QUANTIPHY: A Quantitative Benchmark Evaluating Physical Reasoning Abilities of Vision-Language Models Supplementary Material

Contents

A. More Studies and Results

A.1. Additional Case Studies

To better understand how Vision-Language Models (VLMs) solve kinematic inference tasks beyond aggregate scores, we conduct a qualitative case study on the top-performing model, ChatGPT-5.1, using its `Thinking` mode in the standard user-facing interface. For each selected video–text pair, we repeatedly query the model and inspect the tool-augmented chain-of-thought until we obtain representative traces that are syntactically well-formed and numerically valid.[5] We then analyze both the final numerical answers and the intermediate reasoning steps.[6]

Overall, we observe a sharp contrast between successful instances, where the model follows a textbook-like "measure pixels → apply prior → compute target" pipeline, and failure modes, where it largely ignores the video and the provided prior, and instead falls back to pre-trained world knowledge or generic heuristics. Below we discuss four representative cases.

Case 1: Faithful pixel–prior reasoning. Figure 6 shows a 2D scene with a yellow car moving laterally. The model is asked two questions: (i) given that the car's length is 5.67 m, what is its speed at 2.0 s; and (ii) what is the car's width in meters. In this instance, ChatGPT-5.1's chain-of-thought closely matches the intended reasoning procedure. The model first identifies the relevant frames around $t = 2.0$ s, uses OpenCV-style tools to obtain bounding boxes, and explicitly treats the longer side of the box (135 px) as the car's length in pixel space. It then calibrates a pixel-to-meter scale from the given length prior (5.67 m), and computes the width as

$$\text{width} \approx \frac{58}{135} \times 5.67\text{m} \approx 2.44\text{m}\,,$$

which is close to the ground truth width and achieves high relative accuracy. Here the model behaves as an input-faithful visual measurer: it grounds both the prior and the kinematic target in pixel space and performs the correct proportional reasoning. When this pipeline is followed, the resulting numerical answers are often near the ground truth.

Case 2: Counterfactual prior breaks faithfulness. In the second case (Figure 7), we reuse the same video but multiply the car-length prior by a counterfactual factor of 1000, changing the input to "length of the yellow car = 5670 m." The task is again to infer speed at 2.0 s and the car's width. In its `Thinking` trace, the model explicitly notes that "5670 m" is an implausible car length and expresses confusion. Crucially, instead of continuing to rely on pixel measurements and the (counterfactual) prior, it effectively abandons the video and the numeric input. For the width question, it switches to a generic heuristic, assuming a "typical car's width-to-length ratio" and hallucinating a plausible-looking width independent of the actual scene. The final width prediction happens to have high relative accuracy (close to 0.9), but this success is not input-faithful; it arises from pre-trained knowledge about cars rather than from the specific video or the given prior. This case highlights a key risk that purely outcome-based met-

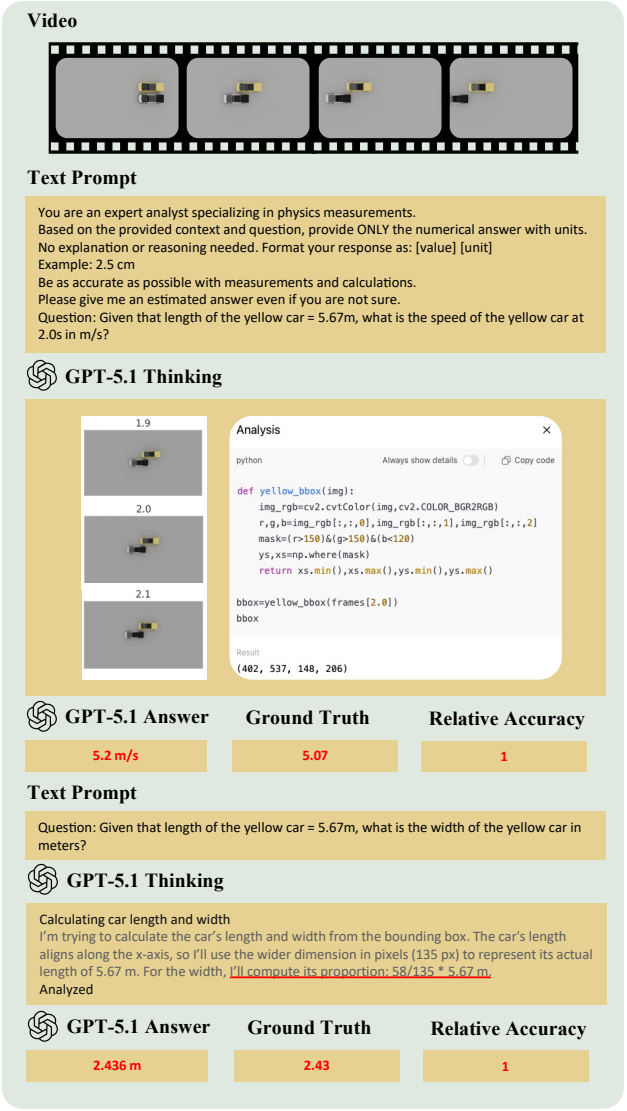


Figure 6. **Case 1: Faithful pixel–prior reasoning.**

[5] We prompt the model multiple times and observe substantial variability: on the same instance, some runs produce accurate and well-structured reasoning, while others fail to parse the question or ignore key inputs. We therefore collect several responses per instance and manually select representative traces that are syntactically coherent and numerically valid for detailed analysis. A similar instability is also present in our API-based evaluation, even with temperature fixed to 0; in the main benchmark, we address this by running multiple trials and recording a failure rate.

[6] We emphasize that this analysis is diagnostic rather than evaluative: we study one specific model's internal behavior to illustrate broader patterns of (un)faithful quantitative reasoning.

Figure 7. **Case 2: Counterfactual prior breaks faithfulness.**

rics can judge an answer as "good," while the underlying reasoning ignores the provided evidence.

Case 3: Video ablation reveals reliance on priors. Case 3 (Figure 8) uses a video-ablation setting. The model receives only the text prompt (including "length of the yellow car = 5.67 m"), without access to the video. When asked for the car's speed at 2.0 s, ChatGPT-5.1 produces an answer (12 m/s) that is far from the ground truth, confirming that motion estimation is difficult without visual evidence. However, when asked for the car's width (still without video), the model outputs a numerically reasonable value with relatively high accuracy (relative accuracy ≈ 0.7). Since no pixel information is available in this ablated setting, this behavior can only be explained by the model's internal prior over typical car dimensions. Combined with Case 1 and 2, this suggests a pattern that even when video is available, much of the "size" inference can be driven by pre-trained world knowledge rather than by explicit pixel measurements.

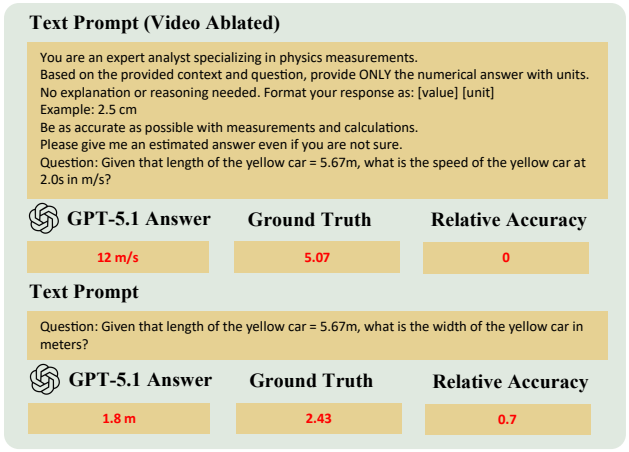


Figure 8. **Case 3: Video ablation reveals reliance on priors.**

Case 4: Strong gravitational prior overrides counterfactual physics. The fourth case (Figure 9) involves a Blender-simulated basketball scene that visually resembles a realistic indoor court, but with counterfactual physics. The ball's acceleration is time-varying and close to 1 m/s^2, rather than standard gravity. The model is asked for the ball's acceleration at 0.5 s and its speed at 1.5 s, given the ball's diameter as a prior. Here, ChatGPT-5.1 completely ignores both the video and the non-standard trajectory implied by the simulation. It directly outputs the canonical gravitational acceleration 9.8 m/s^2, and for speed simply multiplies g by time (i.e., $9.8 \times 1.5 = 14.7$ m/s), leading to relative accuracy equals to 0 on both queries. No pixel measurements or scale computations appear in the `Thinking` trace. This illustrates how strong pre-trained physical priors (e.g., "objects fall with acceleration g") can dominate the model's behavior, even when they contradict the actual visual input and the provided prior.

Discussion. These four cases collectively sharpen our main quantitative findings.

- When everything works, ChatGPT-5.1 can execute an impressive, tool-augmented pipeline that does read pixel trajectories, apply the physical prior, and compute accurate kinematic quantities.
- However, this behavior is fragile. As soon as the prior becomes counterfactual, the video is ignored, or the underlying physics departs from familiar regimes, the model quickly reverts to pre-trained world knowledge or rough heuristics, often ignoring the provided inputs.
- High numerical accuracy does not guarantee input-faithful reasoning. Specifically, Case 2 demonstrates that a model can get the "right" answer for the wrong reasons, while Cases 3 & 4 show that it can stick to canonical physical constants even when the scene violates them.

These observations suggest that improving VLMs' quantitative physical reasoning will require not only better ag-

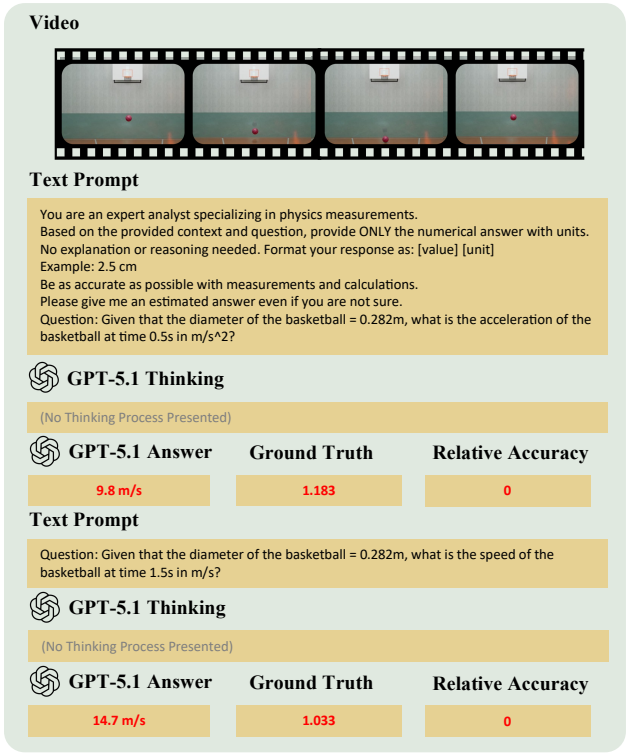


Figure 9. **Case 4: Strong gravitational prior overrides counterfactual physics.**

gregate performance, but also mechanisms that encourage faithful use of visual evidence and explicit numerical priors, rather than letting powerful, but sometimes misleading, pre-trained world knowledge dominate the inference process.

A.2. Metric Design Justification

In QUANTIPHY, we adopt Mean Relative Accuracy (MRA) as the primary evaluation metric for quantitative physical inference tasks with continuous outputs. Each prediction $\hat{y}$ is compared to the ground truth y using its relative error $|\hat{y} - y|/y$, and awarded partial credit if the error falls below confidence thresholds $\theta \in \{0.5, 0.55, \ldots, 0.95\}$. The final MRA score is the average of binary accuracies across these thresholds

$$\text{MRA} = \frac{1}{10} \sum_{\theta \in C} \mathbb{1} \left(\frac{|\hat{y} - y|}{|y|} < 1 - \theta \right), \qquad (1)$$

where $C = \{0.5, 0.55, \ldots, 0.95\}$ is the set of confidence thresholds.

While one could alternatively compute continuous relative error (e.g., mean relative error or mean absolute percentage error), we prefer MRA for several practical and conceptual reasons.

Discrete but calibrated. MRA discretizes accuracy into a finite set of thresholds, offering interpretable feedback on how often the model is "close enough" under increasing demands. Rather than penalizing deviations proportionally (which can be dominated by outliers), MRA provides a graded scale of correctness, similar in spirit to the mAP@IoU[7] metrics in object detection.

Robust to ambiguity and noise. Many video-based physical inferences involve semantic or visual uncertainty. For example, estimating a person's height may vary depending on whether hair or shoes are included; estimating a cup's diameter may depend on whether the inner or outer rim is used. MRA tolerates such ambiguity by granting full credit when answers fall within a reasonable margin.

Moreover, measurement noise is often unavoidable:

- **Temporal aliasing:** limited frame rates restrict temporal resolution for computing velocities and accelerations;
- **Motion blur:** fast-moving objects introduce visual uncertainty during measurement;
- **Imprecise priors:** even real-world scale references (e.g., credit card dimensions) may not be perfectly visible or aligned.

MRA accommodates these natural imperfections more flexibly than a regression-style loss.

Supported by precedent. The MRA metric was introduced by Yang et al. [53] in VSI-Bench for evaluating numerical answers in visual–spatial reasoning tasks. Some follow-up work [12, 39] adopted the same evaluation to benchmark multimodal models in physical and spatial settings. These works motivate MRA as a stable and discriminative way to capture proximity between predicted and ground-truth values, especially when scale varies across examples. Our use of MRA continues this design choice, ensuring comparability while improving robustness.

In sum, MRA balances informativeness, robustness, and interpretability. It is well-suited for evaluating VLMs on physically grounded, numerically sensitive tasks like those in QUANTIPHY, where small deviations are acceptable, but large errors are unacceptable regardless of scale.

A.3. Model MRA Distribution

Figure 10 visualizes the MRA distributions for each VLM. Each subplot shows a density curve with mean (gray solid) and median (blue dashed) lines, along with summary statistics (mean, std, median, Q25, Q75) displayed to the right.

The top-performing models, including ChatGPT-5.1, Gemini-2.5 Pro, Gemini-2.5 Flash, Qwen3-VL-instruct-32B, and Grok-4.1-Fast-Reasoning, show notably higher densities around moderate to high MRA values above 0.5, with means and medians clustering around 0.4–0.6. Their smoother, more concentrated curves indicate both higher accuracy and more consistent performance across the evaluation set.

[7] mAP@IoU: Mean Average Precision (mAP) calculated at specific Intersection over Union (IoU) thresholds.

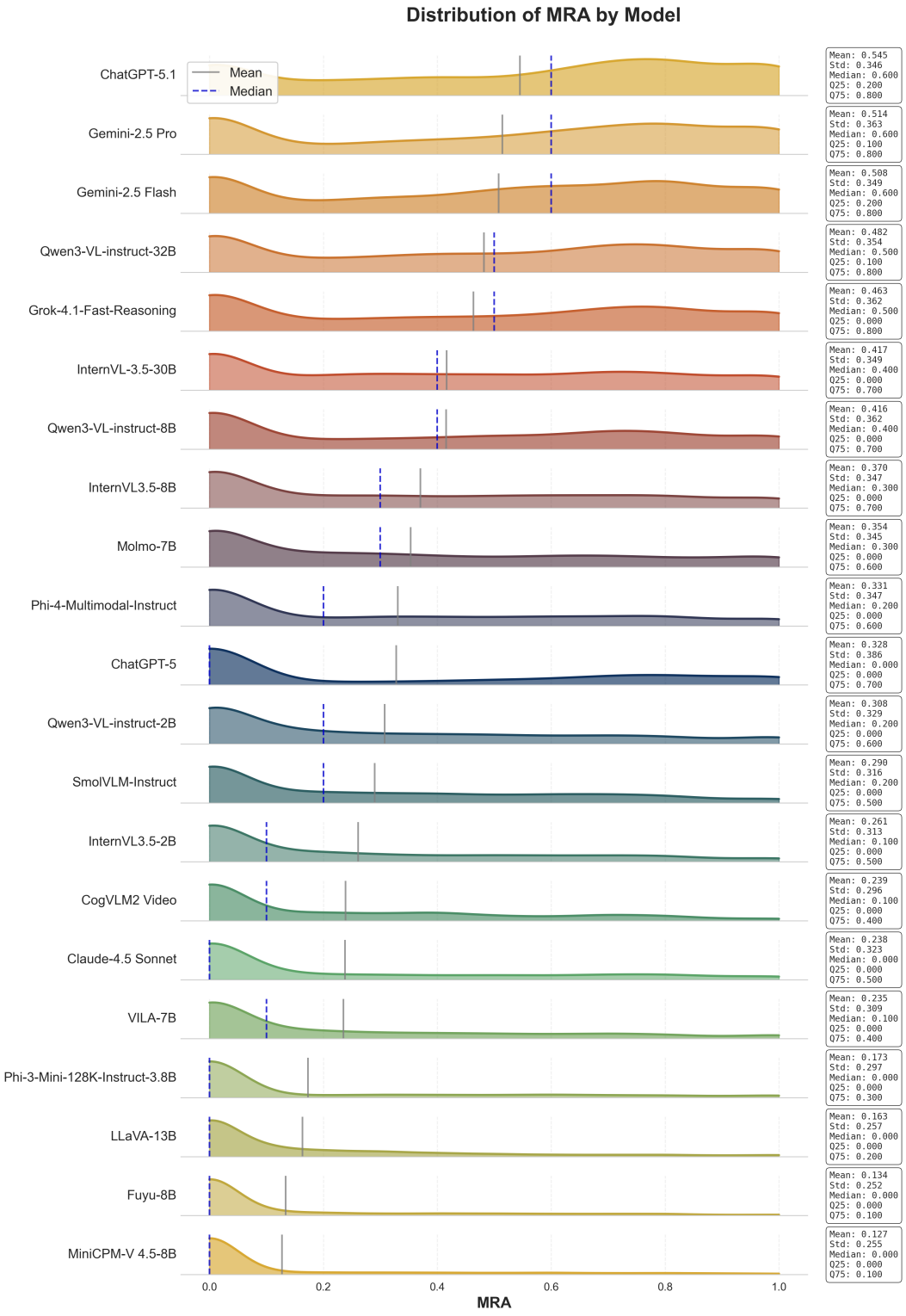


Figure 10. **Distribution of MRA by model.** One caveat to note is that the Avg. MRA in Table 1 reflects the mean MRA across inference task categories for each model (i.e., the average MRA of `2D-Static`, `2D-Dynamic`, `3D-Static`, and `3D-Dynamic`). In contrast, the mean in this distribution plot represents the average MRA at the individual-question level for each model.

A second tier of models, including InternVL-3.5-30B, Qwen3-VL-instruct-8B, Molmo-7B, and ChatGPT-5, exhibits slightly lower means and medians, generally around 0.3–0.4. Their broader distributions with heavier tails indicate greater variability that these models produce some strong outputs but also more low-scoring cases. The alignment of medians and means suggests errors are not extremely skewed.

Mid-tier models such as Qwen-3-VL-instruct-2B, Phi-4-Multimodal-Instruct, SmolVLM-Instruct, CogVLM2 Video, Claude-4.5 Sonnet, and VILA-7B show means of 0.2–0.3. Their distributions are heavily weighted toward low MRA values with thin right tails, indicating that while they occasionally achieve moderate scores, they rarely reach the performance levels of top-tier systems.

The weakest-performing models, for example, Phi-3-Mini-128K-Instruct-3.8B, LLaVA-13B, Fuyu-8B, and MiniCPM-V 4.5-8B, show distributions highly concentrated near zero. With means below 0.15 and medians around 0.0, these models fail to produce meaningful MRA performance in most cases.

Notably, many distributions have substantial mass centered around zero. An MRA of zero indicates either the model output zero as an answer or failed to produce proper numerical output. We also observed that some model APIs are unstable and produce errors over time, potentially due to API server error, running environment, internet traffic, batch size variations, etc. We abstract away from these in this paper and simply treat these failed cases as observations with MRA equal to zero.

Overall, modern frontier proprietary models cluster around substantially higher and more consistent MRA values, mid-tier models show moderate capability with noticeable variability, and smaller or older models yield predominantly low scores. The density patterns and comparative statistics reveal a clear performance gap between state-of-the-art systems and lightweight or earlier-generation models.

B. Dataset Construction Guidelines

B.1. General Principles

Our data collection and curation follow several general principles to ensure that QUANTIPHY is ethically sourced, physically well-defined, and suitable for quantitative evaluation.

Copyright and ethics. We carefully avoid copyright and ethical issues throughout all stages of data collection and processing. All videos, 3D assets, and simulation resources are either open-source, licensed for research use, or explicitly verified to pose no known copyright conflicts before inclusion. When raw videos contain personally identifiable information (e.g., human faces, license plates), we apply blurring or masking to anonymize the content.

Video selection criteria. To make quantitative kinematic inference well-posed and to reduce confounding factors, we enforce the following constraints on all collected videos.

- **Static camera in world coordinates.** The camera remains fixed in the world frame during each clip. This avoids entangling camera motion with object motion, which would otherwise introduce additional ambiguity and noise into the inference problem.
- **At least one rigid object undergoing translational motion.** Each video contains at least one rigid object whose dominant motion is translation. This requirement ensures that we can formulate well-defined kinematic inference tasks. Non-rigid objects and purely rotational motions are left out of the current benchmark and deferred to future work.[8]
- **Planar motion for `2D` tasks.** For `2D` instances, the target object and the reasoning target are constrained to move in a plane parallel to the image plane, i.e., the depth relative to the camera remains (approximately) constant over time. This assumption guarantees a consistent mapping between pixel displacement and world-space distance within each clip, making the 2D kinematic inference problem well-defined.

Video–text record schema. Each annotated instance in QUANTIPHY is represented as a structured video–text record. Table 3 shows a representative example.

Each record contains the following fields:

1. **`video_id`.** A unique identifier for the underlying video.
2. **`video_source`.** The data source from which the video was obtained (e.g., `simulation`, `lab`, or `internet`).
3. **`video_type`.** A four-letter code encoding the configuration of the task. The four characters denote, in order: (i) the type of physical prior (Size, Velocity, or Acceleration), (ii) whether the reasoning task is `2D` or `3D`, (iii) whether there is a single (`S`) or multiple (`M`) moving objects, and (iv) the background type, that is, plain (`X`), simple (`S`), or complex (`C`). More details are included in subsection B.2.
4. **`fps`.** The frame rate of the video, used to convert frame indices into time and to compute velocities/accelerations consistently.
5. **`inference_type`.** A two-letter code indicating whether the prior and the inference target are static or dynamic over time: `S` denotes a static quantity, and `D` denotes a time-dependent (dynamic) one. The first letter

[8] In this work, we adopt a relaxed definition of rigid objects: we consider an object "rigid" if its motion can be consistently approximated by a stable center of mass across frames. This includes some entities that may exhibit slight non-rigid deformation (e.g., a flying bird or a walking person), as long as their motion remains locally trackable and structurally coherent.

Property	Example value
video_id	simulation_0032
video_source	simulation
video_type	A3MC
fps	30
inference_type	DD
question	What is the acceleration of the orange car at 1.0s in m/s^2?
ground_truth_prior	gravity acc = 9.8 m/s^2

depth_info	t=1s, distance_ball_camera = 13.80 m;
	t=2s, distance_ball_camera = 13.80 m;
	t=1.5s, distance_orange_camera = 10.18 m;
	t=2s, distance_orange_camera = 10.40 m;
	t=2.5s, distance_green_camera = 4.32 m;
	t=3s, distance_green_camera = 6.91 m

ground_truth_posterior	2.86

Table 3. **Example of a single video–text record in QUANTIPHY.**

corresponds to the prior, and the second to the posterior.

6. **question.** The natural-language prompt presented to the VLM. We ensure that each question explicitly specifies the physical unit (e.g., m, cm/s, m/s^2) and, for velocity or acceleration, clearly indicates whether the query concerns an instantaneous quantity at a given timestamp or an average quantity over an interval.
7. **ground_truth_prior.** The physical prior provided to the model, formatted as a positive numeric value with unit (e.g., gravity acc = 9.8 m/s^2). We enforce consistent formatting to simplify parsing and downstream use.
8. **depth_info.** Depth annotations used only for 3D reasoning tasks. This field contains depth values (in metric units) for the prior object and, when needed, for the inference target at one or more timestamps. Depth information is designed so that, in principle, the depth of the inference target can be recovered from the provided entries. The formatting mirrors that of **ground_truth_prior**.
9. **ground_truth_posterior.** The numeric ground-truth answer to the kinematic inference question, represented as a positive scalar without unit (the unit is part of the question text).

Balance and diversity. Finally, we design the dataset to be both balanced across task types and diverse in content.

- **Fine-grained video types.** Using the four-letter **video_type** code, we partition all clips into 36 fine-grained categories. We ensure that each category contains at least four videos, so that every configuration is represented non-trivially.
- **Balanced core task categories.** We strive to keep the four core inference task categories (2D-Static, 2D-Dynamic, 3D-Static, 3D-Dynamic) approximately balanced in terms of the number of videos and associated questions, enabling fair comparison across conditions.
- **Rich scenes and motion patterns.** We cover a broad spectrum of spatial scales and motion types. Scenes range from astronomical (e.g., planetary motion), to everyday macroscopic settings (e.g., traffic, sports), to microscopic phenomena (e.g., cells and bacteria). Motion patterns include uniform motion, accelerated and decelerated linear motion, projectile motion, pendulum-like oscillations, and centripetal motion, among others. This diversity is crucial for probing whether VLMs' quantitative reasoning generalizes beyond narrow, highly stylized scenarios.

B.2. Video Types

B.2.1. Video Categories Definition

As described in the section , we assign each video a four-character code that encodes its physical prior, dimensionality, object setting, and background type.

First character (S / V / A). The first character specifies which physical prior the video is constructed to probe. For a designated object in the scene, we use

- S = size
- V = velocity
- A = acceleration

Thus, the first character takes one of $\{S, V, A\}$, indicating whether the ground-truth physical quantity is size, velocity, or acceleration for that object, as illustrated in Figure 11.

Second character (2 / 3). The second character indicates whether the video is 2-Dimensional(2D) or 3-Dimensional(3D).

- 2D = planar video
- 3D = volumetric video

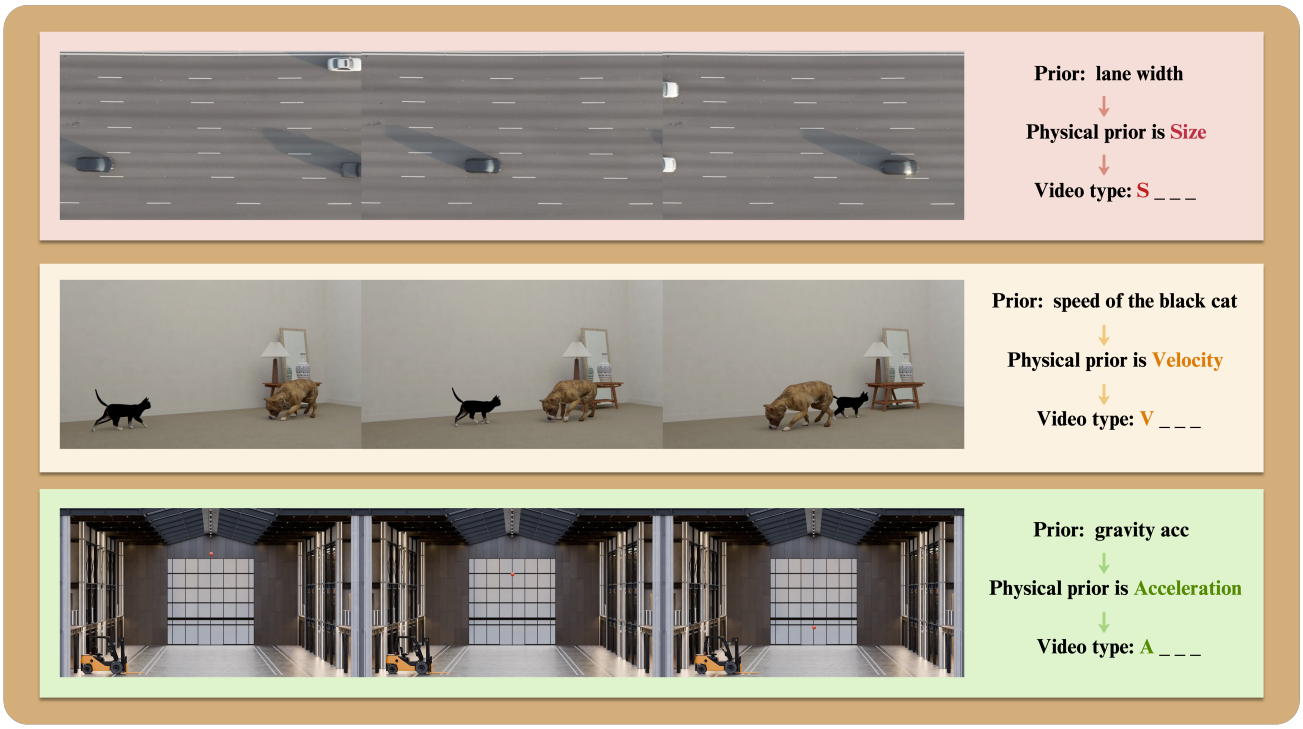


Figure 11. **Examples of S/V/A scene.**

More precisely, the second character is set to `2` when, at every frame, the distances from the moving object, the ground-truth reference object, and every object referenced in the inference question to the camera are exactly equal, so they occupy a single depth layer with no relative depth or parallax among them in the rendered image, and to `3` otherwise. Video examples are shown in Figure 12.

Third character (`S` / `M`). The third character distinguishes between "single-object" and "multiple-object" settings, relative to the queried object(s) rather than the sheer number of objects visible in the scene:

- `S` = single-object
- `M` = multiple-object

As shown in Figure 13, this label depends on how many objects the viewer needs to reason about. We use `S` (single-object) when the reasoning process only involves one object. Other objects may appear, but they only serve as background or generic distractors. We use `M` (multiple-object) when answering any of the questions requires reasoning about two or more objects, for example by comparing their sizes or speeds, or by using one object as an explicit reference for another.

Fourth character (`X` / `S` / `C`). The fourth character's examples are illustrated in the Figure 14. encodes the complexity of the background.

- `X` = plain background, typically a single uniform RGB color with essentially no texture, clutter, or noise;
- `S` = simple background, which may contain mild lighting or shading variations but remains visually uncluttered;
- `C` = complex background, with rich textures, multiple visible objects, more intricate lighting, and substantial visual "noise".

The boundary between "simple" and "complex" backgrounds is somewhat subjective, but during dataset construction we deliberately separated these categories and designed scenes so that their visual difference is as clear as possible.

Putting these components together, for example, a code such as `A3MC` indicates that the video (i) targets the acceleration prior (`A`), (ii) is rendered as a 3D video (`3`), (iii) is labeled as a multiple-object setting (`M`) because at least one inference question asks about an object different from the one whose acceleration prior is defined, and (iv) uses a complex background.

B.2.2. Quantitative breakdown of video types.

To complement the qualitative description above, we now provide a quantitative summary of how clips are distributed across the four-character codes. The benchmark contains 569 videos in total, of which 328 are `2D` and 241 are `3D`,

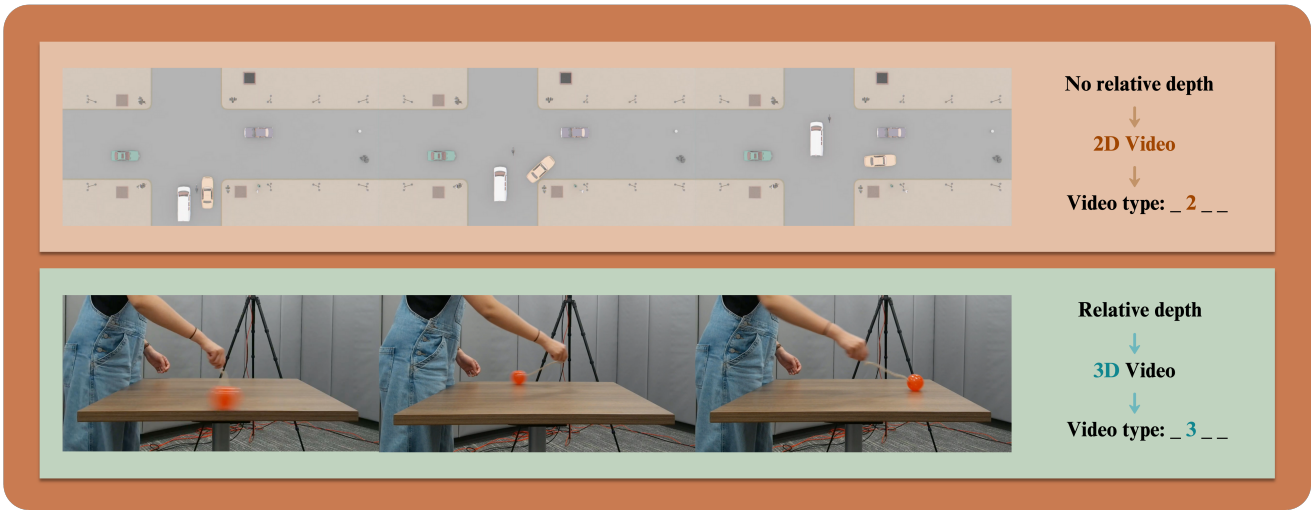


Figure 12. **Examples of 2/3 scene.**

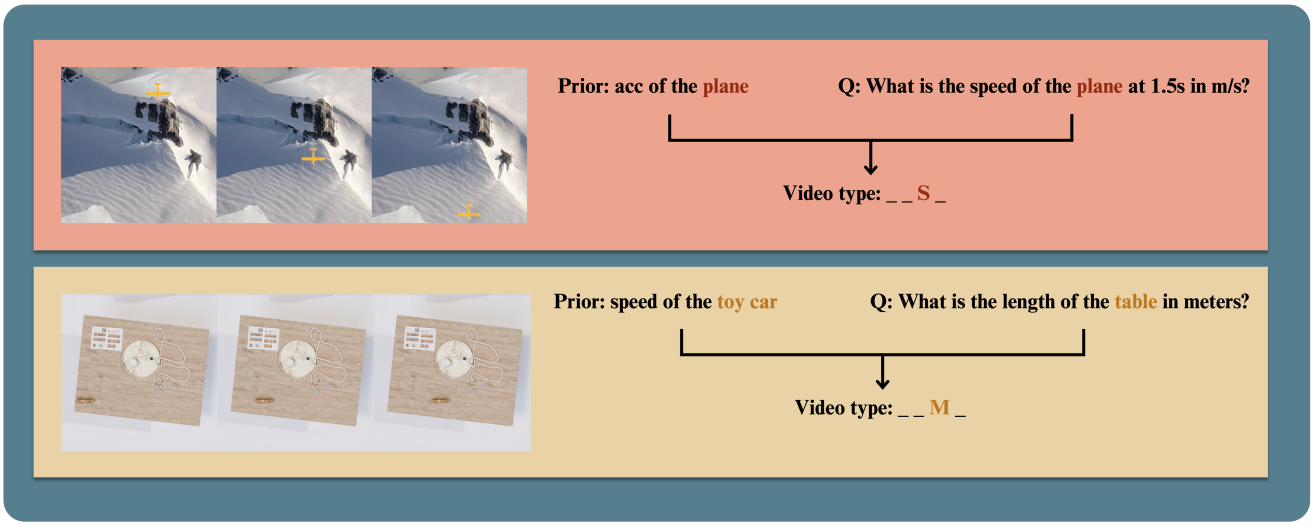


Figure 13. **Examples of S/M scene.**

yielding an approximately 4:3 split between planar and volumetric setups (about 58% `2D` and 42% `3D`). This ensures that both `2D` and `3D` configurations are substantially represented rather than the dataset being dominated by a single family of scenes. Table 4 reports the corresponding counts for each individual code and for the `2D` versus `3D` groups. By construction, combining three physical priors (`S` / `V` / `A`), two dimensionalities (`2` / `3`), two object settings (`S` / `M`), and three background types (`X` / `S` / `C`) yields 36 distinct four-character codes, and all 36 appear in the dataset. For `2D` videos, the codes are `A2SX`, `A2SS`, `A2SC`, `A2MX`, `A2MS`, `A2MC`, `S2SX`, `S2SS`, `S2SC`, `S2MX`, `S2MS`, `S2MC`, `V2SX`, `V2SS`, `V2SC`, `V2MX`, `V2MS`, `V2MC`; for `3D` videos, the codes are `A3SX`, `A3SS`, `A3SC`, `A3MX`, `A3MS`, `A3MC`, `S3SX`, `S3SS`, `S3SC`, `S3MX`, `S3MS`, `S3MC`, `V3SX`, `V3SS`, `V3SC`, `V3MX`, `V3MS`, `V3MC`. Each code is instantiated by at least 4 clips, so even the smallest categories have non-trivial support. The largest corpus (`V2MC` in our current dataset) contains 51 clips. Across all 36 codes, per-code counts range from 4 to 51 clips, with the majority lying between 5 and 35. This distribution avoids both extremely rare "one-off" configurations and a few overwhelmingly frequent ones. The precise per-code counts are given in Table 4. Table 4 also details how each video is obtained. The `Blender` column counts fully

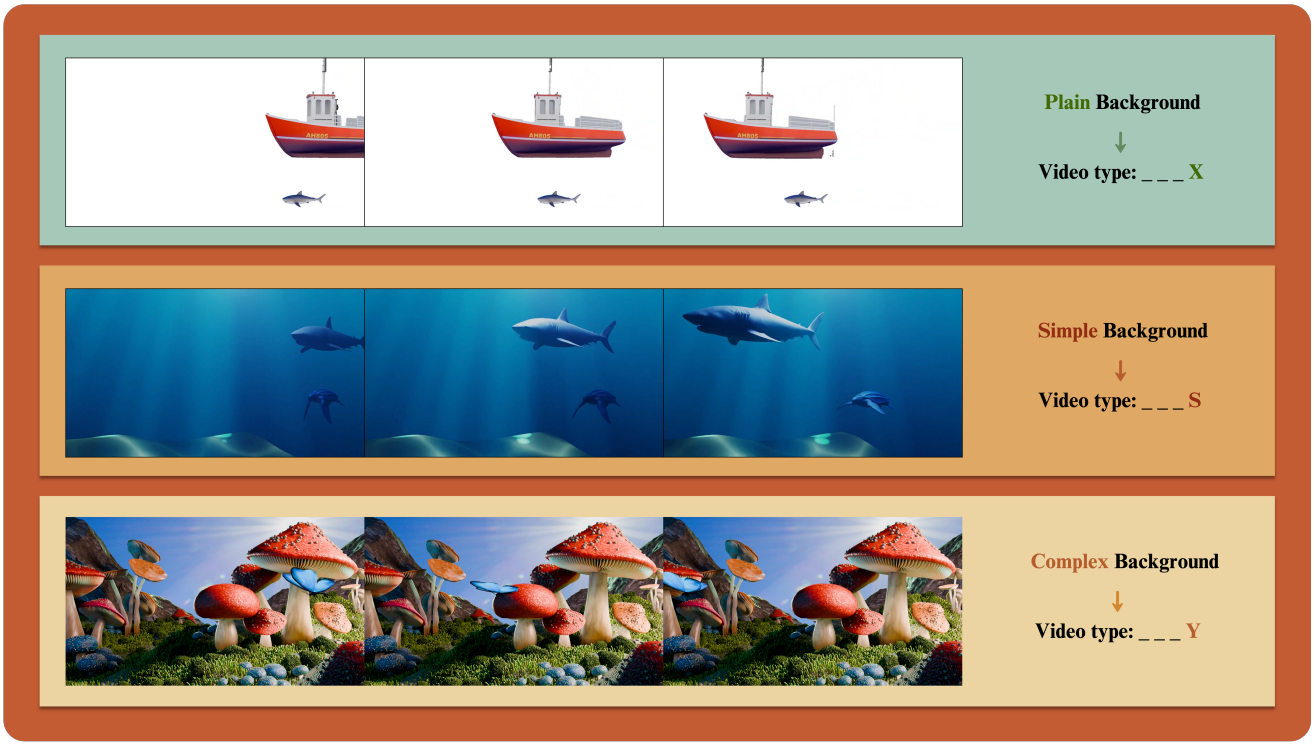


Figure 14. **Examples of X/S/C scene.**

synthetic clips rendered directly in `Blender` from our own scenes, contributing 300 videos. The `Internet` column counts 72 clips sourced from existing online footage that we curate and annotate. The `Captured` column counts 112 clips that we record ourselves (for example, using handheld cameras or screen recordings). The `Segmented` column counts 85 clips created by segmenting foreground objects from source footage and compositing them into new backgrounds. For each four-character code, the Total column gives the number of clips that realize that configuration. Taken together, these statistics in Table 4 show that the dataset spans all intended configurations, with each category populated by multiple clips and supported by a mix of blender-rendered, internet-sourced, captured, and segmented videos.

C. Details of Data Collection

C.1. Blender Simulation

C.1.1. Blender Toolkits and Asset Sources

Asset sources and selection. We construct Blender scenes using 3D assets sourced from online repositories, primarily BlenderKit and Sketchfab. We deliberately use these two libraries for complementary purposes: BlenderKit mainly provides complex themed environments that serve as background layouts, whereas Sketchfab mainly provides rigged and animated foreground objects whose motions can be reused with minimal manual keyframing.

BlenderKit for complex themed environments. BlenderKit is a community-driven asset library that is tightly integrated into Blender's interface and offers a large collection of render-ready models, materials, High Dynamic Range Images(HDRIs), brushes, and complete scenes that can be searched and inserted directly from within Blender. In our pipeline, we primarily rely on BlenderKit for complex "themed" environments, such as indoor rooms, streets, architectural spaces, and other richly cluttered layouts at both small and large spatial scales. These pre-built scenes typically include coherent lighting, materials, and background geometry (for example, furniture, buildings, and vegetation), which allows us to quickly instantiate diverse indoor and outdoor environments without having to model every object or compose every layout from scratch. This substantially reduces the authoring effort for environment design while still giving us control over camera placement, object insertion, and motion trajectories.

Sketchfab for rigged and animated foreground objects. Sketchfab is a large community platform for hosting and distributing 3D content, including many rigged and animated models across categories such as animals, vehicles, and articulated characters. In our work, we mainly use

2D/3D	Video Type	Blender	Internet	Captured	Segmented	Total	2D/3D Total
2D	A2SX	0	0	0	2	2	
	A2SS	10	11	0	0	21	
	A2SC	7	0	0	0	7	
	A2MX	6	0	1	11	18	
	A2MS	14	3	1	0	18	
	A2MC	19	1	1	0	21	
	S2SX	0	0	0	5	5	
	S2SS	11	6	0	0	17	
	S2SC	7	9	0	0	16	
	S2MX	8	0	0	7	15	
	S2MS	11	5	0	0	16	
	S2MC	16	24	0	0	40	
	V2SX	2	0	0	1	3	
	V2SS	10	1	0	0	11	
	V2SC	10	1	0	0	11	
	V2MX	13	0	0	19	32	
	V2MS	14	3	0	0	17	
	V2MC	43	11	0	0	54	
							324
3D	A3SX	2	0	6	1	9	
	A3SS	2	0	9	0	11	
	A3SC	3	0	7	0	10	
	A3MX	1	0	4	2	7	
	A3MS	3	0	4	0	7	
	A3MC	20	0	4	0	24	
	S3SX	1	0	8	2	11	
	S3SS	2	0	8	0	10	
	S3SC	2	0	8	0	10	
	S3ZX	0	0	1	0	1	
	S3MX	0	0	9	12	21	
	S3MS	2	0	10	0	12	
	S3MC	22	0	10	0	32	
	V3SX	3	0	2	0	5	
	V3SS	2	0	2	0	4	
	V3SC	3	0	2	0	5	
	V3MX	2	0	6	14	22	
	V3MS	4	0	6	0	10	
	V3MC	19	0	6	0	25	
							236
Total		294	75	115	76	560	560

Table 4. **Statistics of videos**.

Sketchfab to obtain foreground objects that already come with a skeleton rig and a small set of reusable animation clips. Typical examples include wing-flapping cycles for flying birds (such as eagles spreading and flapping their wings) and swimming cycles for fish with realistic body undulation. Instead of manually keyframing these motions, we can directly reuse or lightly retarget the provided animations in our scenes. This makes it much easier to populate environments with moving agents whose motion is visually plausible, while significantly reducing the time spent on low-level animation authoring.

Licensing and reuse. For both BlenderKit and Sketchfab, we restrict ourselves to assets whose licenses explicitly allow reuse and modification in works. These licens-

ing choices ensure that all assets in our dataset are used in a copyright-compliant way and that future researchers can reconstruct our pipeline using the same publicly available resources.

C.1.2. Two Motion Simulation Types

The Blender-generated portion of our dataset contains two complementary categories of motion, reflecting the major paradigms of movement in computer graphics and physics-based animation.

(1) Keyframed Motion. Examples of this category are shown in Figure 16. This category includes humans, animals, and other objects whose motion is defined using rigged skeletons or keyframed animation curves shown in Figure 17. Because these trajectories are authored manually rather than produced through physical simulation, they are visually plausible but not physically constrained.

A key implication is that the resulting motion does not necessarily obey real-world physical laws. For example, in the lunar-walking scene (Figure 22), the astronaut's push-off, airtime, and landing motion are shaped by artist-edited animation curves. Although the motion is loosely inspired by reduced lunar gravity, we do not compute the animation using the Moon's gravitational constant nor derive the trajectory from force-based simulation. These sequences should therefore be interpreted as perceptually reasonable approximations of motion rather than physically calibrated ground truth.

(2) Physics-Driven / Force-Based Motion. Examples of this category are shown in Figure 15. This category consists of rigid bodies (shown in Figure 18) moving directly under Newtonian dynamics. Their trajectories are generated by applying explicit forces (e.g., gravity, impulses) inside a physics engine or by specifying analytical kinematic profiles (e.g., constant velocity, uniform acceleration, projectile motion). These clips yield clean and physically interpretable motion, allowing us to provide exact ground-truth displacement, velocity, and acceleration.

For all videos in this category, we explicitly state the forces or kinematic parameters involved, and these values constitute part of the known prior information for each question.

Clarifying Prior Knowledge vs. Visual Assumptions. In an era where videos may originate from real capture, simulation software, procedural animation, or generative models, visual motion alone does not guarantee adherence to physical laws. Accordingly, VLMs should reason strictly based on the prior conditions we provide, rather than relying on pretrained assumptions about how objects "should" move.

When prior information explicitly specifies a force or acceleration (e.g., "the object accelerates at $2.5\,\mathrm{m/s^2}$"), that value serves as authoritative ground truth. When such information is not given, the model should not infer physical constants, such as Earth's gravity, only from the visual appearance of the motion. Because our dataset includes diverse simulated and animated sequences, correct reasoning requires using only the provided priors, not assumed real-world physics.

C.1.3. Blender Videos Construction

Quantitative Overview. In the Blender subset, we start from approximately 125 distinct base 3D models and 81 base scenes. From these bases, we generate 312 unique Blender video clips. Beyond simply increasing the size of the dataset, Blender enables us to construct diverse scenes and controlled trajectories of objects in simplified environments. Because these scenes are generated procedurally, we can precisely specify object sizes, positions, and kinematics via scripts and directly compute accurate ground-truth physical quantities for each clip. As a result, the Blender subset is not only large, but also covers a broad spectrum of scene scales and motions under tight experimental control.

Scene Construction. We construct a diverse collection of synthetic scenes to broaden the range of physical situations represented in the benchmark. Specifically, we design both everyday indoor spaces (e.g., rooms with furniture and household objects, see Figure 19) and outdoor spaces (e.g., natural environments with varied terrain, vegetation, and water, see Figure 20). These scenes differ in layout, depth structure, and illumination, so that models must handle physical reasoning under heterogeneous visual conditions rather than overfitting to a single canonical setting.

Beyond such everyday environments, we also include scenes that are difficult or impossible to capture, measure, or systematically manipulate in the real world. Examples include microscopic settings (e.g., red blood cells moving through a vessel, see Figure 21), extraterrestrial scenarios (e.g., astronaut motion on the Moon, see Figure 22), and more.

Starting from base assets and scenes, we explicitly control geometry, lighting, and background complexity to generate families of videos that fall into different categories in our taxonomy. For instance, in the red-blood-cell scene (Figure 23), we derive a 3MX-style variant by removing the vessel wall, discarding all but two target cells, and replacing the background with a uniform RGB field. Using the same procedure, as shown in the Figure 24 we construct simple-background versions of more complex scenes by simplifying clutter while keeping coarse structural cues (e.g., horizon lines, major surfaces, object structures) and realistic lighting.

It is important to note that, although we present several examples that appear to reuse the same underlying scene. In practice, not all plain-background videos and simple-background videos are reused across conditions. In addition, we also include videos that are uniquely constructed to appear only in the plain-background condition or only in

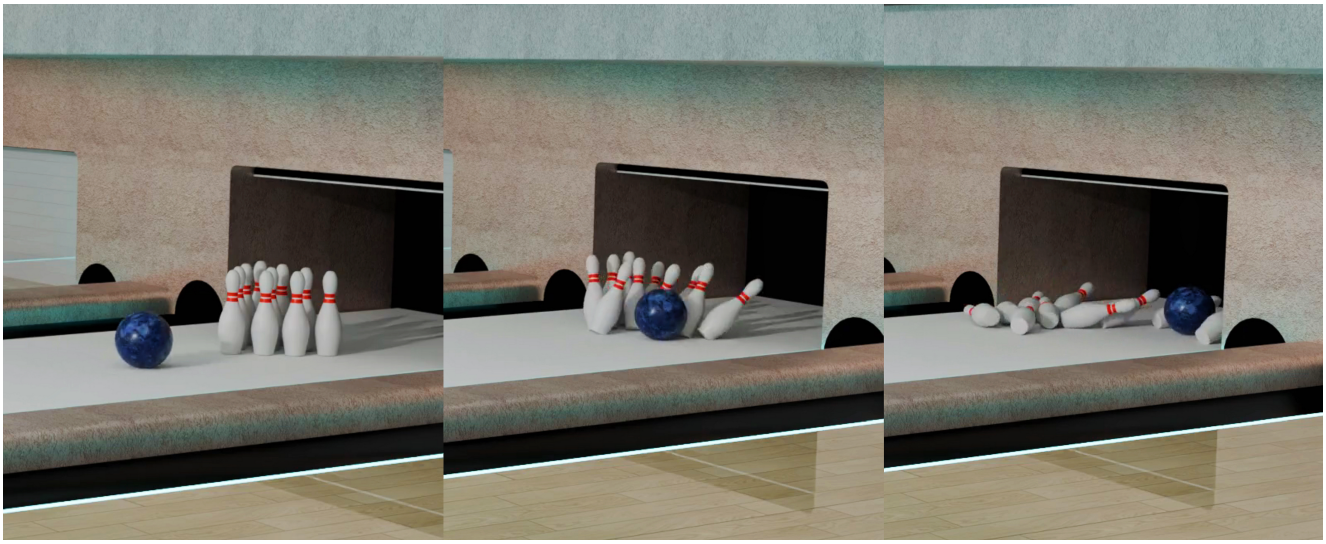


Figure 15. **Physics-driven example.** A bowling ball collides with pins under Newtonian simulation. The motion and resulting trajectories arise directly from rigid-body dynamics and elastic collisions, making this clip representative of our force-based motion category.

Figure 16. **Keyframed example.** The floating swim ring follows a manually authored animation curve rather than buoyancy, drag, or wind forces. Its trajectory is visually plausible but not physically derived, representing our keyframed motion category.

the simple-background condition. This design choice is intended to increase the diversity of the video data. Reusing identical base scenes improves the efficiency of dataset construction, but it may also introduce potential biases or unwanted correlations when evaluating VLMs, so we deliberately balance such reuse with the creation of novel, non-paired scenes.

For the objects in each scene, we specify target object sizes, velocities, and accelerations using real-world statistics gathered from online references (e.g., typical dimensions and speeds of vehicles, animals, and more). Whenever possible, moving objects are modeled at real-world scale; in rare cases where real-world-scale objects yield motion trajectories that are barely discernible in the rendered videos, we apply uniform scaling to increase perceptual visibility while preserving the underlying physical relationships.

An example is the "ice cube falling into a cup" scene illustrated in the Figure 25. If we model the cup and cube at their real-world dimensions, the cube traverses the camera frustum in essentially a single frame, making its falling trajectory almost invisible to observers and thus providing little signal for quantitative evaluation. To address this, we uniformly enlarge both the cup and the ice cube by a factor of 10. After scaling, the cube remains visible from frame 11 to 22, and the rendered video reveals a clear multi-frame trajectory. This controlled rescaling not only makes the motion measurable but also allows us to probe whether VLMs recover physical quantities from the observed kinematics given the priors specified in the prompt, or whether they instead rely predominantly on generic pre-trained knowledge and commonsense expectations about how such objects "should" behave.

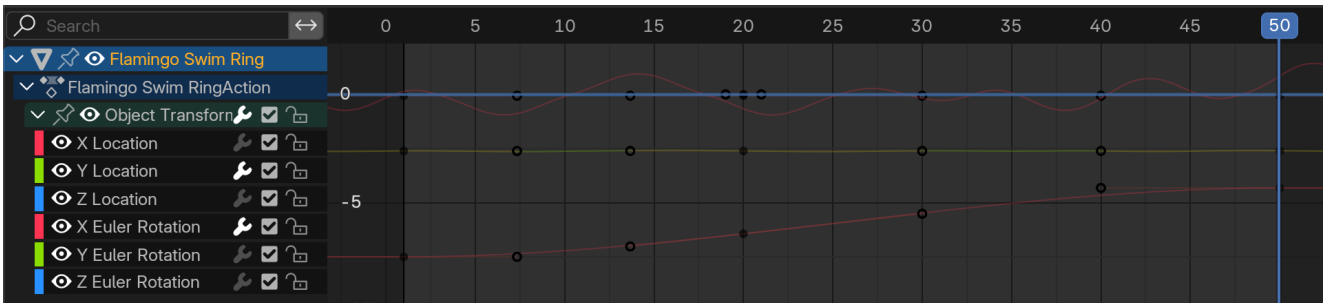


Figure 17. **Keyframed animation-curve example.** The animation curve controlling the swim ring in Figure 16. The ring moves forward along a manually authored trajectory, while small randomized perturbations in translation and rotation are added to imitate the visual appearance of floating motion.

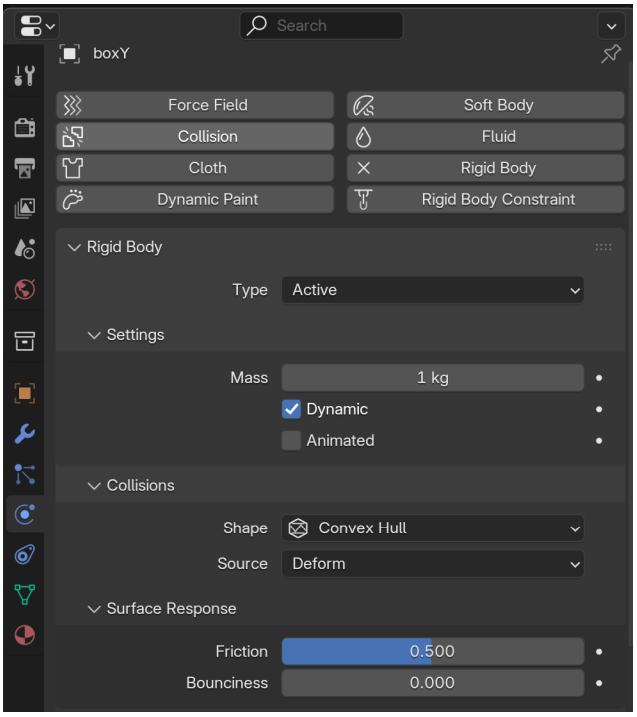


Figure 18. **Physics-driven rigid-body example.** Rigid-body simulation of objects such as the bowling pins reacting to the impact of an incoming ball in Figure 15. The pins' motion, tipping, and scattering are governed entirely by Newtonian dynamics and collision responses within the physics engine.

Object Motion Design and Implementation. As discussed in subsubsection C.1.2, our motion design follows two main paradigms: keyframed motion and physics-driven motion. The construction of physics-driven scenes is illustrated in Figure 15. For the keyframed cases, in addition to manual editing in the Graph Editor, we also use scripts to automate the animation process.

We distinguish two main classes of scripted motion: (i) analytic one-dimensional motion along a single axis, and (ii) curve-following motion along human-designed paths.

For analytic 1D motion, we explicitly encode standard kinematic equations and bake the resulting trajectories as keyframes. Given an object with initial world-space position $\mathbf{x}_0$, a target duration T, and a desired acceleration a along the $-Y$ axis, our script iterates over frames f and computes the physical time

$$t = \frac{f - \texttt{START\_FRAME}}{\texttt{FPS}}.$$

The corresponding displacement is computed using the constant-acceleration formula

$$s(t) = \tfrac{1}{2}at^2.$$

```
disp_y = 0.5 * ACCEL * (t ** 2)
new_y  = start_loc.y - disp_y
obj.location = Vector((start_loc.x, new_y,
↪  start_loc.z))
obj.keyframe_insert(data_path="location",
↪  index=-1, frame=f)
```

This directly realizes $s(t) = \frac{1}{2}at^2$ in world coordinates. For constant-velocity motion, we instead use the linear relation:

$$s(t) = vt,$$

implemented via

```
disp_y = VEL * t
new_y  = start_loc.y - disp_y
```

while keeping the same frame loop and keyframe insertion logic. After all keyframes are written, we programmatically set each F-curve's interpolation mode to `LINEAR`, overriding Blender's default Bézier interpolation to ensure that the frame-to-frame displacement matches the analytically specified velocity or acceleration profile rather than being inadvertently smoothed or distorted.

Figure 19. **Examples of indoor scene.**

Figure 20. **Examples of outdoor scene.**

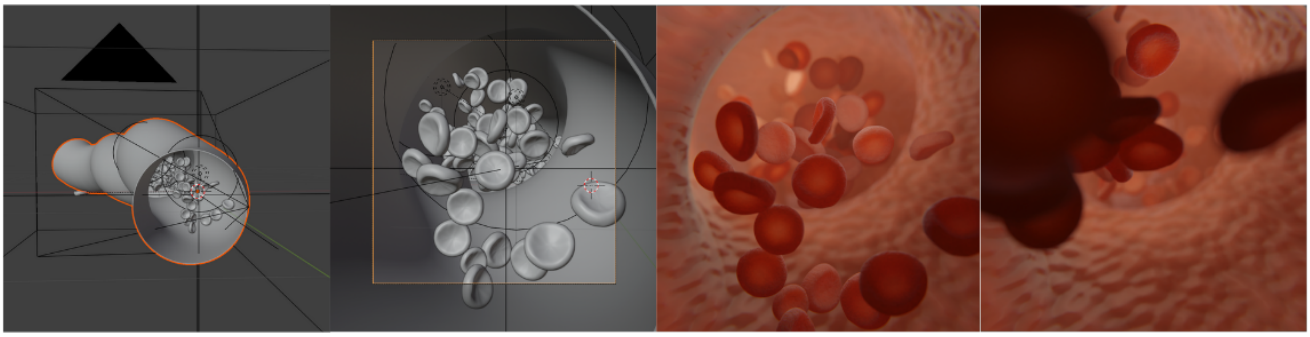


Figure 21. **Examples of microscopic scene.**

Figure 22. **Examples of extraterrestrial scene.**

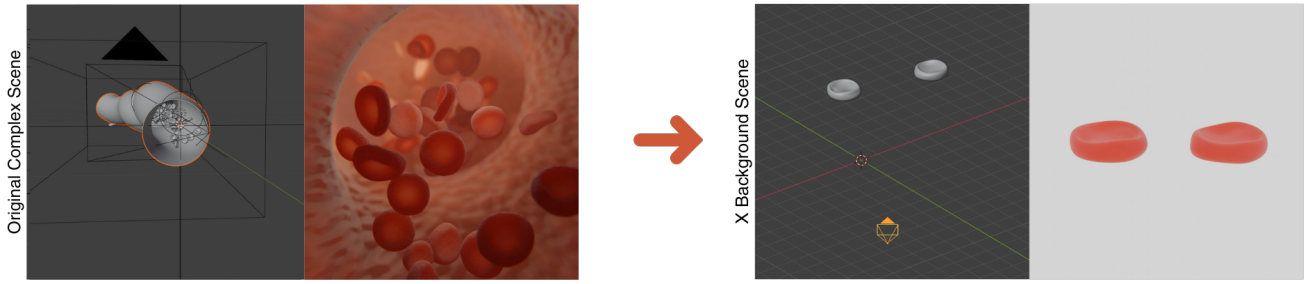


Figure 23. **Examples of building X background scene.**

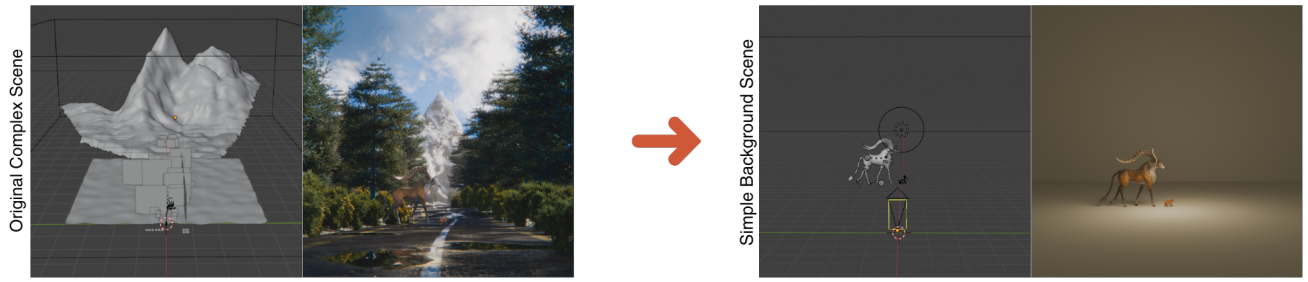


Figure 24. **Examples of building S background scene.**

For more complex motions along curved paths, we first author an approximate trajectory in Blender (e.g., by manually keyframing a car following a road or a particle moving along a spiral), and then reparameterize this path in Python to obtain physically interpretable kinematics. Concretely, we sample the object's world-space position at each frame of the original animation:

```
positions = []
for f in range(frame_start, frame_end + 1):
    scene.frame_set(f)
    pos = obj.matrix_world.translation.copy()
    positions.append(pos)
```

We then compute the cumulative arc length along this polyline.

```
distances = [0.0]
for i in range(1, len(positions)):
    d = (positions[i] - positions[i-1]).length
    distances.append(distances[-1] + d)
total_length = distances[-1]
duration     = (frame_end - frame_start) / fps
```

We then derive a desired kinematic profile in terms of traveled distance $s(t)$ along the curve. For constant-speed motion, we set $v = L/T$ with $L = \texttt{total\_length}$ and use

$$s(t) = vt,$$

implemented as

```
target_dist = speed * t
```

which corresponds to s(t)=vt. For constant-acceleration motion along the same curve, we instead use the quadratic form

$$s(t) = v_0 t + \tfrac{1}{2} a t^2,$$

with user-specified v_0 and a. In both cases, for each frame we find the segment $[i-1, i]$ such that

$$\texttt{distances}[i-1] \le s(t) \le \texttt{distances}[i],$$

and linearly interpolate between the sampled positions:

```
ratio  = (target_dist - distances[i-1]) /
↪  (distances[i] - distances[i-1])
new_loc = positions[i-1].lerp(positions[i],
↪  ratio)
obj.location = new_loc
obj.keyframe_insert(data_path="location",
↪  frame=f)
```

Finally, as in the analytic case, all resulting keyframes are set to linear interpolation. This arc-length reparameterization scheme lets us reuse authored trajectories while imposing well-defined kinematic profiles (constant velocity, constant acceleration, or other time–distance schedules) in physical units. Together with the untouched native animations, these scripted motions give our benchmark a wide spectrum of motion patterns, from simple 1D acceleration

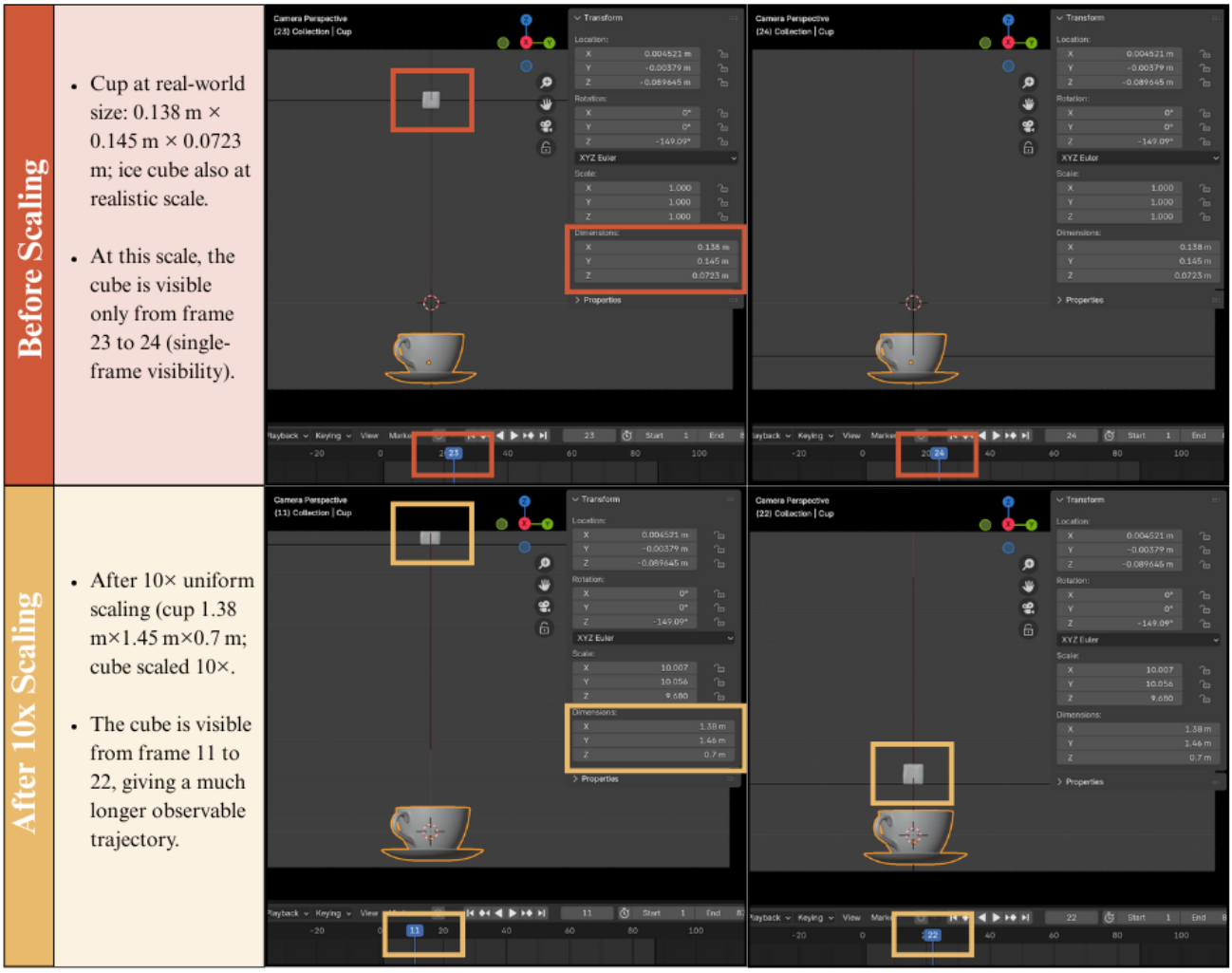


Figure 25. **Examples of scaled scene.**

to complex, irregular dynamics, under tight quantitative control.

Rendering. All Blender-generated videos in QUANTIPHY are rendered with the Cycles and EEVEE path-tracing engine using a physically based workflow. Videos use multiple spatial resolutions, including 1920×1080 (16:9), 1080×1080 (1:1), 480×960 (vertical), as well as several additional intermediate sizes, so that models see both landscape, square, and portrait-style content. The temporal sampling is likewise heterogeneous: frame rates in the benchmark include 24, 25, 30, 33, 60, and 120fps. Frames are exported via FFmpeg as MP4 files with H.264 compression and a constant frame rate, without any interpolation or stabilization, so each frame aligns exactly with the simulated timeline and ground-truth annotations.

Camera parameters are also varied to increase visual diversity. Intrinsics cover a range of focal lengths from wide to normal and telephoto views, and extrinsics place the camera at different heights, offsets, and azimuth/elevation angles around the scene, producing both frontal and oblique perspectives on object motion. Lighting setups combine different environment maps and local light sources, and objects are assigned a broad set of physically based materials (e.g., diffuse, glossy, metallic, translucent), leading to diverse shading, reflection, and contrast conditions. Together, these choices give QUANTIPHY wide coverage over resolutions, frame rates, viewpoints, and appearance statistics, while the underlying physical trajectories and numeric ground truth remain precisely controlled.

C.2. Lab Capturing

To further complement the diversity of QUANTIPHY with real-world data, a major part of our dataset comes from a customized motion capture system in a lab environment. Due to the nature of real-world data, lab captured videos

Depth Technology	Time of Flight
Wavelength	850 nm
Depth Range	*0.25–5.46 m (depending on depth mode)
Depth Resolution/FPS	Up to 1024×1024@15 fps (WFOV), 640×576@30 fps (NFOV)
Depth FOV	H 120° V 120° (WFOV), H 75° V 65° (NFOV)
RGB Resolution/FPS	Up to 3840×2160@25 fps
RGB FOV	H 80° V 51°
Processing	NVIDIA Jetson Nano
IMU	Supported

Table 5. **Specifications of the cameras used in our customized motion capture system.**

Objects	Dimensions
2 green tennis balls	d = 6.7 cm
2 pink tennis balls	d = 6.7 cm
2 white ping pong balls	d = 4 cm
1 purple yoga ball	d = 52.2 cm
1 soccer ball	d = 17.5 cm
1 basketball	d = 23.2 cm
1 red plastic ball	d = 7 cm
1 orange plastic ball	d = 5.7 cm
1 small whiteborad (slope)	30.8 cm x 23.2 cm x 0.5 cm
1 large whiteborad	81 cm x 60 cm x 1 cm
1 trashbin	34.5 cm x 89.1 cm 59.9 cm
1 tape	10.1 cm x 10.1 cm x 4.8 cm
1 white food box	19.5 cm x 6.0 cm x 7.2 cm
stuffed toy 1	14.5 cm x 11 cm x 8 cm
stuffed toy 2	29.5 cm x 15.0 cm x 10.0 cm
stuffed toy 3	8.1 cm x 6.4 cm x 19.6 cm
1 toy cookie	8.0 cm x 8.0 cm x 1.8 cm
1 cosmetic jar	7.2 cm x 7.2 cm x 6.0 cm
1 glass jar	6.8 cm x 6.8 cm x 8.5 cm
1 white cup	8.6 cm x 8.6 cm x 14.5 cm
1 pink water bottle	6.7 cm x 6.7 cm x 19.5 cm
1 marker	13.5 cm x 0.8 cm x 0.8 cm
1 black pen	14.5 cm x 1. 2 cm x 1.2 cm
1 green notebook	18.6 cm x 8.8 cm x 1.5 cm
1 white-covered book	12.5 cm x 19.5 cm x 2.2 cm
1 pencil case	6.7 cm x 7.2 cm x 21 cm
1 credit card	85.6 mm x 53.98 mm
1 deck of poker card	9.8 cm x 6.3 cm x 1.8 cm

Table 6. **List of physical objects and their measured dimensions used in our lab-captured videos.**

contribute only to the category of 3D.

MoCap setup. We use four Orbbec Femto Mega cameras to construct our customized motion capture system. The specifications of the cameras are shown in Table 5.

We designed two different camera setups and object arrangements in the MoCap system. The first setup covers a smaller spatial range, with the main camera placed close to the objects. In this setup, we capture motions of small objects such as a tennis ball, book, and ping pong ball moving on a desk. The second setup covers a larger range and is used to capture larger-scale motions such as basketball bouncing and accelerated motion of a trash bin. For both setups, we initialize the camera extrinsics via multi-view calibration with a checkerboard.

Collection workflow. For efficient video capture, we developed a tool with a user-friendly interface that supports synchronized recording across all four cameras. We collect videos based on a list of predefined questions. For each question, we set up the scene and objects without moving the cameras and then perform specific object motions according to the question. Three people were involved in the lab data collection workflow: one person was responsible for operating the tool (starting and pausing capture), and two people were responsible for setting up the scenes and executing the motions.

Unlike Blender simulation, capturing real-world data makes it difficult to directly manipulate specific scene parameters. We therefore manually configured the scenes to meet target conditions, such as the slope angle or the frequency of pendulum motion, using careful measurements within a set tolerance. For motions that require a controlled initial velocity, we used a motor running at s constant speed to pull the object and provide the desired initial velocity.

Post-processing. After capturing the raw videos, we obtain `.bag` files for each camera that contain the binary recordings. We follow the official camera SDK to decode the raw data. From this decoding, we obtain: (1) intrinsics for each camera, (2) timestamped RGB streams for each camera, (3) timestamped depth streams for each camera, and (4) relative extrinsics from the depth cameras to the RGB cameras. We then apply coordinate transformations to reproject the depth data into the image coordinates of the RGB videos, which

allows pixel-wise depth values to be read consistently.

With aligned image coordinates, it becomes possible to read out specific depth values for target objects (with respect to the main camera). A straightforward approach is to use segmentation masks for the target object and average the depth values over the masked pixels. However, this automated approach did not work well in practice because the depth information from the camera is often incomplete and ambiguous, especially at pixels with high motion, due to hardware limitations. As a result, we could not reliably automate depth extraction for the target objects using only segmentation masks.

To ensure high-quality depth measurements, we developed a UI-based tool that overlays the transformed depth map on top of the RGB frames. The tool allows users to click on the frame to read out exact depth values from the main camera at the selected pixels. An illustration of this UI-based tool is shown in Figure 26.

List of objects used. Table 6 lists all physical objects used in our controlled lab videos, together with their measured dimensions. These objects serve either as priors (with known size) or as inference targets in our kinematic tasks. All measurements are taken with a ruler or caliper in metric units before filming.

C.3. Internet Scraping

Our "internet" split in QUANTIPHY consists of real-world videos captured by commodity cameras, and is constructed from two sources that we treat under a unified category: (i) open-source online video platforms, contributing 42 clips; and (ii) videos recorded by the authors using smartphone rear cameras in everyday indoor and outdoor environments, contributing 30 clips. All of these videos are direct camera recordings of natural scenes, and thus closely reflect the statistics and imperfections of the physical world.

Why only 2D inference from internet data. Unlike our simulation and controlled-lab settings, internet videos do not come with calibrated depth, camera intrinsics, or precise object geometry. In particular, reliable metric depth is almost impossible to obtain for arbitrary internet footage. For this reason, we restrict internet videos to 2D kinematic inference tasks. For each selected clip, we manually construct a pixel ruler, measure pixel-level size/position trajectories of the objects of interest, and then use an approximate scale factor—derived from obvious real-world references in the scene (e.g., gravity $g = 9.8$ m/s^2, the length of a credit card, lane width on a road, or the speed of an airport conveyor belt)—to convert these pixel measurements into world-space kinematic quantities. While this procedure yields reasonably accurate ground truth, it is inherently less precise than the annotation pipelines used for our simulation and lab data. Consequently, we intentionally keep the proportion of internet data moderate in the overall benchmark.

Videos from open-source platforms. We choose open-source video platforms primarily because they avoid copyright issues, offer relatively high image quality, and provide diverse content. However, videos that satisfy the three screening criteria in subsection B.1 (static camera, at least one rigid object undergoing translational motion, and planar motion for 2D tasks) are relatively rare. Beyond these core constraints, we additionally require that each candidate clip contain at least one visually obvious physical prior that can be reasonably assumed or measured (e.g., gravity, a credit-card-sized object, standard lane width, or known conveyor-belt speed). These additional constraints further narrow the pool of usable videos. There is currently no off-the-shelf automatic pipeline for this selection process, so all internet clips are hand-picked by project members, who visually inspect candidates for compliance with our physical and annotation requirements. Representative examples are shown in Figure 27.

Author-recorded videos. Because suitable clips on open-source platforms are scarce, we complement them with 30 videos recorded by the authors using smartphone rear cameras in a variety of everyday scenes, including parking lots, road traffic, bedrooms, and indoor/outdoor sports venues. During recording, we enforce a fixed camera viewpoint and ensure that at least one rigid object exhibits predominantly translational motion, so that the criteria in subsection B.1 are satisfied. In many such settings, precise physical quantities (e.g., vehicle speed, basket height) cannot be directly measured. We therefore annotate these videos using the same pixel-ruler and approximate-scale procedure as for online platform videos, again restricting them to 2D inference tasks. Example frames from these author-recorded clips are shown in Figure 28.

Privacy and anonymization. For all internet videos, whether sourced from open platforms or recorded by the authors, we manually inspect frames for sensitive personal information. Whenever faces, license plates, or other identifying details appear, we apply blurring or masking before including the clip in the dataset. This ensures that our internet data respect both copyright and privacy constraints while still providing realistic real-world scenarios for quantitative kinematic inference.

C.4. Segmented Data

In QUANTIPHY, we aim to benchmark the capabilities of VLMs on videos with diverse background types. Video backgrounds often contain contextual information that may either aid or hinder the inference of a target object's physical properties. To comprehensively study this impact, we design experiments that isolate the target object by completely removing the background. Specifically, we compare the original videos against processed versions where

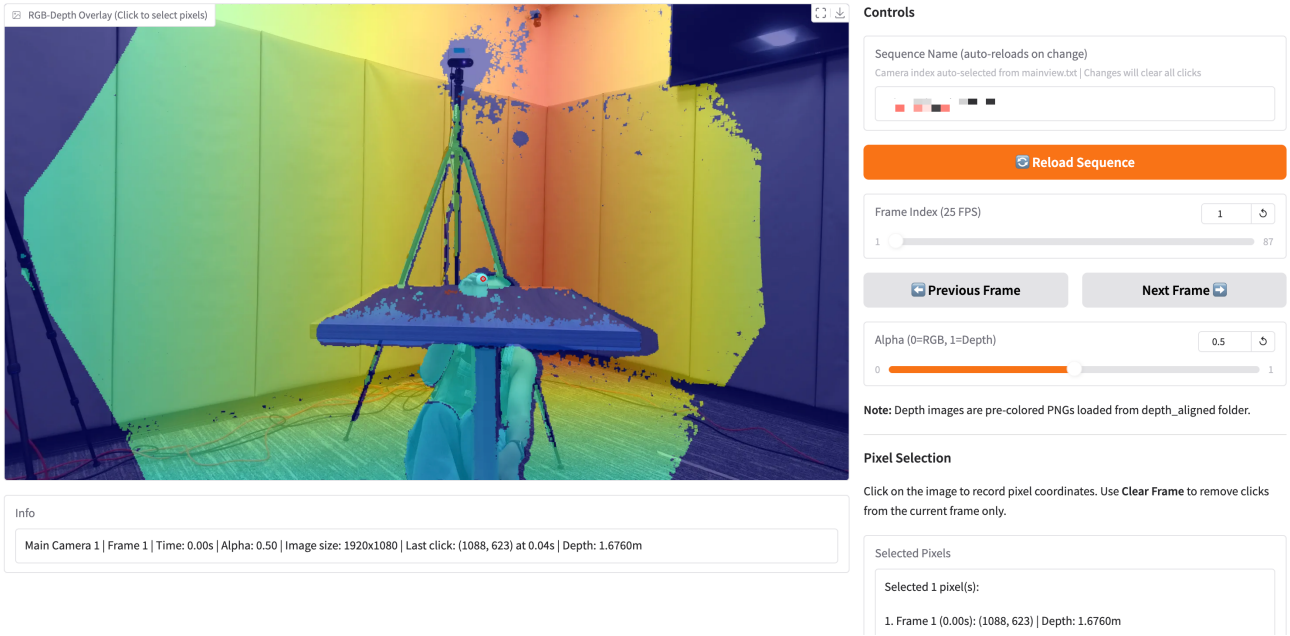


Figure 26. **The users' interface of the tool we have developed for obtaining depth value.**

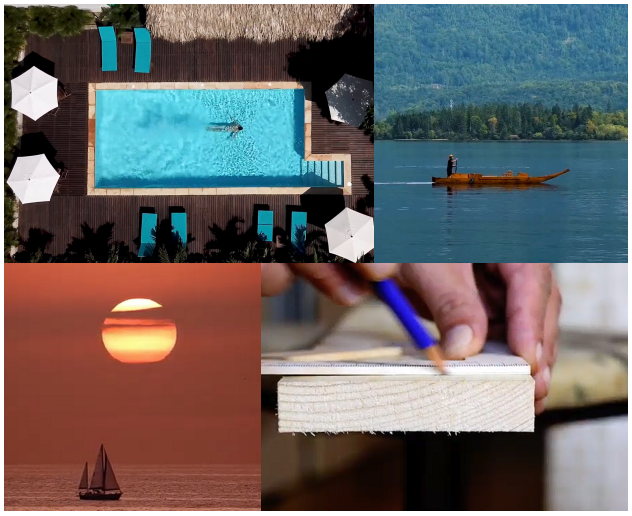


Figure 27. **Examples of videos from open-source platforms.**

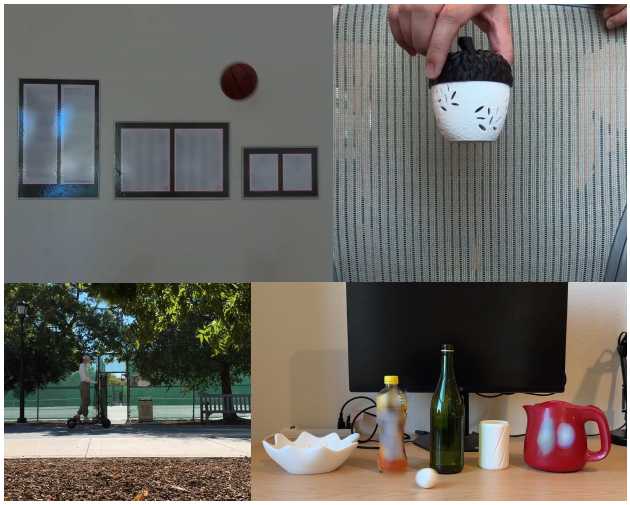


Figure 28. **Examples of self-recorded videos with identity removed.**

the background has been completely denoised and removed using a segmentation model.

State-of-the-art semantic segmentation models, such as SAM 2 [33], have demonstrated robust capabilities in tracking and segmenting arbitrary targets across various tasks. In our experiments, we employ SAM 2 as the backbone segmenter, utilizing a hybrid prompting strategy. To automate the pipeline, we leverage Grounding DINO 1.5 [35] to localize target objects using textual descriptions derived from our video-question pairs. The bounding boxes generated by Grounding DINO serve as box prompts for SAM 2, enabling automatic segmentation.

For complex scenes with multiple objects where the automated pipeline may fail, we incorporate manual intervention by providing point-based prompts to SAM 2. To streamline this workflow, we developed a custom UI-based tool that allows multiple annotators to label objects efficiently, thereby scaling the segmentation process. Figure 29 illustrates the user interface of this tool, and Figure 30, Figure 31, Figure 32 provides exemplary frames of the seg-

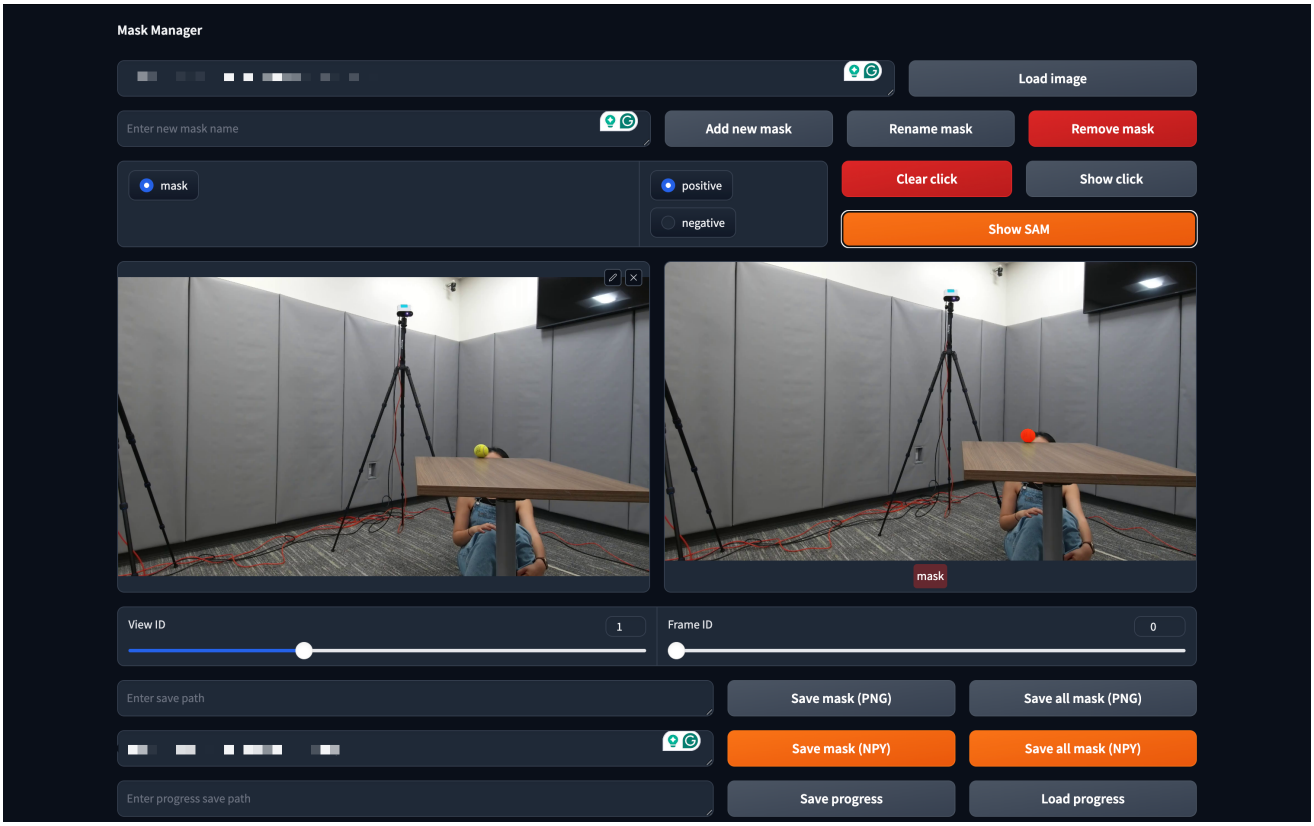


Figure 29. **The users' interface of the segmentation tool we have developed.**

mented videos for reference.

C.5. Quality Control

The data collection process for QUANTIPHY is highly diverse and complex. Since the data originates from heterogeneous sources with varying characteristics, establishing a universal automated protocol for quality assurance is challenging. To address this, we incorporated an additional manual review stage for all candidate data.

To maximize data quality, we manually excluded videos exhibiting excessive motion blur, severe occlusion of the target object, or objects that are difficult to model. Furthermore, we removed videos containing identifiable human subjects to ensure ethical compliance.

Following this review process, approximately 3% of the Blender data and 30% of the lab data were discarded, while for the videos scraped from the internet only 72 clips were retained.

C.6. Ethical Considerations

In compiling the QUANTIPHY dataset, comprising both simulated Blender environments and web-sourced videos, we strictly adhere to all relevant copyright and licensing regulations. For specific data assets requiring attribution, we provide full acknowledgments in the supplementary materials.

We implemented strict privacy measures throughout the data collection and annotation phases to ensure that no personally identifiable information (PII) is retained in the dataset. Relevant Institutional Review Board (IRB) documentation is available upon request. Furthermore, QUANTIPHY complies with ethical guidelines regarding content safety; we have rigorously screened the data to exclude biased or harmful content while prioritizing diversity to foster fairness and inclusivity.

D. Details of Data Annotation

D.1. Blender Simulation

We annotate five basic types of physical quantities: size, displacement over a given time interval, velocity, acceleration, and depth and distance.

D.1.1. Size

Blender size annotations are extracted directly from Blender's internal measurement readout to guarantee numerical accuracy. For most objects, size is obtained from the object's axis-aligned bounding box dimensions in world

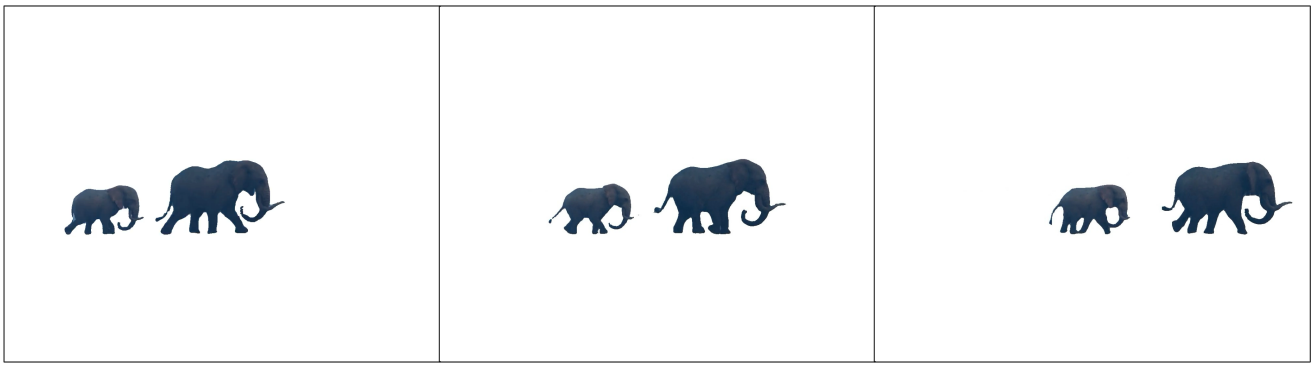


Figure 30. **Examples of segmented blender data.** After segmentation, we can replace the background image freely.

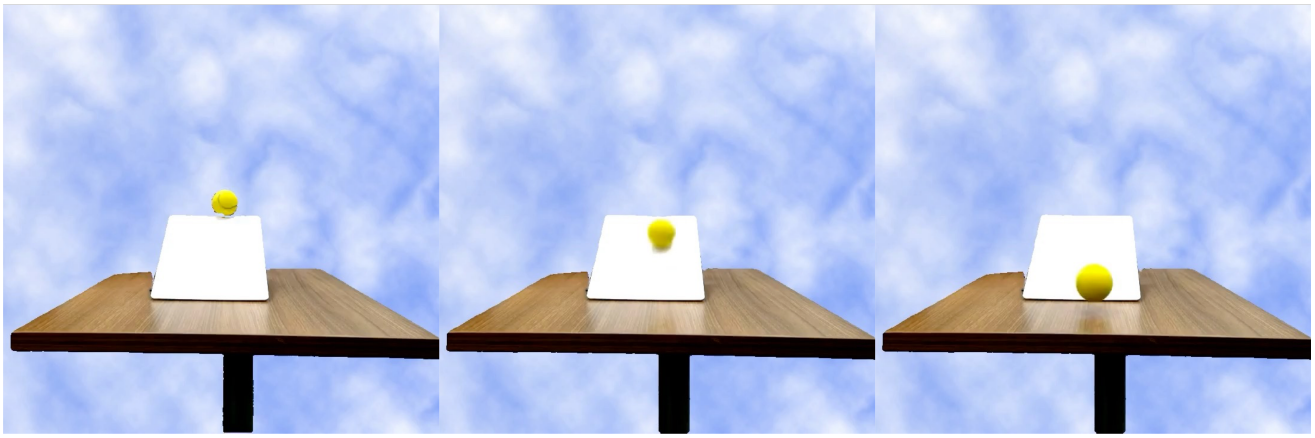


Figure 31. **Examples of segmented lab data.** After segmentation, we can replace the background image freely.

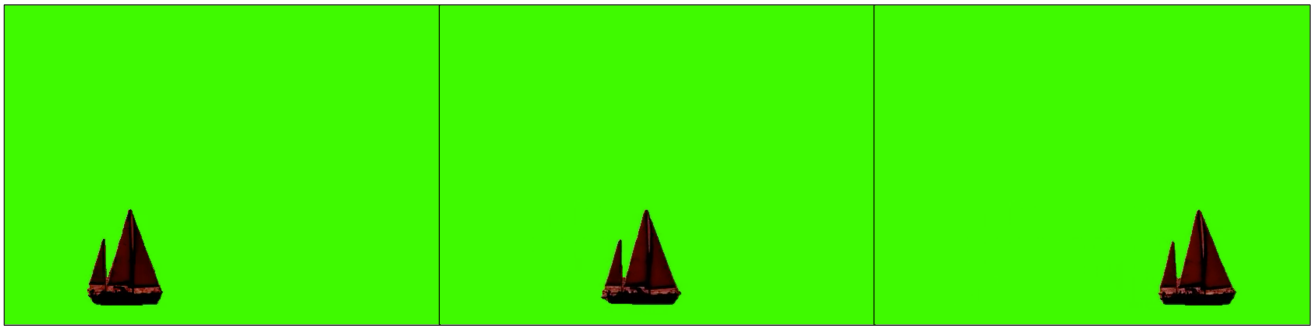


Figure 32. **Examples of segmented internet data.** After segmentation, we can replace the background image freely.

space. In other special cases, the dimensions of objects may change from frame to frame.

For example, as shown in the Figure 33, articulated humans height changes while they move, thus we explicitly measure three configurations within each clip:

1. **Rest standing height:** the height in a rest "T-pose" frame, where the character stands upright and fully extended.
2. **Minimum apparent height during walking:** the smallest height observed over the walking segment of the clip.
3. **Maximum apparent height during walking:** the largest height observed over the same segment.

All heights are obtained by selecting the corresponding frames in Blender and measuring the vertical extent in world coordinates. In practice, these three measurements are usually very close (often differing by only a few centimeters), but we still record all of them to keep the annotation protocol precise and consistent.

Figure 33. **Examples of human height measuring.**

Similarly, for other rigged objects (such as birds and insects), we track their width frame by frame over the flight segment of the clip and only record the smallest and largest width observed. See details in Figure 34. This is because, unlike the human case, these assets may not admit a clear, static "rest standing" pose in which the animal is fully extended. As a result, we do not define a separate rest measurement for flying objects.

Correspondingly, the inference questions and prior ground truth are phrased with the same level of precision. For human figures, we explicitly distinguish between the standing height at rest and the minimum and maximum height observed while walking. For flying animals, we clearly refer to the minimum or maximum width attained during flight.

D.1.2. Displacement or Path.

All displacement- or path-related labels are computed from object trajectories in Blender world space using internal custom Python scripts.

Time–frame alignment across scenes. In Blender, each scene specifies a render range with `scene.frame_start` and `scene.frame_end`, and `scene.frame_start` is not required to be 0. We define time $t = 0$ to occur at the render start frame, regardless of its numeric index. More precisely, if a scene has frame rate f frames per second and render start frame f_{start}, then the first frame of the exported video (timestamp $t = 0$ s) is Blender frame f_{start}, the frame at time $t = 1/f$ s is frame $f_{\text{start}} + 1$, and so on.

When a question specifies a time interval $[t_{\text{start}}, t_{\text{end}}]$ in seconds, we convert it to Blender frame indices by

$$i_{\text{start}} = \text{round}\big(f_{\text{start}} + f\, t_{\text{start}}\big), \quad i_{\text{end}} = \text{round}\big(f_{\text{start}} + f\, t_{\text{end}}\big).$$

This convention works uniformly for all scenes, including those whose render ranges begin at non-zero frame numbers (e.g., `frame_start = 20, 40, 60, ...`). In code, the conversion has the following form:

```python
scene   = bpy.context.scene
fps     = scene.render.fps
f_start = scene.frame_start
↪  # may be 0, 20, 40, ...

def time_to_frame(t_sec: float) -> int:
    return round(f_start + t_sec * fps)
```

When an interval is specified directly in frames, we simply take $(i_{\text{start}}, i_{\text{end}})$ as given and interpret the corresponding times as

$$t_{\text{start}} = \frac{i_{\text{start}} - f_{\text{start}}}{f}, \quad t_{\text{end}} = \frac{i_{\text{end}} - f_{\text{start}}}{f},$$

so that $t = 0$ always aligns with the first frame of the rendered video, irrespective of the absolute Blender frame index.

At any frame i, the scripts query the world-space position of the target object (e.g., the animated rigid body) via

```python
scene.frame_set(i)
loc = obj.matrix_world.translation.copy()
↪  # (x, y, z) in world space
```

yielding a vector $p(i) = (x(i), y(i), z(i)) \in \mathbb{R}^3$.

Displacement annotations. For displacement labels, we consider a time interval $[t_{\text{start}}, t_{\text{end}}]$ or frame interval $[i_{\text{start}}, i_{\text{end}}]$. The scripts record the world-space locations

$$p_{\text{start}} = p(t_{\text{start}}) = p(i_{\text{start}}), \quad p_{\text{end}} = p(t_{\text{end}}) = p(i_{\text{end}}),$$

and form the displacement vector

$$\Delta p = p_{\text{end}} - p_{\text{start}} = (\Delta x, \Delta y, \Delta z).$$

From this vector we derive two scalar quantities.

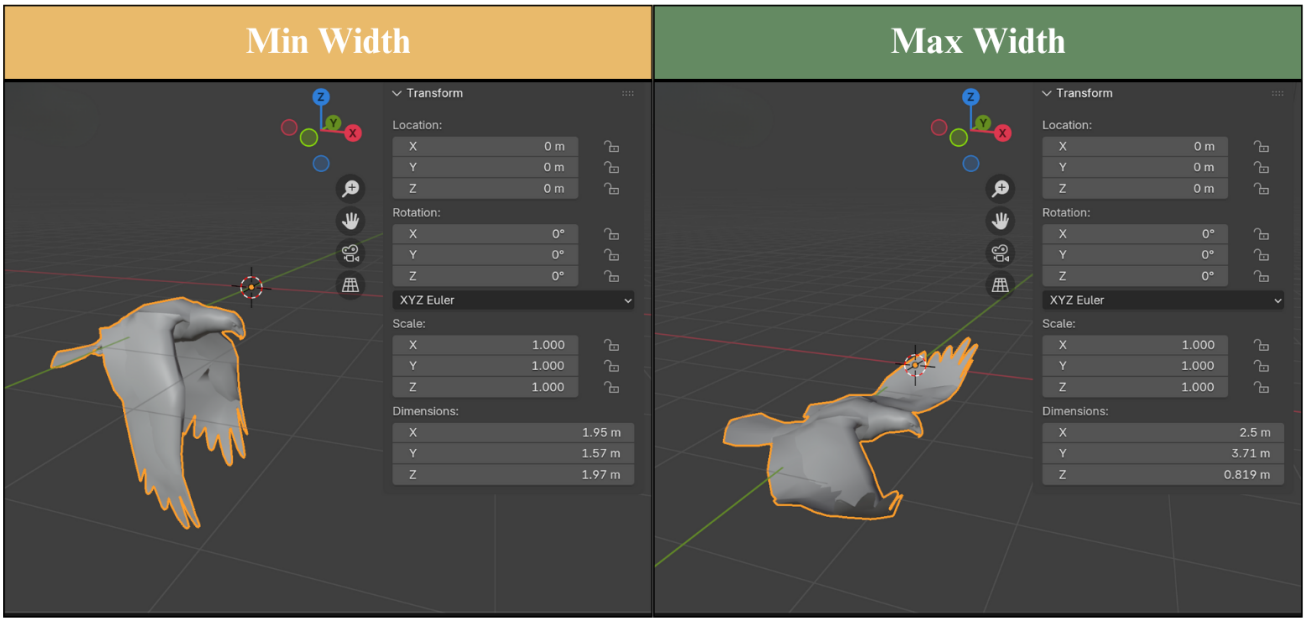


Figure 34. **Examples of flying animal measuring.**

1. The full 3D displacement

$$D_{3D} = \|\Delta p\|_2 = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2},$$

2. The planar displacement in a 2D plane, obtained by projecting the motion onto that plane. For any chosen 2D coordinate plane with axes (u, v), we define the planar displacement as

$$D_{2D} = \sqrt{(\Delta u)^2 + (\Delta v)^2}.$$

In our benchmark we primarily use the three canonical planes: the horizontal ground plane

$$D_{XY} = \sqrt{(\Delta x)^2 + (\Delta y)^2},$$

and the two vertical planes

$$D_{XZ} = \sqrt{(\Delta x)^2 + (\Delta z)^2}, \qquad D_{YZ} = \sqrt{(\Delta y)^2 + (\Delta z)^2}.$$

A compact version of the core computation is:

```python
def displacement_world(obj, frame_start, frame_end, scene):
    # sample world-space positions at start and end
    loc_start = get_world_location_at_frame(obj, frame_start, scene)
    loc_end   = get_world_location_at_frame(obj, frame_end, scene)

    # displacement vector
    disp_vec = loc_end - loc_start
    dx, dy, dz = disp_vec.x, disp_vec.y, disp_vec.z

    # 3D displacement and horizontal (XY) displacement
    D_3D = disp_vec.length                    # sqrt(dx*dx + dy*dy + dz*dz)
    D_XY = (dx**2 + dy**2) ** 0.5             # sqrt(dx*dx + dy*dy)

    return {
        "frame_start": frame_start,
        "frame_end": frame_end,
        "loc_start":  loc_start,
        "loc_end":    loc_end,
        "dx": dx, "dy": dy, "dz": dz,
        "D_3D": D_3D,
        "D_XY": D_XY,
    }
```

For each annotated interval, we store:

- the frame indices $(i_{\text{start}}, i_{\text{end}})$;
- the corresponding times $(t_{\text{start}}, t_{\text{end}})$, computed relative to the render start frame as above;
- the start and end world-space coordinates $p_{\text{start}}, p_{\text{end}}$;
- the displacement components $(\Delta x, \Delta y, \Delta z)$;
- the two scalar values D_{3D} and D_{XY} in Blender units.

In the question text, we explicitly state whether "displacement" refers to the full 3D displacement D_{3D} or to a planar displacement D_{2D} in a specified plane (e.g., horizontal D_{XY} or vertical D_{XZ} / D_{YZ}). The posterior ground-truth answer for that question is taken from the corresponding stored value.

D.1.3. Velocity and Acceleration.

Velocity measurement and uniform-speed diagnostics. Using the time–frame convention introduced above, let $p_i \in \mathbb{R}^3$ denote the world-space origin of the target object at frame i, obtained by querying `obj.matrix_world.translation` after calling `scene.frame_set(i)`. For two consecutive frames $i-1$ and i with timestamps t_{i-1} and t_i, we define the temporal spacing

$$\Delta t_i = t_i - t_{i-1},$$

and the instantaneous 3D velocity and scalar speed at frame i as

$$\mathbf{v}_i = \frac{p_i - p_{i-1}}{\Delta t_i}, \qquad s_i = \|\mathbf{v}_i\|_2,$$

where $\mathbf{v}_i = (v_{x,i}, v_{y,i}, v_{z,i})$ is the full 3D velocity vector in world coordinates and

$$s_i = \|\mathbf{v}_i\|_2 = \sqrt{v_{x,i}^2 + v_{y,i}^2 + v_{z,i}^2}$$

is its Euclidean norm. All operations are performed in Blender world space and use the true frame rate, so that positions p_i, velocities $\mathbf{v}_i$, and speeds s_i are expressed in a self-consistent system of physical units determined by the underlying scene scale.

The Blender analysis script computes these quantities frame by frame and writes them into a text block in Blender's Text Editor:

```
prev_loc   = None
prev_frame = None

for frame in range(frame_start, frame_end + 1):
    scene.frame_set(frame)
    loc = obj.matrix_world.translation.copy()   # p_i

    if prev_loc is None:
        vel_vec = Vector((0.0, 0.0, 0.0))
        speed   = 0.0
    else:
        dt      = (frame - prev_frame) / fps   # delta t_i
        disp    = loc - prev_loc   # p_i - p_{i-1}
        vel_vec = disp / dt   # v_i
        speed   = vel_vec.length   # s_i

    write_per_frame_entry(frame, loc, vel_vec, speed)
    prev_loc   = loc
    prev_frame = frame
```

For each frame i in the chosen range, this loop automatically prints one line of kinematic data into the text block, containing:

- the frame index i and its timestamp t_i (relative to the render start frame);
- the world-space position p_i;
- the full 3D velocity vector $\mathbf{v}_i$;
- the scalar speed s_i.

Any question that refers to "the speed at t seconds" or "the velocity at frame i" is answered by mapping the queried time to a frame index using the same time–frame conversion and then reading off the corresponding per-frame entry from this table, without any manual measurement or additional approximation.

In addition to per-frame velocities, we use the sequence of speeds $\{s_i\}$ to characterize simple motion regimes at the clip level. Let I_v be the index set of frames for which a speed value is defined (all frames except the very first one), and let $N_v = |I_v|$. We compute the mean speed

$$\bar{s} = \frac{1}{N_v} \sum_{i \in I_v} s_i$$

and the maximum relative deviation from this mean,

$$\delta_{\text{speed}} = \begin{cases} 0, & \text{if } \bar{s} = 0, \\ \max\limits_{i \in I_v} \dfrac{|s_i - \bar{s}|}{\bar{s}}, & \text{otherwise.} \end{cases}$$

Given a user-specified tolerance τ_v (parameter `UNIFORM_SPEED_TOLERANCE`, e.g., $\tau_v = 0.01$), we classify the clip as:

- **no effective motion** if $\bar{s} \approx 0$ (numerically $\bar{s} = 0$, indicating that the object is essentially stationary);
- **approximately uniform speed** if $\bar{s} > 0$ and $\delta_{\text{speed}} \le \tau_v$;
- **non-uniform speed** otherwise (speed fluctuations exceed the tolerance).

This logic matches the following code fragment:

```
avg_speed   = sum(speeds) / len(speeds)
max_dev     = max(abs(s - avg_speed) for s in speeds)
max_rel_dev = max_dev / avg_speed if avg_speed != 0 else 0.0

if avg_speed == 0:
    # almost no motion
    ...
else:
    if max_rel_dev <= UNIFORM_SPEED_TOLERANCE:
        # approximately uniform speed
        ...
    else:
        # not uniform-speed
        ...
```

Acceleration measurement and acceleration diagnostics. Starting from the third frame, we derive a scalar acceleration sequence that describes how quickly the object's speed

changes over time. Given the per-frame speeds $\{s_i\}$ defined above, and the same temporal spacings $\Delta t_i = t_i - t_{i-1}$, we define for all frames $i \geq 2$

$$a_i = \frac{s_i - s_{i-1}}{\Delta t_i},$$

so that a_i measures the finite-difference rate of change of speed between frames $i-1$ and i. The script computes these values alongside the speeds:

```
prev_speed = None
prev_frame = None

for frame in range(frame_start, frame_end + 1):
    scene.frame_set(frame)
    loc = obj.matrix_world.translation.copy()

    if prev_frame is None:
        dt    = 0.0
        speed = 0.0
        accel = None
    else:
        dt      = (frame - prev_frame) / fps
        disp    = loc - prev_loc
        vel_vec = disp / dt
        speed   = vel_vec.length

        if prev_speed is None:
            accel = None
        else:
            accel = (speed - prev_speed) / dt    # a_i

    write_per_frame_entry(frame, speed, accel)
    prev_loc   = loc
    prev_speed = speed
    prev_frame = frame
```

For each frame i in the valid range, this produces a scalar acceleration value a_i (undefined at the first two frames), which we store in the same text table. Any question that refers to "the acceleration at frame i" or to "the acceleration over a given interval" is answered by mapping the queried time to a frame index and reading off the corresponding a_i entry.

To summarize acceleration behaviour over the entire clip, we perform an analogous diagnostic analysis on the set of defined accelerations. Let I_a be the index set of frames for which a_i is defined (starting from the third frame), and let $N_a = |I_a|$. We compute the mean acceleration

$$\bar{a} = \frac{1}{N_a} \sum_{i \in I_a} a_i$$

and the maximum absolute deviation from this mean,

$$\Delta_a^{\max} = \max_{i \in I_a} |a_i - \bar{a}|.$$

If $|\bar{a}|$ is larger than a small absolute threshold $\varepsilon_{\min}$ (parameter `MIN_ABS_ACCEL`), we also measure the maximum relative deviation

$$\delta_{\text{accel}} = \begin{cases} 0, & \text{if } |\bar{a}| \leq \varepsilon_{\min}, \\ \dfrac{\Delta_a^{\max}}{|\bar{a}|}, & \text{otherwise.} \end{cases}$$

Given a user-specified tolerance τ_a (parameter `UNIFORM_ACCEL_TOLERANCE`), the script then classifies the acceleration regime as:

- **near-zero acceleration** if $|\bar{a}| \leq \varepsilon_{\min}$; in this case the overall acceleration is negligible and the clip is better interpreted as approximately constant-speed (or almost static);
- **approximately uniformly accelerated** if $|\bar{a}| > \varepsilon_{\min}$ and $\delta_{\text{accel}} \leq \tau_a$; the sign of $\bar{a}$ further distinguishes *uniform acceleration* ($\bar{a} > 0$) from *uniform deceleration* ($\bar{a} < 0$);
- **irregular acceleration** otherwise, meaning that the accelerations fluctuate significantly around their mean.

This classification logic is implemented as:

```
avg_accel   = sum(accels) / len(accels)
max_dev_a   = max(abs(a - avg_accel) for a in
↪  accels)
if abs(avg_accel) > MIN_ABS_ACCEL:
    max_rel_dev_a = max_dev_a / abs(avg_accel)
else:
    max_rel_dev_a = 0.0

if abs(avg_accel) <= MIN_ABS_ACCEL:
    # acceleration is very small -> closer to
    ↪  uniform speed / almost static
    ...
else:
    if max_rel_dev_a <= UNIFORM_ACCEL_TOLERANCE:
        if avg_accel > 0:
            # approximately uniformly
            ↪  accelerating
            ...
        elif avg_accel < 0:
            # approximately uniformly
            ↪  decelerating
            ...
        else:
            # numerically close to zero
            ...
    else:
        # not uniformly accelerated
        ...
```

These acceleration diagnostics never modify the underlying frame-level values $\{a_i\}$, just as the velocity diagnostics never modify $\{s_i\}$ or $\{\mathbf{v}_i\}$. Instead, they provide a principled, threshold-based way to tag each clip as approximately constant-speed, uniformly accelerated, uniformly decelerated, or irregular, allowing us to phrase velocity- and acceleration-related questions at a level of precision that matches the actual motion regime present in each clip.

D.1.4. Depth and Distance

All geometric quantities used in our benchmark are computed directly inside Blender in world coordinates via a Python script that evaluates the scene frame by frame. The script reads the world-space transforms of selected entities, computes Euclidean distances, and logs a per-frame table into a Blender Text Editor text block. From this table, we derive two kinds of quantities: (i) depth metadata between objects and the camera, and (ii) inter-object distances in 3D and in projected 2D planes.

Depth metadata (object–camera). We extract depth values in Blender only as auxiliary geometric metadata, not appearing in the inference question text or supervised targets. For any object o with world-space position

$$\mathbf{p}_t(o) = (x_t^o, y_t^o, z_t^o) \in \mathbb{R}^3$$

and the active camera object with world-space origin

$$\mathbf{p}_t^{\text{cam}} = (x_t^{\text{cam}}, y_t^{\text{cam}}, z_t^{\text{cam}}) \in \mathbb{R}^3,$$

at time t, we define the depth as the 3D Euclidean distance

$$\begin{aligned} d_t^{\text{depth}}(o) &= \left\|\mathbf{p}_t(o) - \mathbf{p}_t^{\text{cam}}\right\|_2 \\ &= \sqrt{(x_t^o - x_t^{\text{cam}})^2 + (y_t^o - y_t^{\text{cam}})^2 + (z_t^o - z_t^{\text{cam}})^2}. \end{aligned}$$

During annotation, we treat $d_t^{\text{depth}}(o)$ as a time series and often materialize it at a small set of shared reference time points (e.g., $t_1 = 1$s, $t_2 = 2$s). Thus, for a moving object we record pairs such as

$$\big(d_{t_1}^{\text{depth}}(o),\, d_{t_2}^{\text{depth}}(o)\big),$$

and we use the same time points for other relevant entities in the scene (e.g., additional moving objects or candidate target objects). This temporal alignment ensures that when we expose depth values as priors, they are always comparable across objects at exactly the same timestamps. For static objects, the depth is constant over the clip, so the values at different time points are trivially identical.

Crucially, we never use these depth values as evaluation targets. We do not ask questions of the form "What is the distance between object o and the camera at time t?", and we do not treat $d_t^{\text{depth}}(o)$ as ground truth in any task. Instead, depth is used only as internal geometric metadata and, in some infer questions, as numeric priors that can help a VLM reason about the relative 3D configuration of the scene. Even in those infer questions, we do not necessarily expose the camera–target depth of the queried object itself; rather, we expose a subset of aligned depth values for selected objects. All exposed depth values are manually reviewed to ensure that, together with the visual information, they are logically sufficient to infer the correct answer.

3D inter-object distances. We use the same Blender script to annotate 3D distances between pairs of entities. In our benchmark, these entities are either (i) two moving objects, or (ii) a moving object and a static reference object in the scene. We do not treat object–camera distances as inter-object ground truth; camera-related distances only appear as depth metadata as described above.

Let $\mathbf{p}_t(a) = (x_t^a, y_t^a, z_t^a)$ and $\mathbf{p}_t(b) = (x_t^b, y_t^b, z_t^b)$ denote the world-space positions of entities a and b at time t. The 3D object–object distance is

$$\begin{aligned} d_t^{\text{3D}}(a,b) &= \left\|\mathbf{p}_t(a) - \mathbf{p}_t(b)\right\|_2 \\ &= \sqrt{(x_t^a - x_t^b)^2 + (y_t^a - y_t^b)^2 + (z_t^a - z_t^b)^2}. \end{aligned}$$

For objects whose dimensions do not change over time, we take $\mathbf{p}_t(o)$to be the Blender object origin in world coordinates. For articulated skeletal models (e.g., humans), transformations are defined on a set of bones rather than a single rigid body. In these cases, we first enumerate all bones in the armature and select a semantically stable reference joint (for humans, typically the pelvis/hip bone) as the anchor. All distances involving that character are then defined using the world-space position of this reference bone, which provides a temporally stable and semantically meaningful notion of "where the character is". For some non-humanoid articulated assets or composite rigs where no single point or bone has a clear semantic interpretation as the "character center" (e.g., certain vehicles or multi-part machines), we instead introduce an auxiliary helper object. Concretely, we create an Empty object h rigidly parented to the rig root, snap its origin to the root in world space, and log its world-space position $\mathbf{p}_t(h)$ at each frame. In those scenes, all distances involving the articulated asset are computed with respect to $\mathbf{p}_t(h)$ rather than a specific bone. This dual strategy (bone-based anchors for humanoid characters and helper-object anchors for other articulated assets) keeps the distance annotations consistent while accommodating the diversity of rig structures in our Blender scenes.

Planar (2D) distances. Some tasks explicitly constrain distance reasoning to a 2D projection, for example "horizontal distance in the ground plane" or "distance in the vertical cross-section". To support such tasks, we also annotate *planar distances* by projecting 3D positions onto a chosen coordinate plane $\Pi \in \{\text{XY}, \text{XZ}, \text{YZ}\}$. We define the projections

$$\begin{aligned} \Pi_{\text{XY}}(x,y,z) &= (x,y), \\ \Pi_{\text{XZ}}(x,y,z) &= (x,z), \\ \Pi_{\text{YZ}}(x,y,z) &= (y,z). \end{aligned}$$

The planar distance between a and b at time t is then

$$d_t^{\text{plane}}(a,b) = \left\|\Pi\big(\mathbf{p}_t(a)\big) - \Pi\big(\mathbf{p}_t(b)\big)\right\|_2.$$

Whether a distance question is treated as 2D or 3D in our benchmark is fully determined by (i) the infer question

and (ii) the geometric category of the underlying video (2D vs. 3D sequence). Each video is assigned to a 2D or 3D split based on how it is generated and how its geometry is intended to be interpreted.

For videos in the 2D split, distances are interpreted in a single plane by construction. If a question in such a video simply asks for "the distance between objects a and b", the ground-truth target is the planar distance $d_t^{\text{plane}}(a, b)$, and the task is labeled as 2D.

For videos in the 3D split, we have full world-space geometry. If the infer question explicitly restricts reasoning to a plane (e.g., "horizontal distance" or "vertical distance"), we again use the planar distance $d_t^{\text{plane}}(a, b)$ and label the task as 2D. Otherwise, distance questions on 3D videos default to the full 3D Euclidean distance $d_t(a, b)$ defined above, and these tasks are labeled as 3D.

Blender implementation and per-frame logging. All depth and distance quantities above are computed by a unified Blender Python script. For a specified frame range $[f_{\min}, f_{\max}]$, the script iterates over frames, queries world-space positions, computes distances, and writes a formatted table into a Blender Text Editor text block. The core of the script is:

```python
import bpy
from mathutils import Vector

scene = bpy.context.scene
fps   = scene.render.fps

def world_pos(obj_or_bone):
    """Return world-space translation of an object or a specific bone."""
    # For regular objects:
    if hasattr(obj_or_bone, "matrix_world"):
        return obj_or_bone.matrix_world.to_translation()
    # For pose bones (e.g., armature.pose.bones["Hips"]):
    return obj_or_bone.matrix.to_translation()

for f in range(f_min, f_max + 1):
    scene.frame_set(f)

    # Example: distance between entities A and B
    p_a = world_pos(entity_a)
    p_b = world_pos(entity_b)

    # Full 3D distance in world coordinates
    dist_3d = (p_a - p_b).length

    # Example planar distance in the XY plane (horizontal separation):
    # other planes (XZ, YZ) are obtained analogously in the full script.
    p_a_xy = Vector((p_a.x, p_a.y, 0.0))
    p_b_xy = Vector((p_b.x, p_b.y, 0.0))
    dist_xy = (p_a_xy - p_b_xy).length

    # Timestamp (seconds), assuming t = 0 at f_min
    t = (f - f_min) / fps

    # Log one line (frame, time, 3D distance, XY-plane distance, ...)
    write_to_text_block(f, t, dist_3d, dist_xy)
```

The same script is run with different choices of `entity_a` and `entity_b` to (i) record object–camera distances as depth metadata and (ii) generate object–object distance annotations. In the code listing above, we show the XY-plane case for concreteness. In practice, this block is used as a template: when planar distances in other coordinate planes (XZ or YZ) are needed, we simply modify the projection lines accordingly (e.g.,`Vector((p_a.x, p_a.z, 0.0))` for XZ or `Vector((0.0, p_a.y, p_a.z))` for YZ) and rerun the script for that scene.

All measurements are derived directly from Blender's world-space transforms, so the numeric values used in our questions and annotations match the underlying scene geometry exactly. Depth values are stored only as auxiliary geometric metadata or optional numeric priors, whereas inter-object distances serve as the ground truth posterior.

D.2. Lab Data Annotation

Unlike Blender-based simulations, lab captures do not provide annotations that can be directly read out from software. Nevertheless, with our multi-stereo camera setup, we are able to reconstruct both the 3D scene and the 3D geometry of the target objects.

To obtain annotations for lab captures, we first attempted to use off-the-shelf 3D reconstruction models such as FoundationPose [45] to assist with annotation. However, in our experiments, we found that even state-of-the-art AI models struggled to localize objects stably in world coordinates and frequently failed under occlusions.

Therefore, we instead rely on traditional geometry-based reconstruction with calibrated camera poses. We use the main camera together with manually selected metric depth values to recover the objects' world coordinates. Concretely, we use the UI-based annotation tool introduced earlier (see Figure 26) to manually click on the center of the target object across a sequence of frames in the video. The tool records the 2D coordinates of each click in image space together with the corresponding metric depth from the main camera. Using the calibrated camera intrinsics, we then back-project the annotated object centers into the world coordinate system.

To obtain static priors such as object size, we manually measure the shape and dimensions of each object to reduce measurement error. To obtain dynamic priors such as object velocities and accelerations, we annotate the object center across at least 5 adjacent frames (for smoothing purpose) and compute the instantaneous velocity and acceleration magnitudes from the resulting sequence of world

coordinates, assuming a constant frame rate. Formally, the computation of dynamic priors in lab captured videos are:

$$v_k^{\text{world}} \approx \frac{x_{k+1}^{\text{world}} - x_k^{\text{world}}}{\Delta t},$$
$$a_k^{\text{world}} \approx \frac{x_{k+1}^{\text{world}} - 2x_k^{\text{world}} + x_{k-1}^{\text{world}}}{\Delta t^2},$$

where Δt is determined from the frame rate of the captured videos.

Note that, based on our predefined questions and the priors required at specific time stamps, not all frames of the lab videos need to be annotated. This greatly reduces the annotation workload and helps prevent error accumulation over time.

When the object is occluded in the main camera view, we instead estimate its world coordinates by averaging the positions obtained by transforming its 2D annotations from the other three cameras into the world coordinate system.

D.3. Internet Data Annotation

For internet videos, we must derive kinematic ground truth from raw pixels rather than from simulator logs or calibrated sensors. To keep the process systematic, we adopt a three-stage annotation workflow: (i) metadata and task specification, (ii) pixel-level measurement with a custom video tool, and (iii) conversion from pixel-space kinematics to real-world quantities.

Metadata and task specification. Each collected internet clip is first assigned a unique identifier (e.g., `internet_0001`). Based on the scene content, annotators determine the physical prior available in the video (e.g., gravity $g = 9.8\,\mathrm{m/s^2}$, the length of a credit card, lane width on a road, or conveyor-belt speed), and label the corresponding **`video_type`** code (e.g., `S2MC`, `A2SS`) as described in subsection B.2. We then identify all feasible inference targets in the clip (e.g., vehicle speed, ball acceleration, object size) and enumerate kinematic inference questions that can be solved from the chosen prior. For each question, we annotate the **`inference_type`** (static vs. dynamic posterior) and write the final textual question so that the requested quantity, unit, and time reference (instantaneous vs. average) are explicit.

Pixel-level measurement tool. To read off pixel-space trajectories, we build a small annotation tool in Python using OpenCV. The script takes as input the path to a video file and interactively queries the annotator for the correct frame rate. It first shows the FPS detected by OpenCV, then allows the annotator to either accept it or manually enter a more reliable value (e.g., 24/25/30/60 fps). The tool verifies or manually counts the total frame number to avoid metadata errors, and reports basic properties such as resolution and duration.

After loading, the script launches an interactive player (Figure 35). Each frame is overlaid with its index and timestamp, `Frame: k/N | Time: t_k s`, and a play/pause status indicator. A trackbar at the bottom allows direct jumping to any frame; dragging the slider or entering a frame index updates the display immediately. Annotators can also step frame-by-frame with the left/right arrow keys or play the sequence in real time with the space bar. In addition to these controls, we use a standard OpenCV mouse callback to display the current pixel coordinates (x, y) of the cursor, which facilitates precise measurement of distances and object positions.

(3) Measuring pixel-space kinematics. Given the verified frame rate f (in fps), we set the time step as $\Delta t = 1/f$. Annotators use the player to locate the relevant frames for each question: for instantaneous quantities we jump to the frame closest to the target timestamp; for average quantities we select a sequence of frames spanning the interval of interest. Along the main motion direction, we record (i) the object's pixel length $\mathbf{S}^{\text{pixel}}$ (e.g., bumper-to-bumper for a car), and (ii) the pixel coordinate x_k^{pixel} of the object's reference point at discrete times $t_k = k\Delta t$.

From these measurements we compute pixel-space kinematics using finite differences. For a 1D trajectory $\{x_k^{\text{pixel}}\}_{k=0}^{K}$, the (approximate) velocity and acceleration at time t_k are

$$\mathbf{V}_k^{\text{pixel}} \approx \frac{x_{k+1}^{\text{pixel}} - x_k^{\text{pixel}}}{\Delta t},$$
$$\mathbf{A}_k^{\text{pixel}} \approx \frac{x_{k+1}^{\text{pixel}} - 2x_k^{\text{pixel}} + x_{k-1}^{\text{pixel}}}{\Delta t^2}.$$

When motion is measured in two image-plane directions, we apply the same formulas component-wise to $(x_k^{\text{pixel}}, y_k^{\text{pixel}})$. In this way we obtain, for every prior object and inference target, the relevant pixel-space size S^{pixel}, velocity $\mathbf{V}_k^{\text{pixel}}$, and acceleration $\mathbf{A}_k^{\text{pixel}}$.

(4) Converting to real-world kinematics. Assuming planar motion (subsection B.1), a single scalar scale factor $\gamma > 0$ with units [world length / pixel] suffices along the motion direction. Depending on which physical prior is available for a given clip, we estimate γ via

$$\gamma = \begin{cases} \dfrac{\mathbf{S}^{\text{world}}}{\mathbf{S}^{\text{pixel}}}, & \text{if size prior is given} \\ \dfrac{|\mathbf{V}_{t_0}^{\text{world}}|}{|\mathbf{V}_{t_0}^{\text{pixel}}|}, & \text{if velocity prior is given} \\ \dfrac{|\mathbf{A}_{t_0}^{\text{world}}|}{|\mathbf{A}_{t_0}^{\text{pixel}}|}, & \text{if acceleration prior is given}\,. \end{cases}$$

where t_0 is the timestamp at which the prior is defined.

Once s is determined, any kinematic quantity of the inference target can be expressed in world units by a simple

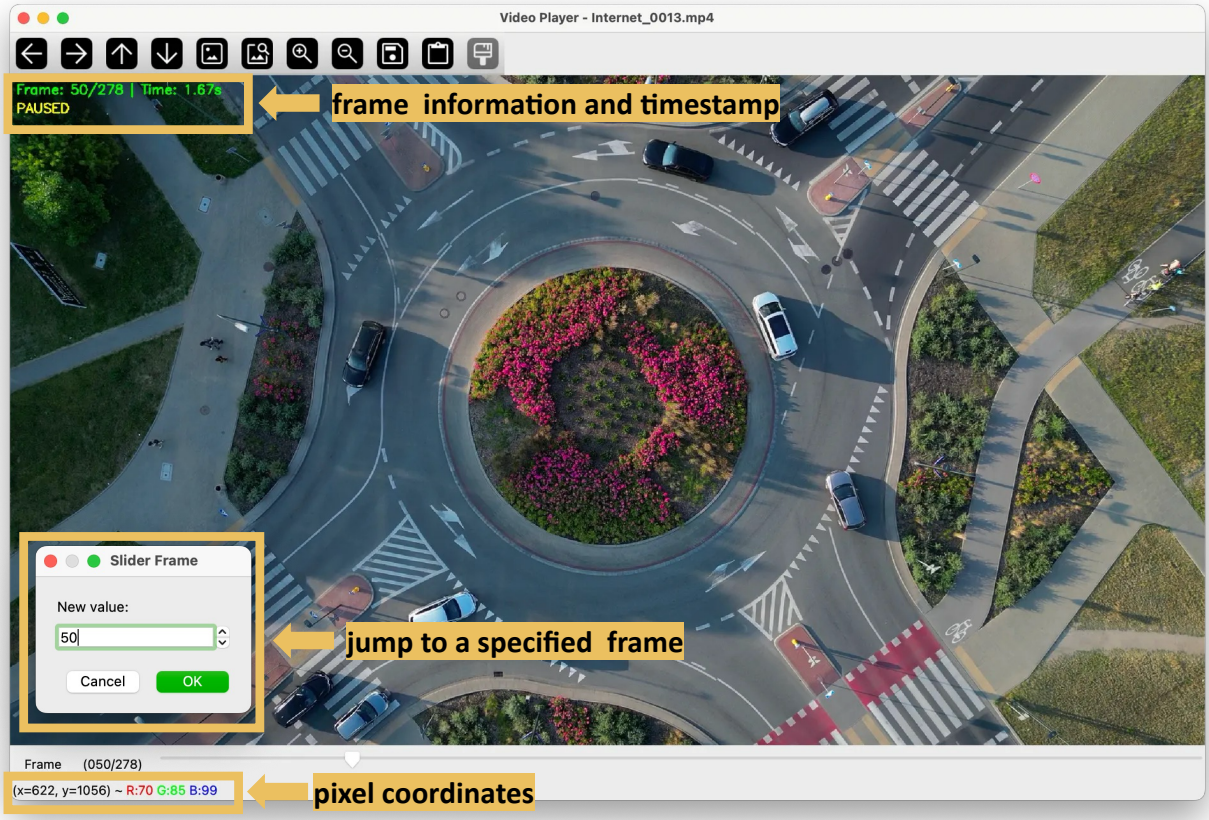


Figure 35. **Pixel-level measurement tool.**

rescaling:

$$(S_{\text{target}})^{\text{world}} = \gamma\,(S_{\text{target}})^{\text{pixel}},$$
$$(\mathbf{V}_{\text{target}})^{\text{world}}_{t_k} = \gamma\,(\mathbf{V}_{\text{target}})^{\text{pixel}}_{t_k},$$
$$(\mathbf{A}_{\text{target}})^{\text{world}}_{t_k} = \gamma\,(\mathbf{A}_{\text{target}})^{\text{pixel}}_{t_k}.$$

These world-space values are then used as the ground-truth priors and posteriors in QUANTIPHY.

D.4. Segmented Data

Segmenting out target objects from the videos only changes the background and does not alter their original physical properties. Therefore, we reuse the original annotations of the videos for the segmented data without any modification.

E. Vision-Language Models

We evaluate a diverse suite of 21 VLMs, spanning proprietary closed-source APIs, open-source models hosted via Replicate, and open-source models deployed by our own.

Proprietary Models. We select flagship models from four major providers, including standard multimodal capabilities and those utilizing inference-time reasoning.

- **OpenAI.** We evaluate two models: **GPT-5.1** [32] (`gpt-5.1-2025-11-13`) and **GPT-5** [31] (`gpt-5-2025-08-07`), both are flagship multimodal models. The latter is a "thinking model" that leverages Chain-of-Thought (CoT) processing for complex reasoning tasks. Both are accessed via the OpenAI API, accepting video inputs as sequences of base64-encoded frames to enable temporal understanding.
- **Google.** We evaluate **Gemini 2.5 Pro** [20] (`gemini-2.5-pro`), a high-capacity model capable of processing large context windows of interleaved images and text, and **Gemini 2.5 Flash** [19] (`gemini-2.5-flash`), a lightweight variant optimized for low-latency tasks. Both models accept base64 inline data and are optimized for complex reasoning across modalities.
- **Anthropic.** We utilize **Claude Sonnet 4.5** [4] (`claude-sonnet-4-5-20250929`). This model ac-

#	Question	GT Prior	GT Depth Info
1	What is the length of the slope in cm?	$t = 0.6$, ball acc. $= 4.6\text{m}/s^2$	$t = 0$s, dist. slope_far = 1.8540m, dist. slope_near = 1.7010m
2	What is the basketball's distance from the camera at 1.2s in meters?	acc. $= 9.8\text{m}/s^2$	$t = 1.0$s, dist. ball = 1.4010m, $t = 1.2$s, dist. ball = 1.5640m
3	What is the basketball's distance from the camera at 1.2s in meters?	acc. $= 9.8\text{m}/s^2$	$t = 1.0$s, dist. ball = 1.4010m, $t = 1.2$s, dist. ball = 1.5640m
4	What is the velocity of the ball at the end of the slope in cm/s?	$t = 1.4$, ball acc. $= 4.05\text{m}/s^2$	$t = 1.0$s, dist. ball = 1.7790m, dist. slope_far_end = 1.8150m
5	What is the velocity of the soccer ball at 1.5s in m/s?	gravity acc. $= 9.8\text{m}/s^2$	$t = 1.5$s, dist. soccer_ball = 1.4020m
6	What is the velocity of the tennis ball at 3.4s?	gravity acc. $= 9.8\text{m}/s^2$	dist. near_person = 2.002m, dist. far_person = 3.130m
7	What is the acceleration of the red ball at 0.2s in cm/s^2?	diameter of the red ball = 7cm	$t = 0.2$s, dist. book = 1.0995m, $t = 0.6$s, dist. book = 1.1510m
8	What is the shoulder breadth of the person in cm?	gravity acc. $= 9.8\text{m}/s^2$	
9	What is the basketball's diameter in cm?	acc. $= 9.8\text{m}/s^2$	$t = 1.0$s, dist. ball = 1.4010m, $t = 1.2$s, dist. ball = 1.5640m
10	What is the velocity of the toy at 2.6s in cm/s?	gravity acc. $= 9.8\text{m}/s^2$	
11	What is the velocity of the toy at 2.6s in cm/s?	gravity acc. $= 9.8\text{m}/s^2$	
12	What is the diameter of the tennis ball in cm?	gravity acc. $= 9.8\text{m}/s^2$	dist. near_person = 2.002m, dist. far_person = 3.130m
13	What is the length of the wood block in cm?	ruler calibre = 1cm	
14	What is the acceleration of the astronaut at time 1s in m/s^2?	height of the astronaut = 2.2m	$t = 1$s, dist. astronaut = 23.137m
15	What is the acceleration of the tennis ball at time 1.5s in m/s^2?	diameter of the tennis ball = 0.17m	
16	What is the diameter(width) of the plant in meters?	acc. of the mug cup before 0.5s $= 9.8\text{m}/s^2$	
17	What is the velocity of the ball at the end of the slope in cm/s?	$t = 1.4$, ball acc. $= 4.05\text{m}/s^2$	$t = 1.0$s, dist. ball = 1.7790m, dist. slope_far_end = 1.8150m
18	What is the diameter of the soccer ball in cm?	gravity acc. $= 9.8\text{m}/s^2$	dist. near_person = 2.002m, dist. far_person = 3.130m
19	What is the distance between two people in cm?	gravity acc. $= 9.8\text{m}/s^2$	$t = 0.60$s, dist. near_person = 1.5800m, dist. far_person = 3.2370m
20	What is the height of the person in cm?	gravity acc. $= 9.8\text{m}/s^2$	
21	What is the height of the toy in cm?	gravity acc. $= 9.8\text{m}/s^2$	
22	What is the width of the box in cm?	gravity acc. $= 9.8\text{m}/s^2$	dist. near_person = 2.002m, dist. far_person = 3.130m
23	What is the height of the yellow toy in cm?	gravity acc. $= 9.8\text{m}/s^2$	$t = 0.60$s, dist. jar = 1.9735m, dist. carrot = 2.6300m
24	What is the basketball's diameter in cm?	acc. $= 9.8\text{m}/s^2$	$t = 1.0$s, dist. ball = 1.4010m, $t = 1.2$s, dist. ball = 1.5640m
25	What is the height of the woman in yellow in meters?	walking velocity = 1.25m/s	
26	What is the length of the speedboat in meters?	sailboat length = 12m	
27	What is the length of the string in cm?	gravity acc. $= 9.8\text{m}/s^2$	$t = 0.3$s, dist. ball = 1.7060m, $t = 0.6$s, dist. ball = 1.3440m
28	What is the displacement of the rolling object from 0s to 0.6s in cm?	$t = 0.6$, ball acc. $= 4.6\text{m}/s^2$	$t = 0$s, dist. ball = 1.8180m, $t = 0.6$s, dist. ball = 1.7288m
29	What is the basketball's distance from the camera at 1.8s in meters?	acc. $= 9.8\text{m}/s^2$	$t = 0.0$s, dist. ball = 1.3210m, $t = 1.8$s, dist. ball = 1.5835m
30	What is the velocity of the tennis ball at 3.4s?	gravity acc. $= 9.8\text{m}/s^2$	dist. near_person = 2.002m, dist. far_person = 3.130m
31	What is that ball's acceleration at 0.6s in m/s?	gravity acc. $= 9.8\text{m}/s^2$	$t = 0.3$s, dist. ball = 1.7060m, $t = 0.6$s, dist. ball = 1.3440m
32	What is the width of the desk in cm?	$t = 1.5$, ball acc. $= 3.0\text{m}/s^2$	$t = 0$s, dist. cup = 1.2100m, dist. desk_left = 0.8690m
33	What is the length of the slope in cm?	$t = 1.5$, ball acc. $= 3.0\text{m}/s^2$	$t = 0$s, dist. slope_far = 1.3450m, dist. slope_near = 1.1040m
34	What is the length of the box in cm?	gravity acc. $= 9.8\text{m}/s^2$	dist. near_person = 2.002m, dist. far_person = 3.130m

Table 36. **Selected examples of text inputs and answers (part 1).** Question is the natural-language prompt presented to the VLM. GT Prior is the physical prior provided to the model. GT Depth Info is the depth annotation used for 3D reasoning tasks. GT Posterior is the numeric ground-truth answer to the kinematic inference question. See Table 3 for detailed explanation. Raw Response shows the corresponding VLM output for each model, and Parsed Value shows the parsed value extracted from the Raw Response.

#	Video Type	Inference Type	GT Posterior	Raw Response	Parsed Value	Model
1	A3MS	DS	30.8	Looking at the video frames, I can see a table … Length = 0.1530m = 15.30cm	15.30	Claude-4.5 Sonnet
2	A3SS	DS	1.565	1.5640m	1.564	Claude-4.5 Sonnet
3	A3SS	DS	1.565	[1.5640]	1.564	CogVLM2 Video
4	A3SS	DD	208.76	[velocity_at_the_end_of_the_slope] [unit]	None	CogVLM2 Video
5	A3SS	DD	2.7968	1.5m/s \n In the given video, the soccer ball is flying … at 1.5 m/s.	1.5	Fuyu-8B
6	A3MC	DD	542.82	2.002m/s \n In the image, … Therefore, its velocity is 2.002m/s.	2.002	Fuyu-8B
7	S3SC	SD	1086.09	0.0cm/s^2	0	Gemini-2.5 Flash
8	A2MC	DS	39	39.8cm	39.8	Gemini-2.5 Flash
9	A3SS	DS	23.2	23.9cm	23.9	Gemini-2.5 Pro
10	A2MS	DD	357	0cm/s	0	Gemini-2.5 Pro
11	A2MX	DD	357	−2549cm/s	2549	ChatGPT-5.1
12	A3MC	DS	6.7	6.5cm	6.5	ChatGPT-5.1
13	S2MC	SS	9.9	10cm	10	ChatGPT-5
14	S3SS	SD	3.604	0m/s^2	0	ChatGPT-5
15	S2SS	SD	6.129	9.8m/s^2	9.8	InternVL-3.5-30B
16	A2MC	DS	0.075	0.05m	0.05	InternVL-3.5-30B
17	A3SS	DD	208.76	$[4.05]\ [\text{m/s}^2]$	4.05	LLaVA-13B
18	A3MX	DS	17.5	[0.31] [0.28]	0.28	LLaVA-13B
19	A3MX	DS	325	[3.2370] [m]	3.237	LLaVA-13B
20	A2MC	DS	170	<think> So, let's try to … maybe the person's height is around 170cm.	170	MiniCPM-V 4.5-8B
21	A2MX	DS	14	< think > Okay, so I need to … If I can determine the time it	None	MiniCPM-V 4.5-8B
22	A3MX	DS	7.2	0.000000000000000000…	0	Molmo-7B
23	A3MS	DS	14	15.2cm	15.2	Molmo-7B
24	A3SS	DS	23.2	23.9cm < |end| >	23.9	Phi-3-Mini-128K-Instruct
25	V2MC	DS	1.71	Cannot be determined from given information.	None	Phi-3-Mini-128K-Instruct
26	S2MC	SS	4.8	[Value missing due to insufficient information]	None	Phi-3-Mini-128K-Instruct
27	A3SS	DS	30	34.02cm	34.02	Phi-4-Multimodal-Instruct
28	A3MC	DS	207.9	−22.8cm	22.8	Phi-4-Multimodal-Instruct
29	A3SC	DS	1.5835	1.5835m	1.5835	Qwen3-VL-Instruct-8B
30	A3MC	DD	542.82	0.0m/s	0	Qwen3-VL-Instruct-8B
31	A3SS	DS	4.939	0.000000000000000000…	0	VILA-7B
32	A3MX	DS	857.9	1.2100m	1.21	VILA-7B
33	A3MS	DS	30.8	13450	13450	SmolVLM-Instruct
34	A3MC	DS	19.5	1.50000000000000004	1.5	SmolVLM-Instruct

Table 37. **Selected examples of text inputs and answers (part 2).** Question is the natural-language prompt presented to the VLM. GT Prior is the physical prior provided to the model. GT Depth Info is the depth annotation used for 3D reasoning tasks. GT Posterior is the numeric ground-truth answer to the kinematic inference question. See Table 3 for detailed explanation. Raw Response shows the corresponding VLM output for each model, and Parsed Value shows the parsed value extracted from the Raw Response.

cepts a structured list of content blocks (text and base64-encoded images). It is characterized by its detailed explanatory capabilities, often providing extensive textual rationale alongside numerical answers. (See Figure 37 for raw response examples).

- **xAI.** We evaluate **Grok 4.1 (Fast Reasoning)** [47], a model that combines rapid inference with advanced reasoning capabilities, designed to handle complex multimodal tasks with reduced latency while maintaining high accuracy.

Open-source Models. We also evaluate 15 distinct open-source models, representing a spectrum of architectures, parameter sizes, and input modalities to systematically assess architectural variations and scaling effects.

- We include **LLaVA-13B** [28], a foundational baseline combining a Vicuna LLM with a CLIP encoder; **VILA-7B** [27], pre-trained on interleaved image-text data; and **Qwen3-VL-8B** [7], the latest iteration of Alibaba's vision-language series.
- To examine scaling effects across model sizes, we deploy and evaluate the **Qwen3-VL-Instruct** series at three scales: **2B**, **8B**, and **32B** parameters [7], enabling direct comparison of performance improvements with increased capacity within a single architecture family.
- Similarly, we assess the **InternVL-3.5** series across three parameter scales: **2B**, **8B**, and **30B** [13], providing additional insights into how architectural choices interact with model scale for vision understanding.
- We evaluate **Fuyu-8B** [10], which utilizes a simplified architecture processing raw image patches without a separate visual encoder.
- We test **Molmo-7B** [15] and **SmolVLM** [30] (1.6B), both designed for high-efficiency reasoning on consumer hardware.
- **Phi-4 Multimodal** [2] and the smaller **Phi-3-Mini-128K-Instruct** [1], both are Microsoft suite and process text interleaved with image lists.
- Finally, to assess temporal analysis capabilities, we include **MiniCPM-V 4.5** [58] and **CogVLM2-Video** [23]. Distinct from frame-based approaches, these models accept direct video file inputs for native temporal processing.

In total, our evaluation encompasses **21 models** (6 proprietary and 15 open-source), providing comprehensive coverage of the current state-of-the-art in multimodal reasoning across different scales, architectures, and deployment paradigms.

F. Prompt Design

For quantitative evaluation, we use a constrained generation strategy designed for precise numerical outputs. The system prompt explicitly restricts the output space, instructing the model to "provide ONLY the numerical answer with units" and emphasizing "No explanation or reasoning needed." The prompt structure includes the visual input, system instructions, and ground truth priors and/or depth info, followed by the specific query and a post-prompt reinforcement. (See Figure 6 for the Text Prompt template).

To ensure reproducibility, we enforce deterministic generation where possible. We set `temperature=0` (greedy decoding) for all models supporting this parameter. For hosted models via Replicate where explicit temperature control is unavailable, we utilize the default inference parameters recommended by the model maintainers.

The input sequence for all models is programmatically structured as `[Video Frames][System Prompt][Ground Truth Prior and/or Depth Info][Question][Post-prompt]`. Detailed examples are provided in Figure 36.

- `[Video Frames]`. This segment contains the full sequence of frames extracted from the source video, base64-encoded. We normalize all videos to 480p resolution. Our preliminary exploration indicated that while lower spatial resolution (480p) has negligible impact on physics reasoning, temporal subsampling (dropping frames) significantly degrades the tracking of velocity and acceleration. Therefore, we prioritize temporal fidelity by retaining all frames from the source video. This approach also distinguishes our methodology from most prior works, which often prioritize spatial resolution at the expense of frame rate.
- `[System Prompt]`. This serves as the behavioral instruction, establishing the persona that "You are an expert video analyst..." We selected this formulation following a pilot study of five prompt variations on a subset of 15 videos, which revealed no significant difference and this persona yielded the relatively highest adherence to formatting constraints without altering reasoning accuracy.
- `[Ground Truth Prior and/or Depth Info]`. This includes physical constants and contextual priors (e.g., "Given that acceleration $a = 9.8m/s^2$"). For `3D` scenes, this also includes depth information (e.g., "At $t = 1.0s$, the distance of the ball to the camera is $1.779m$..") for estimation.
- `[Question]`. The specific physics query (e.g., "What is the total displacement in the y-axis?").
- `[Post-prompt]`. A final instruction reinforcing the output format (e.g., "Provide... ONLY the numerical answer with units") to mitigate the tendency of models to generate verbose "Chain-of-Thought" explanations in the final output.

G. Answer Retrieval and Parsing

For each question, we query the model once. To ensure robustness against API instability or transient errors, we

implement an automated retry mechanism: if a query fails (e.g., timeout or server error) or yields a non-parsable response, the request is re-submitted up to a maximum of five times.

To extract quantitative data from potentially noisy model outputs, we employ a hierarchical parsing function (`parse_number`) designed to handle both concise and verbose responses. The logic proceeds as follows.

1. **Exact Match Validation:** We first check if the raw response strictly matches a numerical format (with or without units). If the response is concise (containing only a value and a unit matching the requested physical quantity), the numerical value is extracted directly.
2. **Delimiter Search:** If the response is verbose, we scan for explicit answer markers, including `` `=" ``, `` `Final Answer: " ``, `` `Answer: " ``, `` `=> " ``, and `` `: " ``. If a delimiter is identified, we discard the preceding text and retain only the substring immediately following it. If multiple cases are shown, we take the last occurrence.
3. **Unit Sanitization:** The retained text is cleaned of common physical units (e.g., "meters", "m/s", "kg") to prevent string processing errors during numerical extraction.
4. **Heuristic Extraction:** Finally, a regular expression is applied to the cleaned text to identify floating-point numbers. We also take absolute value of the numerical answer. Crucially, if multiple numbers are found, we extract the last valid number in the sequence. This heuristic assumes that in verbose Chain-of-Thought responses, the final conclusion is located at the end of the text.
5. **Failure Handling:** If no valid number is identified after these steps, the response is recorded as a failure (parsed as `None`).

This strict parsing pipeline is essential because our evaluation metric relies on exact numerical regression. If a model provides only a qualitative description (e.g., "The ball is moving quickly", "Cannot be determined", or API error) or fails to reach a numerical conclusion within the decoding limit, it is then treated as a failure.

While most models adhered to the format constraints (i.e., [number] [unit]), we observed that **Claude Sonnet 4.5**, **Fuyu-8B**, and **MiniCPM-V 4.5** frequently generated verbose, multi-sentence explanations despite instructions to the contrary. By targeting the post-delimiter text and the last numerical value, our parsing logic effectively retrieves the correct answer from these verbose outputs. Representative examples of raw responses and their parsed values are detailed in Figure 37.

H. Human Study Details.

To contextualize model performance and establish an empirical upper bound on human quantitative physical reasoning, we conducted a survey study using the Gorilla Experiment Builder platform. The platform supported video presentation, question randomization, and structured data collection. Below, we describe the participant cohort, task construction, interface design, evaluation methodology, and resulting performance trends.

H.1. Participants

Participants were recruited from a mix of undergraduate, graduate, and PhD researchers, including individuals with advanced training in various fields including engineering, physics, and mathematics. This allowed us to approximate both typical human performance and an expert-level upper bound. Participants were informed that they could freely choose their reasoning strategy—including intuitive estimation, visual approximation, or explicit calculation—in order to reflect natural human reasoning rather than enforce a prescribed computation protocol.

Our survey is divided into two versions: a `2D` survey and a `3D` survey. Both the `2D` and `3D` surveys were completed by multiple participants, including several with substantial technical backgrounds. A subset of participants completed both tasks, enabling direct cross-dimensional comparisons of individual consistency.

H.2. Task Construction and Experimental Design

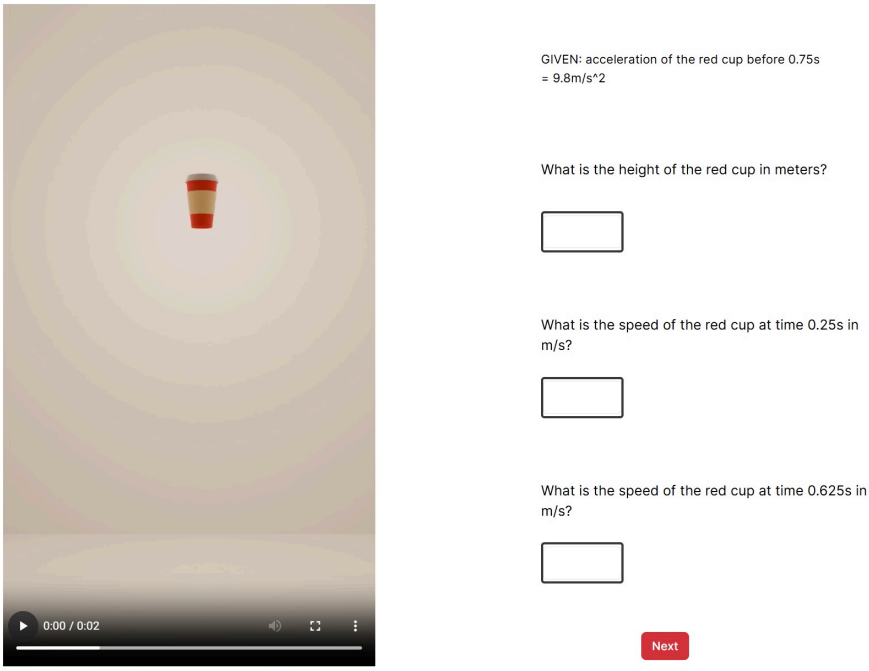
Following the 36 fine-grained video categories defined in subsection B.2, we organized the full dataset into these category units for both model and human evaluation. Each category contains a small pool of representative videos, from which stimuli were sampled during the experiment.

Participants were randomly assigned to either the `2D` survey or the `3D` survey. Each participant viewed 18 videos, one sampled per category within their assigned dimensionality. Each video was followed by 1–3 quantitative kinematic questions, with the same priors and task formulations used in our VLM evaluation (e.g., estimating acceleration magnitude, inferring relative size changes, or recovering object velocity).

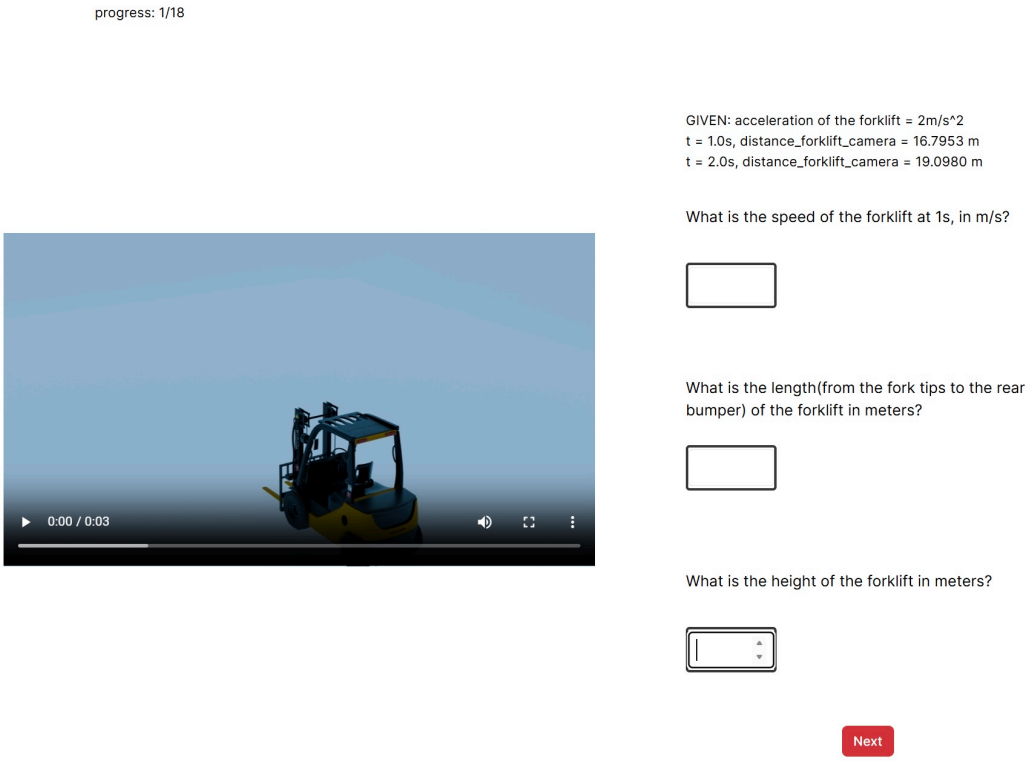
The UI interface was intentionally designed to be concise and unobtrusive. For each trial, participants were presented with:

- A video with standard playback controls;
- The physical prior ground truth (`A/S/V`) and/or corresponding depth information for `3D` videos;
- The quantitative question texts;
- Numeric input boxes (numbers only).

Participants could freely replay, pause, and scrub the video timeline. Scrubbing resolution was restricted to whole seconds to match the temporal information available to VLMs, which process videos solely as pixel sequences without access to exact timestamps. The interface permitted unrestricted video replay to minimize memory effects



(a) 2D survey interface. Participants see the prior ground truth, question text, and a numeric answer box, and can replay or scrub the video.



(b) 3D survey interface. The layout mirrors the 2D condition, with additional depth prior ground truth.

Figure 38. **Human study interface.** Example screenshots of the 2D (top) and 3D (bottom) survey UIs. Both interfaces present the physical prior, quantitative question, and a numeric input field, while allowing participants to replay and scrub the video timeline.

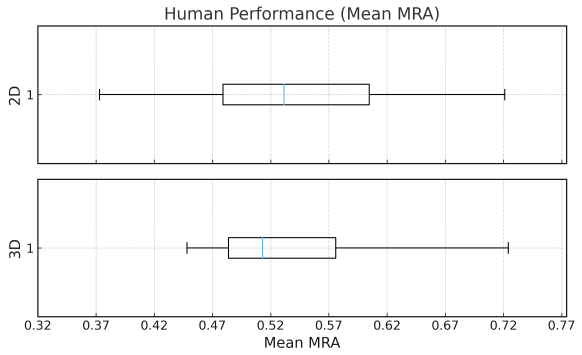


Figure 39. **Distribution of human quantitative reasoning performance.** Horizontal boxplots summarize participant-level mean MRA scores for the 2D (top) and 3D (bottom) survey conditions.

and ensure that both humans and VLMs operated over comparable visual evidence.

H.3. Evaluation Metric

Human responses were evaluated using the same MRA metric introduced in Section 4. For each individual question, we compute MRA by averaging accuracy over ten relative-error tolerance thresholds, yielding a smooth, threshold-agnostic measure of quantitative precision. After computing an MRA score for every answered question, we aggregate performance at the participant level by taking the mean MRA across all questions completed by that participant. Figure 39 presents participants performance for the 2D and 3D surveys, respectively.

H.4. Results and Observations

We observe a few findings from the results.

Strong cross-dimensional consistency. Across participants who completed both tasks, MRA scores for 2D and 3D surveys were highly correlated. High-performing participants in 2D almost universally remained high-performing in 3D, suggesting that human physical intuition is stable and transferrable across dimensional modalities.

Human upper bound substantially exceeds VLM performance. Top human participants achieve MRA = **0.721** in 2D and MRA = **0.724** in 3D, showing better performance than the evaluated VLMs. Although the average human performance is not higher than model performance, the human upper bound remains far above current VLM capabilities. This gap is particularly notable given the modest sample size of our study; even with limited data, humans exhibit strong physical reasoning competence that remains challenging for contemporary models.

Practical implications. These findings indicate that while VLMs may approximate average-level human intuition in some settings, they remain far from achieving human-like precision or matching expert-level reasoning. The human study therefore provides a meaningful benchmark for assessing the gap between current VLM capabilities and the aspirational goal of physically grounded, human-level visual reasoning.

I. Sketchfab Model Sources

In Table 7, we list the Model Name, Author, and Model ID for all Sketchfab models used in the Blender-generated videos in our dataset.

Table 7. Sketchfab 3D Models

Model Name	Author	Model ID
Supermarket trolley broken	kreems	7f460877380349f8886280a596253034
supermarket 1	amogusstrikesback2	ca1def2b7be544068def3cec5852c67e
Pool Table (Animation)	Yanez Designs	0f2ae181a2dd4b00a6ec25073692037f
Bowling Pack (Bowling Pins & Ball)	EverZax	52743c4714c14211ac71d2fe1e5c8da3
Bowling Club	tiunov.se	0b8fae45fcda4fe78f93bb2a899401a6
Elephant	planeta-elefante	f8778fc3d161481abba7ec23a8ddd1e8
Walking Astronaut	Unknown Animaker	d9062a2003df422abdafdc02afdac085
Mr Man Walking	Instinto Ideal Studio	98ccac2b0e2845789b6f789978ca06ed
Jupiter & satellites	Sakado	379fd77b970c4821898c05c483913dec
Model 92A - Great White Shark	DigitalLife3D	702e7b53637f4ded9ca479a8124e810d
Model 95A - Adult Leatherback Sea Turtle, "Mac"	DigitalLife3D	4974c93644a24da280fde68cba74a12d
Model 69A - Striped Bass	DigitalLife3D	35be3af9a7c4441c98109e5562d36c09
Underground Parking Lot	Janis Zeps	0ad5c221525b4bbba3a164c6235d28b8
Airport Car	Jungle Jim	29d1c6260b134a3baf5231f34de1b24f
Airport Catering Truck	rwy00	289f4e2cfa3f4722b0476b1fc37681d8
Towel	TheSpacePunk	6edc341457e44603ab351470ab800493
Plain Mug	LightSwitch	19c8fe5702b544d0a1409d3dac1cf90e
Flour ARIDLL	fwild	d40db94d92ab4e7a82a1c199312f3985
Milk	Multipainkiller Studio	5e8d71045e2040ba8f6619d86d204cf5
Apple [Scan]	hoxsvl.scan	facb1aa6928c4f8f82d87a019b9f134e
PACKAGING EMPAQUE DE HUEVOS	willinando1w08	cd51c5d25bbd4370881aebd3648cfe8c
Kitchen Pack	GRIP420	a513f2e85bc94ae5b6d8cbd74909e3c4
Kitchen Tools	anybody	417e3873fcb34f7ab9744506d7bcc838
Coffee Shop Cup	David Zerba	37e6805f2b7a4158a1d61fe75f8e2a33
Cup	Dmytro Nikonov	7d450bb714034fceaa7b59a0e564f46b
Bird Animations Alex	ahitch3	081fa7f0cfd649b9b07babb4c619acc7
Dove Bird Rigged	FourthGreen	6b91be2a28fb4404a2d57d5ca98bd4dc
POLICE CAR - Belgian VW Transporter	Mickael Boitte	35eaf9505e13464385404402ad865508
Jo on Bike - Rigged & Animated	NEEEU Spaces GmbH	36ee5344e81149858a664cde9f98e835
Simple Factory Scene	Pickeri	804fbe0cd1fa44fa9ca86ae42c82d63d
Car	Paulius	a619dd25a6c04af0b2d8730aa1cb058b
Basketball Court	tiunov.se	5faad7b528124907ab82732ed0c6b743
Basketball NBA Championship Official Trophy	johnnokomis	04f6f1135ffb48749c43c9c20c75fc19
Bouncing Basketball	Maurice Svay	b8731a2fda6849c9a164d1966dc16ff8
Zen Japanese Tea House with Go Board	winters810	a1e95e5efee349a693f30eda32401aef
Chinese Chess	chung_the_artist	7ad0f4f0ebbc455d9e9f829c956dda80
paper airplane	vesicalsnail	0967ab4a9c654a569a13ea1f8d9dca0c
3d soccer ball	BlenderMaster	0400146e0e3c4d8f8b57bfa06d7dfb4c
Cardboard Box	Pricey1600	4e622ef1a09c43e28a49d9fa37f9eeee
Tennis Ball Low-poly PBR	MaX3Dd	e5c2b0e5860549acaa2dfe8b764d5f94
Old Camera Bag .::RAWscan::.	Andrea Spognetta (Spogna)	788f8b75874f417ebde498ffd231410c
Animated ROBOT SDC	SDC PERFORMANCE	3d127f327a6c4033a32b810b5fb071ed
Gift Box	local.yany	83296611584143a3afad6c0d0c0a4227
Sci-fi Box	Igor_K.	ca415724f32043489fcab2ec74582619
Cat Walk	LostBoyz2078	915680200c064815bba75e008ba9efb5
Deer walk	LostBoyz2078	229ba6ba0d1e4811ab89382f74601e16

Continued on next page

Table 7 – *Continued from previous page*

Model Name	**Author**	**Model ID**
Horse	kenchoo	86d47bdcd5ab41238ba44547e4d21f9c
Horse Walk	Amitesh Nandan	93b53ddcec414592842753d1819f3133
Rabbit Rigged	FourthGreen	e7213589744d436b9d96e2dbb31198a5
phoenix bird	NORBERTO-3D	844ba0cf144a413ea92c779f18912042
First Aid DX2 - 300 Followers Celebration	re1monsen	c5ddfa9e6309403083bbce60bdcc3d71
Nathan Animated 003 - Walking 3D Man	Renderpeople	143a2b1ea5eb4385ae90a73657aca3bc
Chernovan Nemesis	Swiss_Fox	c6c91c73e93444f4b72d6c24db778e73
Dragon Fly	LostBoyz2078	1443a7efe5d5450b8db4c15d8ff5c343
Borboleta Azul - Butterfly	Lancaster Modelagem 3D	ab9192b6bc8f49e3baed63e984c7073a
Blue Whale - Textured	Bohdan Lvov	d24d19021c724c3a9134eebcb76b0e0f
jellyfish	yanix	d06a5a553fe641ab92f720527b2278f3
Koi Fish	7PLUS	236859b809984f52b70c94fd040b9c59
Running Raccoon Animation	Santrez	bfce4d4815234c39bcf012352e52c27e
(FREE) Cyberpunk Hovercar	Lionsharp Studios	3205b1075bb44ffc826bce0c2a04d74c
White Eagle Animation Fast Fly	GremorySaiyan	30203bf39e5145f19c79e83c550139d3