempting more complete and realistic physics simulation. **Best** and runner-up results are highlighted.

		Segmentation Map IoU (\uparrow)		Optical Flow EPE (\downarrow)	
		First Frame	Full Sequence	First Frame	Full Sequence
<i>VLM Models</i>					
InternVL3-8B	RGB	0.01	0.02	7.12	6.10
Qwen2.5-VL-7B	RGB	0.01	0.01	9.22	7.41
Claude-4-Sonnet	RGB	0.03	0.04	6.34	5.43*
<i>Ours</i>					
Δ YNAMICS	RGB	0.43	0.19	3.68	7.13
Δ YNAMICS	Opt. Flow	0.63	0.24	3.51	6.85
+ Motion Reasoning	Opt. Flow	0.67	0.30	2.79	5.94

Table 10. **Evaluation of Test-Time Sampling and Preference Optimization on CLEVRER.** We compare the base, reasoning-enhanced, and preference-optimized Δ YNAMICS under greedy decoding and best-of- N sampling. For each case, 32 samples are generated with a temperature of 0.1; *Best@1* denotes the average, while *Best@32* reports the best. **Best** and runner-up results are highlighted.

	Segmentation Map IoU (\uparrow)						Optical Flow EPE (\downarrow)					
	Greedy	First Frame		Greedy	Full Sequence		Greedy	First Frame		Greedy	Full Sequence	
		Best@1	Best@32		Best@1	Best@32		Best@1	Best@32		Best@1	Best@32
Δ YNAMICS	0.63	0.63	0.67	0.24	0.24	0.28	3.66	3.65	2.92	6.91	6.86	6.21
+ Motion Reasoning (MR)	0.67	0.68	0.76	0.30	0.30	0.38	2.92	2.93	2.22	5.94	5.95	5.17
+ MR + PRO	0.68	<u>0.69</u>	0.77	0.31	0.31	<u>0.39</u>	2.90	2.94	<u>1.85</u>	5.78	5.81	<u>4.78</u>
+ MR + CMA-ES	0.62	-	-	0.66	-	-	0.13	-	-	0.11	-	-

ing shoebox, drug spray, Pringles can, baseball, tennis ball, tissue box, soccer ball, basketball, pool ball, massage roller, gel container, aerosol can, handcrafted vehicle with wheels, apple, tumbler, soda can, whiteboard eraser, napkin roll, insulation pad, box, banana, and plum. Among all videos, 86 exhibit object collisions.

We also conduct a pilot human study: two annotators each estimated parameters for three real-world videos with iterative refinement (25 minutes per video). Humans achieved mean IoU of 0.44 and EPE of 1.38, versus Δ YNAMICS’s 0.61 and 0.71, respectively.

D.2. Qualitative Results.

In Figure 8 we show more real-world examples; in Figure 9 we show the results of irregular-shaped objects such as apples, wooden aircraft, and soda cans. We also present failure cases in Figure 10, where the failure modes comes from irregular shapes that are not seen during training or that results in unexpected bouncing trajectory. In some other cases, the model can predict wrong primitive shapes.

D.3. Non-object-centric scenes:

We show in-the-wild vehicle dynamics from DEXRAC (Figure 7), where Δ YNAMICS reconstructs vehicle motions using primitive geometry.

D.4. Future Work

While our results are promising in capturing rigid-body motions using a language-centric, VLM-based approach, we



Figure 7. Δ YNAMICS reconstructs vehicle *dynamics* in non-object-centric, in-the-wild scenes using primitive geometry.

identify several directions for future work: (1) incorporating 3D shape tokens [43] to move beyond primitive shapes, (2) extending to articulated objects [33] and sloped environments to cover more types of rigid-body motion, and (3) adopting more powerful engines such as Genesis [62] to model deformable objects and motions.

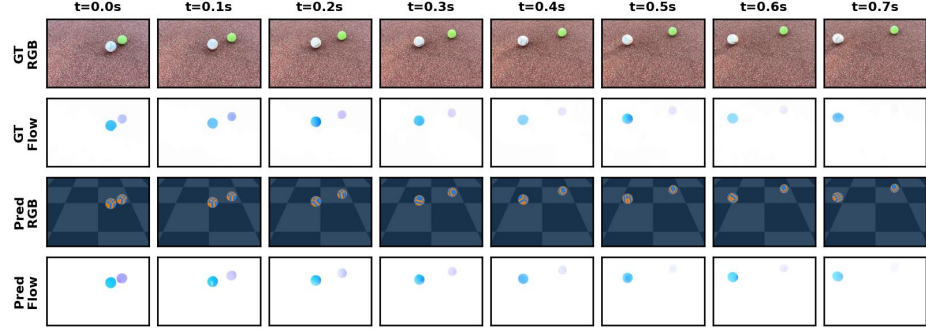
E. Physically Plausible Editing

Our framework naturally supports physically consistent editing of object dynamics and scene parameters. Because Δ YNAMICS represents each scene using an explicit and interpretable YAML configuration, user-provided editing instructions can be translated directly into modified physical parameters. This enables a closed-loop editing system that integrates a physics engine, a language model, and a video synthesis model to generate physically plausible edited videos.

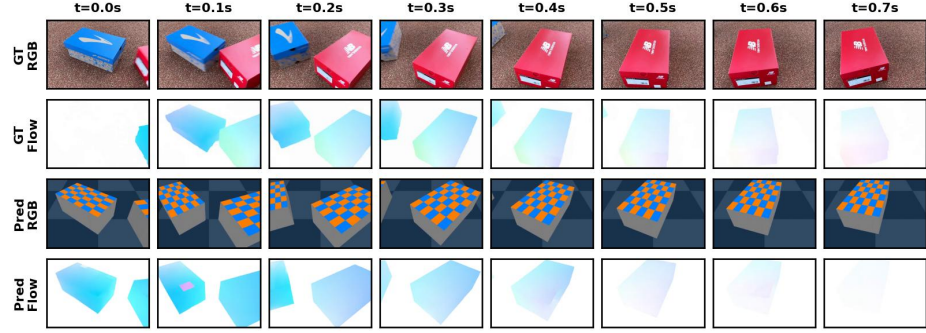
E.1. Editing Pipeline Overview

Figure 11 illustrates our four-stage editing pipeline:

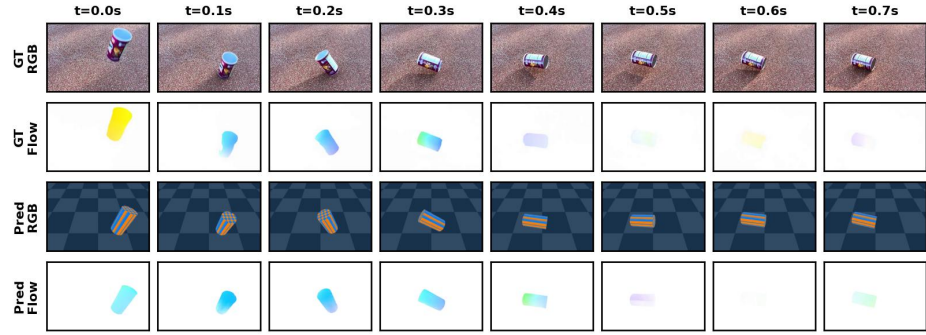
- **Configuration Extraction with Δ YNAMICS.** Given an



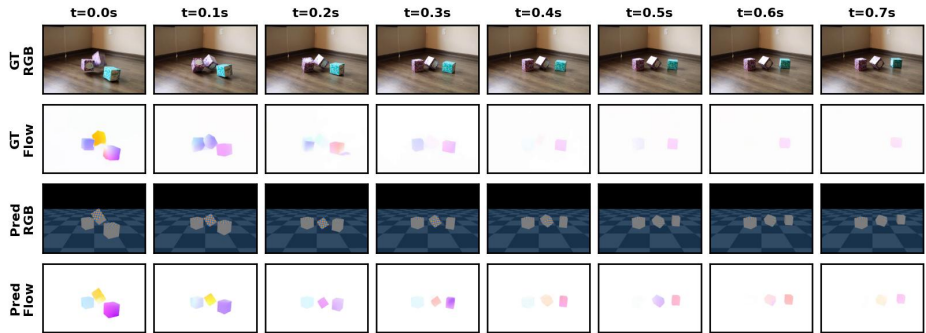
(a) A baseball and a tennis ball moving in different directions.



(b) A red shoe box hits a blue shoe box.



(c) A cylindrical container bouncing on the ground.



(d) Three tissue boxes dropping on the floor.

Figure 8. **Rigid-Body Motion Estimation on Our Real-World Dataset.** Δ YNAMICS reconstructs physically plausible trajectories from real-world videos of rigid-body motion, capturing object interactions, material properties, and dynamics across diverse conditions.

input video, we first infer its underlying physical configuration using Δ YNAMICS. The model outputs a complete YAML file specifying object geometries, initial poses, ve-

locities, masses, gravity, friction, and other physical attributes. This YAML file serves as an editable and fully interpretable *source code* for the scene.

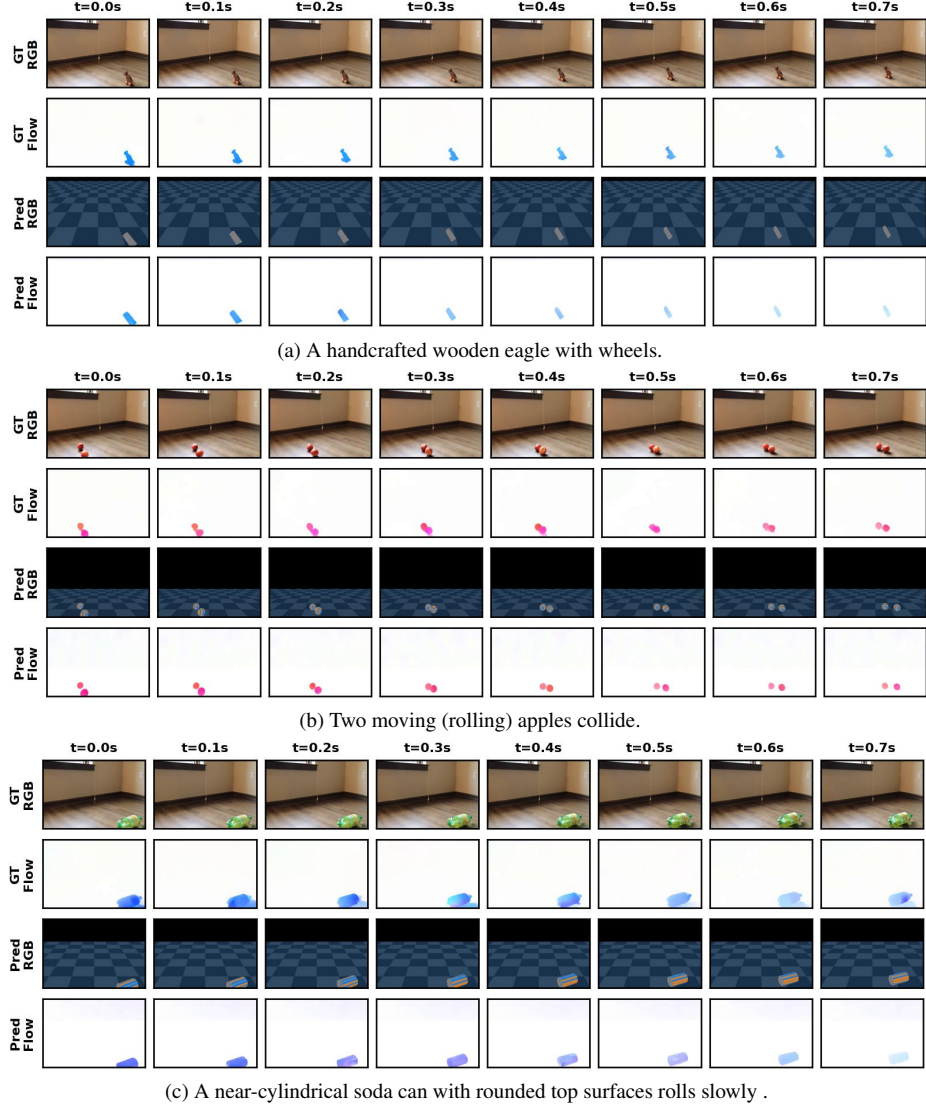


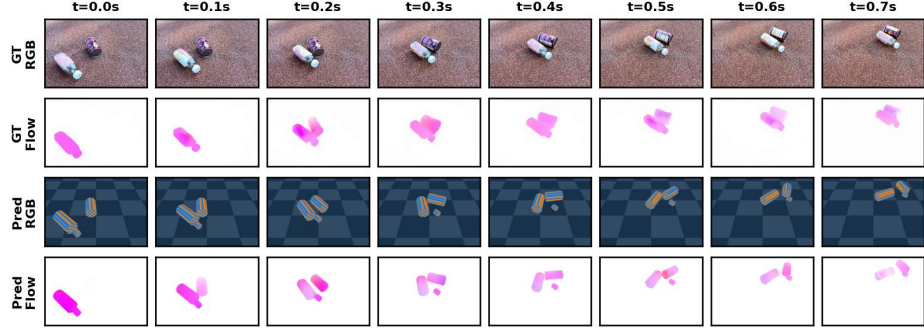
Figure 9. Rigid-Body Motion Estimation on Our Real-World Dataset, Focusing on Irregularly Shaped Objects.

- **Language-Guided Configuration Editing.** To incorporate a user instruction (e.g., “reduce the x-velocity by 80%” or “decrease gravity by 50%”), we prompt Claude-3-Haiku with (i) the full YAML configuration predicted by Δ DYNAMICS, and (ii) the editing instruction. Claude outputs a revised YAML file with localized and semantically appropriate modifications (e.g., updating only the fields for initial velocity, gravity, or angular velocity). Because YAML is structured, line-addressable, and semantically meaningful, the language model reliably edits only the intended parameters while leaving the rest of the configuration intact. This enables precise and controllable manipulation of physical properties that would otherwise be entangled in a latent space.
- **Simulation and Flow Generation.** The edited configu-

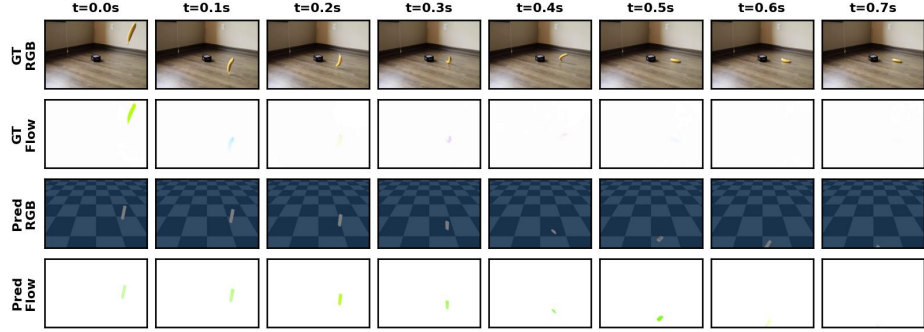
ration is executed in MuJoCo, producing a modified motion trajectory consistent with the user’s edit. We then compute dense optical flow using RAFT [49], yielding a physically grounded flow field that encodes the new dynamics.

- **Video Synthesis via Go-With-The-Flow.** Following Burgert et al. [13], the Go-With-The-Flow model synthesizes the edited video by warping noise according to the edited optical flow, conditioned on the first RGB frame of the original video. This preserves the appearance of the scene while enforcing the motion cues determined by the edited flow.

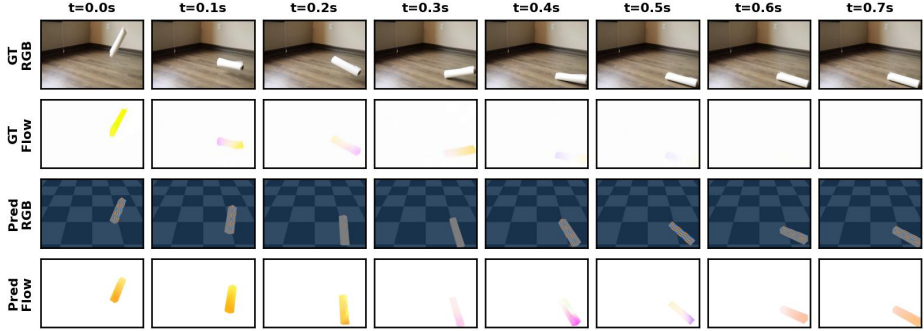
Empirical results are shown in Fig. 12, which demonstrates edits to both box dynamics (e.g., reducing x- or y-velocities) and ball dynamics (e.g., modifying velocity di-



(a) Due to the bottle's irregular shape, our model approximates it with two cylinders and focuses on faithfully reconstructing the motion in the first frame ($t = 0.0s$).



(b) A banana falls to the floor, bounces once, and then comes to rest. Its irregular shape produces an unexpected trajectory, making it difficult to capture object positions and camera poses accurately.



(c) In this case, the model predicts an incorrect primitive shape and an imprecise camera angle.

Figure 10. **Rigid-Body Motion Estimation on Our Real-World Dataset, Focusing on Failure Cases.**

rection or reducing gravity). The pipeline produces physically correct motion and high-quality visual results in most cases.

A primary limitation arises from **appearance preservation under complex motion**. Although Go-With-The-Flow accurately follows the edited optical flow, it sometimes struggles with fine-grained dynamic appearance details, e.g., maintaining the texture fidelity of a spinning or rolling basketball. As this remains an open challenge, future work could explore fine-tuning the generative model conditioning on edited optical flow fields to achieve better physically plausible video editing results.

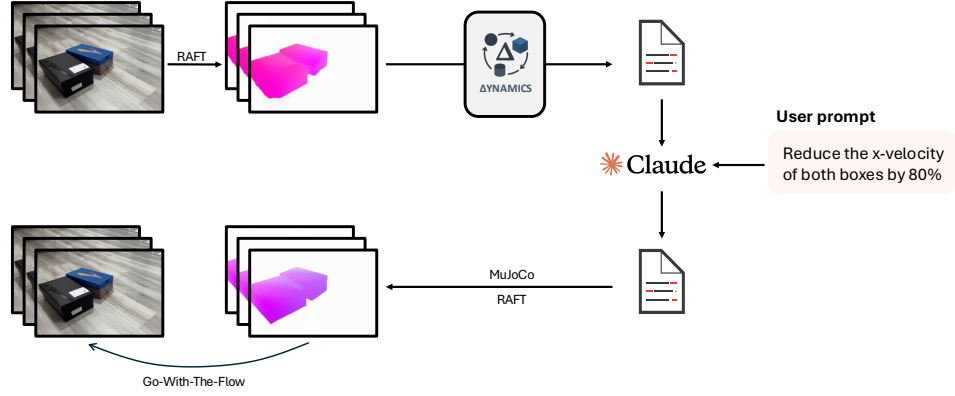
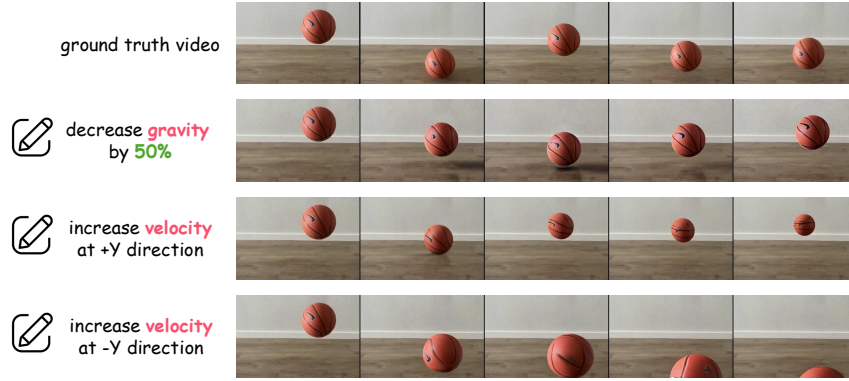
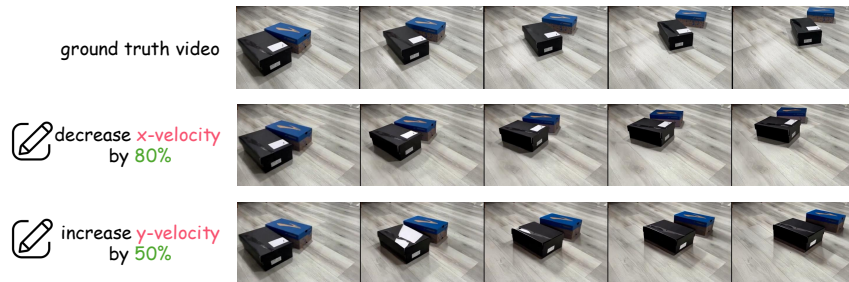


Figure 11. **Physics Editing Pipeline.** Given a user-provided editing instruction (e.g., “reduce the x-velocity by 80%”), we first infer the original scene configuration using Δ YNAMICS. Next, we prompt a large language model (Claude) with both the predicted configuration and the user instruction to generate a revised, physically consistent configuration. The edited configuration is then executed in MuJoCo to produce a modified motion trajectory, from which we compute RAFT optical flow. Finally, we feed the edited flow fields to Go-With-The-Flow [13] to synthesize the edited video.



(a) **Ball Motion Editing.** Example edits include decreasing gravity by 50% and modifying velocity magnitude or direction. These edits are reflected in the regenerated simulated trajectories and corresponding edited videos. Here, positive y motion indicates movement *away from the camera* (deeper into the scene), while negative y motion brings the object *closer toward the viewer*.



(b) **Box Collision Editing.** Here we apply velocity reductions along specified axes (e.g., reducing x-velocity by 80% or y-velocity by 50%). The resulting interactions follow physically plausible adjustments in motion and contact behavior.

Figure 12. CLEVRER Dataset Results.