

# MORSE-500: A Programmatically Controllable Video Benchmark to Stress-Test Multimodal Reasoning

Zikui Cai<sup>1</sup> Andrew Wang<sup>1</sup> Anirudh Satheesh<sup>1</sup> Ankit Nakhawa<sup>1</sup> Hyunwoo Jae<sup>1</sup>  
Keenan Powell<sup>1</sup> Minghui Liu<sup>1</sup> Neel Jay<sup>1</sup> Sungbin Oh<sup>1</sup> Xiyaow Wang<sup>1</sup> Yongyuan Liang<sup>1</sup>  
Tom Goldstein<sup>1</sup> Furong Huang<sup>1,2</sup>

<sup>1</sup> University of Maryland, College Park

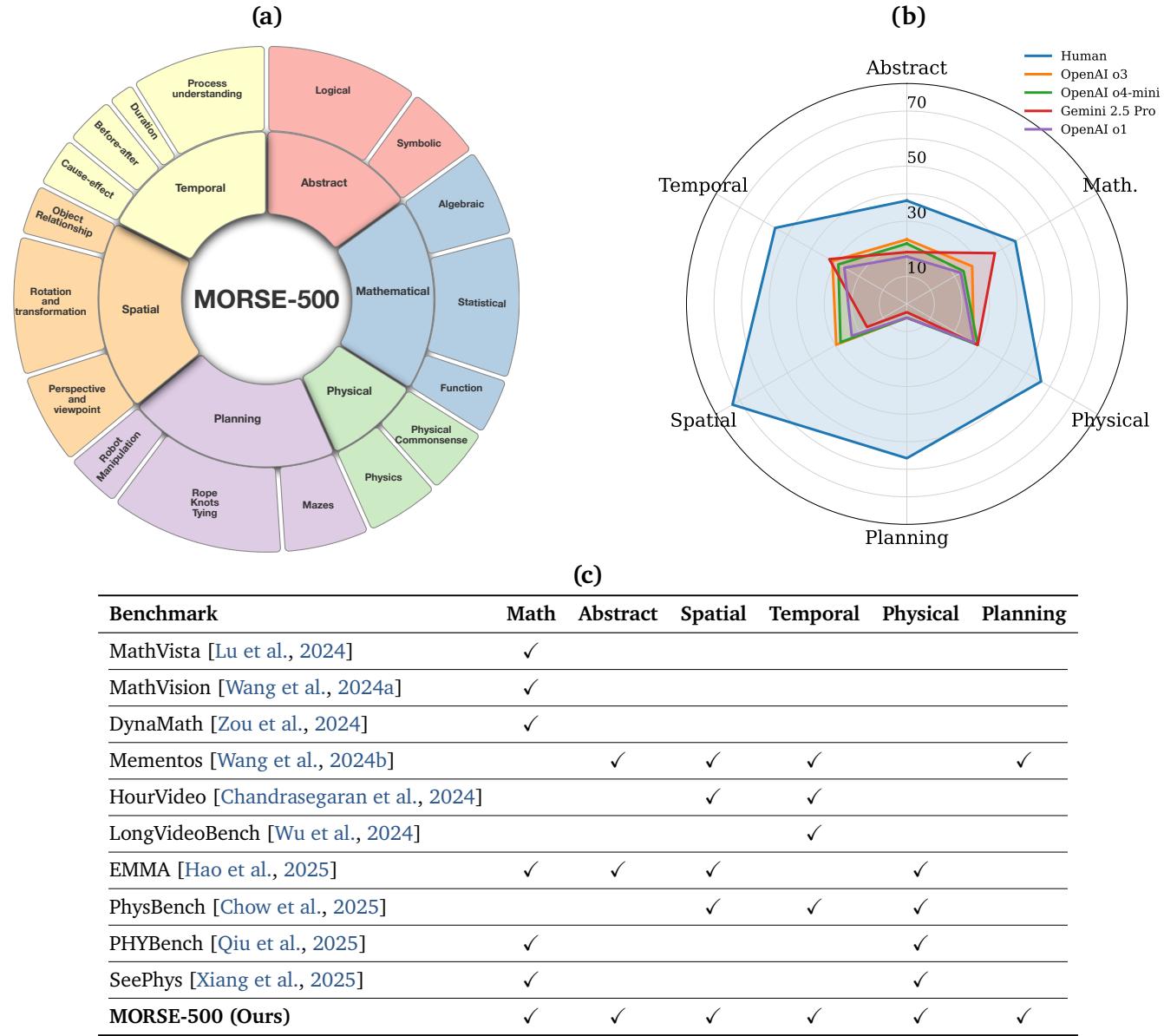
<sup>2</sup> Capital One

Despite rapid advances in vision-language models (VLMs), current benchmarks for multimodal reasoning fall short in three key dimensions. First, they overwhelmingly rely on static images, failing to capture the temporal complexity of real-world environments. Second, they narrowly focus on mathematical problem-solving, neglecting the broader spectrum of reasoning skills—including abstract, physical, planning, spatial, and temporal capabilities—required for robust multimodal intelligence. Third, many benchmarks quickly saturate, offering limited headroom for diagnosing failure modes or measuring continued progress. We introduce MORSE-500 (*Multimodal Reasoning Stress-test Environment*), a video benchmark composed of 500 fully scripted clips with embedded questions spanning six complementary reasoning categories. Each instance is *programmatically generated* using deterministic Python scripts (via Manim, Matplotlib, MoviePy), generative video models, and curated real footage. This script-driven design allows fine-grained control over visual complexity, distractor density, and temporal dynamics—enabling difficulty to be scaled systematically as models improve. Unlike static benchmarks that become obsolete once saturated, MORSE-500 is built to evolve: its controllable generation pipeline supports the creation of arbitrarily challenging new instances, making it ideally suited for stress-testing next-generation models. Initial experiments with state-of-the-art systems—including various Gemini 2.5 Pro and OpenAI o3 which represent the strongest available at the time, alongside strong open-source models—reveal substantial performance gaps across all categories, with particularly large deficits in abstract and planning tasks. We release the full dataset, generation scripts, and evaluation harness to support transparent, reproducible, and forward-looking multimodal reasoning research.

-  **Project:** <https://morse-500.github.io/>
-  **Datasets:** <https://huggingface.co/datasets/video-reasoning/morse-500>
-  **Video Viewer:** <https://huggingface.co/datasets/video-reasoning/morse-500-view>
-  **Code:** <https://github.com/morse-benchmark/morse-500-code>
-  **Contact:** [zikui@umd.edu](mailto:zikui@umd.edu)

## 1. Introduction

**Multimodal reasoning is the frontier.** Recent advances in vision-language models (VLMs) have pushed the boundaries of perception and retrieval [Alayrac et al., 2022, Li et al., 2023, OpenAI, 2023], but robust reasoning remains elusive [Zhang et al., 2023, Lu et al., 2024]. As these models are increasingly deployed in domains requiring inference, planning, and interaction—from embodied agents [Shridhar et al., 2023] to scientific assistants [Shen et al., 2023]—there is a growing need to evaluate and develop their capacity for genuine reasoning over multimodal inputs. This shift demands capabilities that go beyond recognition or retrieval, toward causal, temporal, abstract, and physically grounded understanding.



**Figure 1:** (a) Task distribution of MORSE-500. (b) Performance of the best-performing models on MORSE-500. (c) Comparison of benchmarks across six reasoning categories. Only MORSE-500 spans all categories and supports programmatic control over difficulty and content.

**Benchmark evolution—and persistent blind spots.** The trajectory of evaluation has mirrored model capabilities: early benchmarks focused on recognition (e.g., TextVQA [Singh et al., 2019], DocVQA [Mathew et al., 2021], OCR-VQA [Mishra et al., 2019]), then knowledge retrieval (e.g., MMMU [Yue et al., 2023], ScienceQA [Lu et al., 2022]), and more recently mathematical reasoning [Lu et al., 2024, Wang et al., 2024a, Zou et al., 2024], and physical reasoning [Chow et al., 2025, Qiu et al., 2025, Xiang et al., 2025]. However, most of these datasets rely on static images and narrowly scoped question types, overlooking reasoning in dynamic, interactive environments where the ability to process sequences, anticipate outcomes, and generalize abstract patterns is essential.

**Limitations of current benchmarks.** Despite recent progress, today’s reasoning benchmarks suffer from three structural limitations:

- **Static modality bias:** Most benchmarks rely on single-frame images, ignoring the temporal evolution and causality inherent to many real-world tasks.
- **Narrow reasoning spectrum:** They often focus heavily on math word problems [Lu et al., 2024, Wang et al., 2024a, Zou et al., 2024], underrepresenting reasoning types such as spatial logic, temporal inference, physical causality, abstraction, and multi-step planning.
- **Rapid saturation:** Many benchmarks are quickly saturated by current models [Lu et al., 2024], offering little diagnostic signal once performance plateaus.

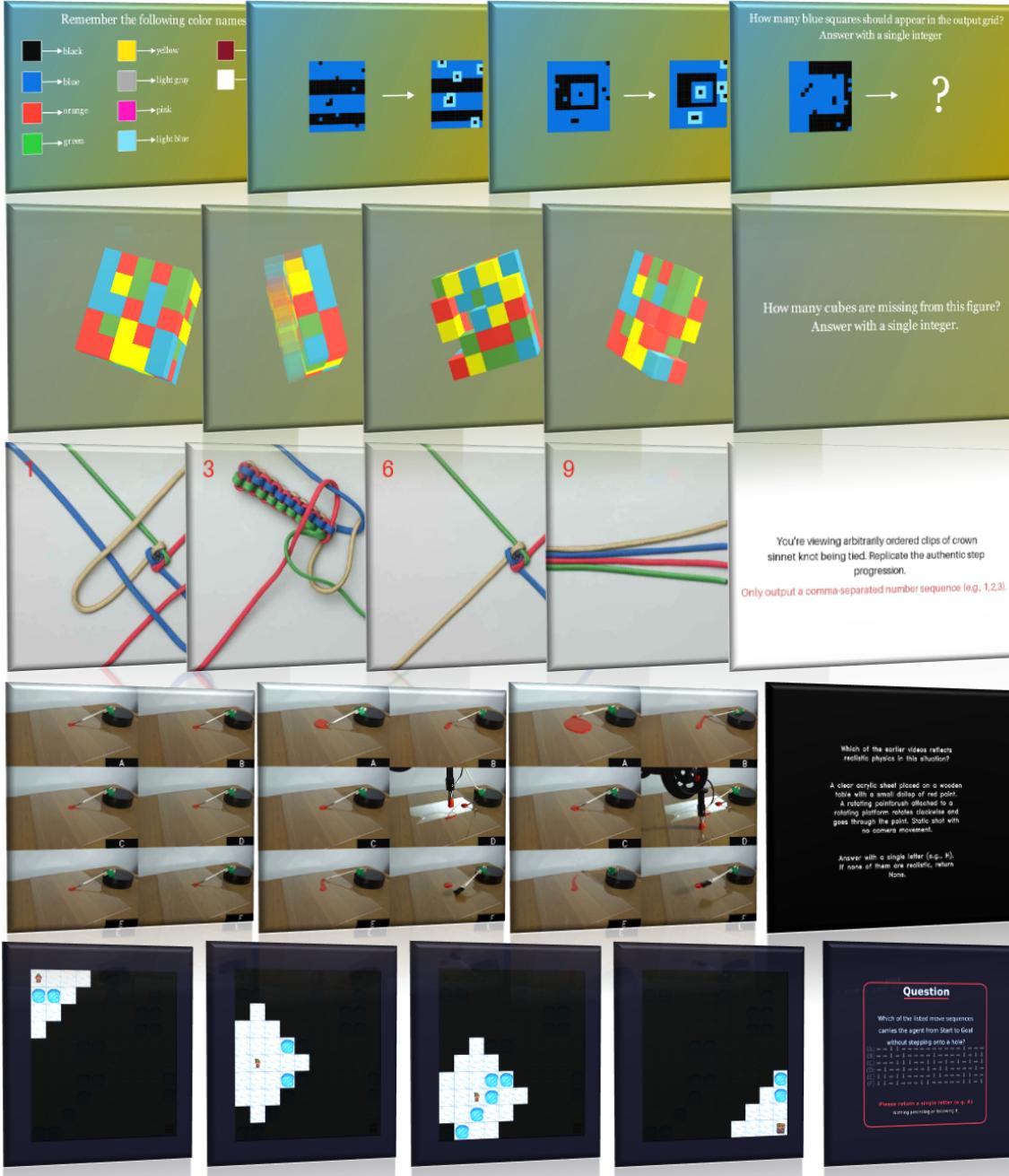
Moreover, current benchmarks often conflate reasoning with perception and retrieval [Zhang et al., 2023], making it difficult to assess whether models are genuinely reasoning or merely pattern-matching. This highlights the need for a principled evaluation framework that systematically varies difficulty while explicitly controlling for perceptual and knowledge-based confounds.

**Introducing MORSE-500.** To address these challenges, we present **MORSE-500**—a video benchmark explicitly designed to evaluate diverse forms of multimodal reasoning across time. The key advantages of MORSE-500 include:

- **Diverse Reasoning Categories:** MORSE-500 comprises 500 fully-scripted videos, each embedding a question within its visual narrative. These span six complementary reasoning types: mathematical, abstract, spatial, temporal, physical, and planning, offering a comprehensive and balanced stress test across the full spectrum of reasoning challenges.
- **Truly Vision-Centric:** Questions are embedded directly within the video content, rather than provided as separate textual prompts. This ensures models must extract and reason over information grounded in visual input alone, eliminating shortcut cues and better simulating real-world visual understanding.
- **Scalable Difficulty:** A core innovation of MORSE-500 is its programmatic controllability. All videos are deterministically generated via Python scripts that combine [Manim Community \[2025\]](#), Matplotlib, MoviePy, generative video models, and curated real footage, enabling fine-grained control over scene complexity, distractor density, and temporal duration. Difficulty can be precisely scaled by adjusting script parameters, allowing MORSE-500 to serve as a stress test that evolves with model capabilities. We plan to release ever-harder versions as models approach saturation.

**SOTA Models Demonstrate Significant Room for Improvement in Multimodal Reasoning.** In our initial evaluation, state-of-the-art models—OpenAI-o3 [OpenAI, 2025], Gemini 2.5 Pro [Google, 2025]—show relatively low performance across all reasoning categories in the MORSE-500 benchmark, with especially pronounced challenges in abstract reasoning and planning tasks (see Table 2). These results underscore the need for models with improved temporal memory, compositionality, and generalization across dynamic contexts.

**A Testbed for Long-Term Progress.** MORSE-500 offers a scalable and reproducible foundation for advancing research in multimodal reasoning. By releasing the full corpus, programmatic generation scripts with ground-truth annotations, and a lightweight evaluation harness, we aim to support transparent and forward-compatible benchmarking. As a dynamic, extensible benchmark, MORSE-500 is well positioned to catalyze



**Figure 2:** Representative examples from MORSE-500 demonstrating different reasoning categories. Each row shows sequential frames sampled from a video task: **(top)** abstract reasoning with ARC-AGI2 pattern recognition requiring rule induction from visual transformations, **(2nd)** spatial reasoning through cube rotation testing 3D transformation understanding and mental rotation abilities, **(3rd)** planning reasoning via rope tying sequences assessing multi-step procedural understanding, **(4th)** physical reasoning using real vs. AI-generated video discrimination to test intuitive physics understanding, and **(bottom)** planning reasoning through maze navigation testing spatial pathfinding and goal-directed behavior. Questions are embedded directly within the video content, requiring models to extract relevant information from the temporal sequence rather than from separate text prompts. Models are simply prompted with "Answer the question in this video" with no additional context, ensuring evaluation of true multimodal reasoning capabilities across the temporal dimension. Visit our [Website](#) and [HuggingFace dataset](#) to view more videos.

the development of reasoning-centric architectures, evaluation methodologies, and failure diagnostics for the next generation of vision–language models.

## 2. MORSE-500

### 2.1. Design Principles

The development of MORSE-500 was guided by four foundational principles aimed at addressing critical limitations in existing multimodal reasoning benchmarks while establishing a robust framework for systematic evaluation and future extensibility. Our design emphasizes **temporal-first evaluation** through video-based tasks that require genuine temporal understanding, **truly vision-centric assessment** where questions are embedded directly within visual content rather than provided as separate text, a **comprehensive reasoning taxonomy** grounded in established cognitive frameworks spanning six complementary reasoning categories, and **programmatic generation with scalable difficulty** enabling systematic complexity control and forward compatibility as model capabilities advance.

**Comprehensive Reasoning Taxonomy.** MORSE-500 spans six complementary reasoning categories (see table 1), each grounded in established cognitive science frameworks and designed to evaluate distinct cognitive capabilities essential for robust multimodal intelligence. Our taxonomy draws from the Cattell-Horn-Carroll (CHC) theory of cognitive abilities [Carroll, 1993], dual-process theory [Evans and Stanovich, 2013], and computational models of reasoning [Holyoak and Morrison, 2013]. **Abstract reasoning** targets pattern recognition, logical inference, and symbolic reasoning associated with fluid intelligence, requiring operation at multiple levels of abstraction [Gentner, 1983]. **Mathematical reasoning** evaluates fluid reasoning (Gf) and quantitative knowledge (Gq), assessing arithmetic operations, algebraic relations, and quantitative analysis through dynamic visualizations that integrate visual-spatial information with numerical processing [Dehaene, 2011]. **Physical reasoning** examines intuitive physics understanding through object dynamics and causal interactions, bridging perceptual experience with abstract physical knowledge [McCloskey et al., 1983]. **Planning reasoning** evaluates executive function and goal-directed behavior, emphasizing multi-step reasoning and sequential decision-making central to cognitive control theories [Miyake et al., 2000]. **Spatial reasoning** corresponds to visual-spatial processing (Gv), testing 3D transformations, perspective understanding, and object relationships through tasks requiring mental model construction and manipulation [Shepard and Metzler, 1971]. **Temporal reasoning** addresses sequence understanding and causal inference over time, evaluating temporal order tracking and future state prediction aligned with event segmentation theory [Zacks et al., 2007].

**Programmatic Generation with Scalable Difficulty and Forward Compatibility.** To ensure extensibility, reproducibility, and precise experimental control, all videos are generated through deterministic Python scripts utilizing established libraries including Manim for mathematical visualizations and 2D/3D object rendering and animation, Matplotlib for statistical graphics, MoviePy for image transforming effects, and video generative models for realistic scenario generation. This programmatic foundation enables fine-grained manipulation of complexity parameters including entity count, reasoning depth, distractor density, temporal dynamics (static to highly dynamic sequences), and visual complexity (minimal to high-noise environments).

A core innovation of MORSE-500 lies in its systematic difficulty progression that can evolve alongside model capabilities. Unlike static benchmarks that quickly saturate and become obsolete, MORSE-500 functions as a living evaluation framework where new instances can be generated with precisely specified complexity profiles.

Difficulty scaling operates across multiple orthogonal dimensions: structural complexity (number of entities, interaction patterns), cognitive demands (reasoning depth, abstraction level), environmental challenges (visual noise, occlusion, temporal irregularity), and task-specific parameters (plan length for planning tasks, transformation complexity for spatial reasoning). The deterministic generation process ensures perfect reproducibility while supporting systematic difficulty scaling as model capabilities advance, enabling the identification of specific reasoning weaknesses and targeted evaluation of architectural improvements. This scalable architecture ensures that MORSE-500 remains diagnostically valuable as models improve, supporting the generation of arbitrarily challenging instances on demand while maintaining consistent evaluation standards and serving as an effective stress test for next-generation multimodal systems.

## 2.2. Dataset Statistics

MORSE-500 comprises 500 carefully curated video instances with embedded reasoning questions, systematically distributed across six complementary reasoning categories to ensure comprehensive cognitive coverage and balanced evaluation.

Table 1: Dataset Distribution by Reasoning Domain

Abstract Reasoning	Mathematical Reasoning	Physical Reasoning
<b>12.8%   64 instances</b> Pattern recognition, logical inference, symbolic reasoning	<b>16.8%   84 instances</b> Arithmetic operations, algebraic relations, geometric analysis, quantitative comparisons	<b>12.8%   64 instances</b> Object dynamics, causal interactions, physics laws, physical commonsense
Planning Reasoning	Spatial Reasoning	Temporal Reasoning
<b>20.0%   100 instances</b> Multi-step reasoning, goal-directed problem solving	<b>21.6%   108 instances</b> Object relationships, spatial transformations, 3D reasoning	<b>16.0%   80 instances</b> Sequence understanding, causal inference over time

**Category Distribution and Strategic Allocation.** As shown in table 1, the dataset employs a purposeful distribution across reasoning domains, with category allocation reflecting both cognitive importance and evaluation priorities: Spatial reasoning (21.6%, 108 instances) receives the largest allocation given its fundamental role in multimodal understanding; Planning reasoning (20.0%, 100 instances) emphasizes multi-step reasoning capabilities critical for autonomous systems; Mathematical reasoning (16.8%, 84 instances) covers structured problem-solving across arithmetic, algebraic, and geometric domains; Temporal reasoning (16.0%, 80 instances) evaluates sequence understanding and causal inference over time; while Abstract reasoning (12.8%, 64 instances) and Physical reasoning (12.8%, 64 instances) provide focused assessment of pattern recognition and physics-based inference respectively.

Within each category, tasks span multiple specialized subcategories: mathematical reasoning includes arithmetic operations, algebraic relations, geometric analysis, and quantitative comparisons; abstract reasoning encompasses pattern recognition, logical inference, and symbolic reasoning; spatial reasoning covers object relationships, spatial transformations, and 3D reasoning; temporal reasoning evaluates sequence understanding and causal inference over time; physical reasoning tests object dynamics, causal interactions, and physics laws; and planning reasoning examines multi-step reasoning and goal-directed problem solving.

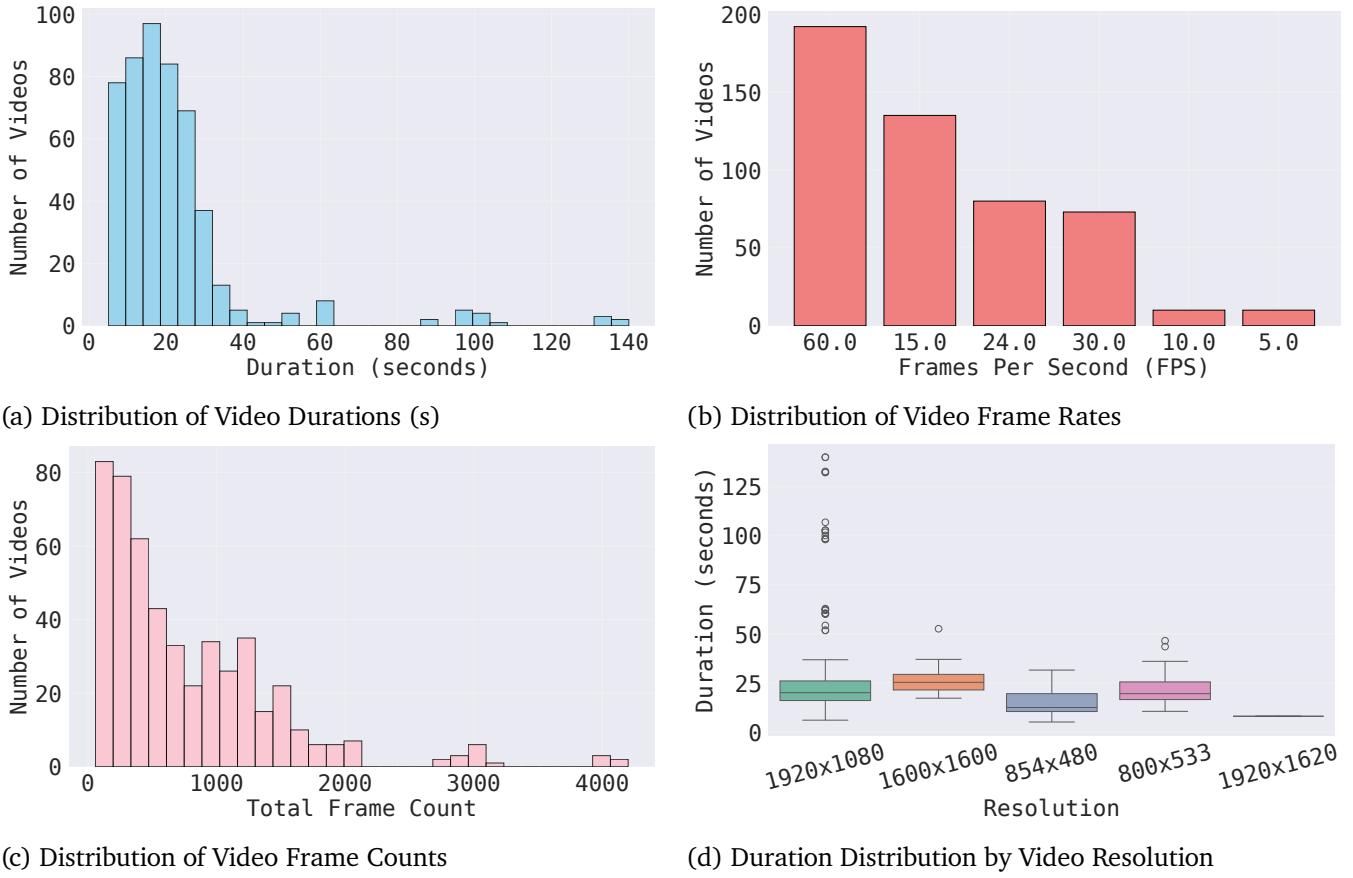


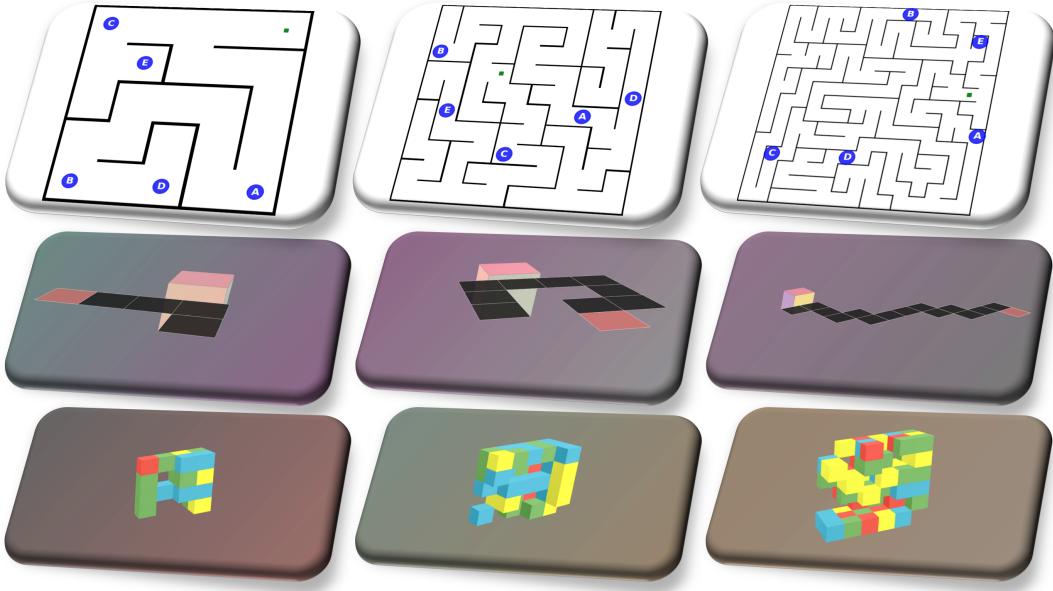
Figure 3: Dataset statistics including duration, fps, frame count, and resolution.

**Video Characteristics.** Our dataset comprises 500 videos with a total duration of 3.1 hours and an aggregate file size of 1.4 GB. As shown in Figure 3, the videos exhibit considerable diversity in temporal characteristics, with durations ranging from 5.1 to 140.0 seconds (mean: 22.1s, median: 18.0s, std: 19.3s), indicating a predominance of short-form content with high information density. Frame rates vary significantly across the dataset, spanning from 5.0 to 60.0 FPS with a mean of 35.6 FPS and median of 30.0 FPS. Notably, 60 FPS emerges as the most frequent frame rate, reflecting modern high-quality video capture standards. In terms of spatial resolution, the dataset demonstrates a multi-modal distribution: 45.2% of videos are recorded in Full HD (1920×1080), while 27.0% are in standard definition (854×480), and 16.0% utilize an intermediate resolution of 800×533 pixels. The remaining videos span various resolutions including square formats (1920×1620, 1600×1600), collectively representing diverse recording devices and platform requirements. File sizes exhibit high variability, ranging from less than 0.1 MB to 68.2 MB (mean: 2.9 MB, median: 0.8 MB), with the substantial difference between mean and median suggesting a right-skewed distribution dominated by smaller files with occasional larger outliers. This heterogeneous composition reflects the natural diversity of developer-generated content across different software environments and conditions.

**Complexity Scaling and Difficulty Gradation.** Each reasoning category incorporates systematic difficulty variation through programmatically controlled parameters. Complexity is measured along multiple orthogonal dimensions: (1) *entity complexity*, ranging from 2-3 simple objects to 15+ interconnected elements; (2)

*reasoning depth*, spanning 1-2 step direct inference to 5+ step multi-hop reasoning chains; (3) *distractor density*, varying from minimal noise to high-distraction environments with 8+ irrelevant elements; (4) *temporal complexity*, from static sequential presentation to dynamic concurrent processes; and (5) *visual complexity*, including occlusion patterns, perspective changes, and rendering fidelity variations.

Difficulty distributions are calibrated to span from tasks solvable by current models (ensuring non-trivial baseline performance) to challenges requiring advanced reasoning capabilities. Approximately 20% of instances target current model capabilities, 50% represent moderate extensions requiring improved reasoning, and 30% constitute stress tests for next-generation systems.

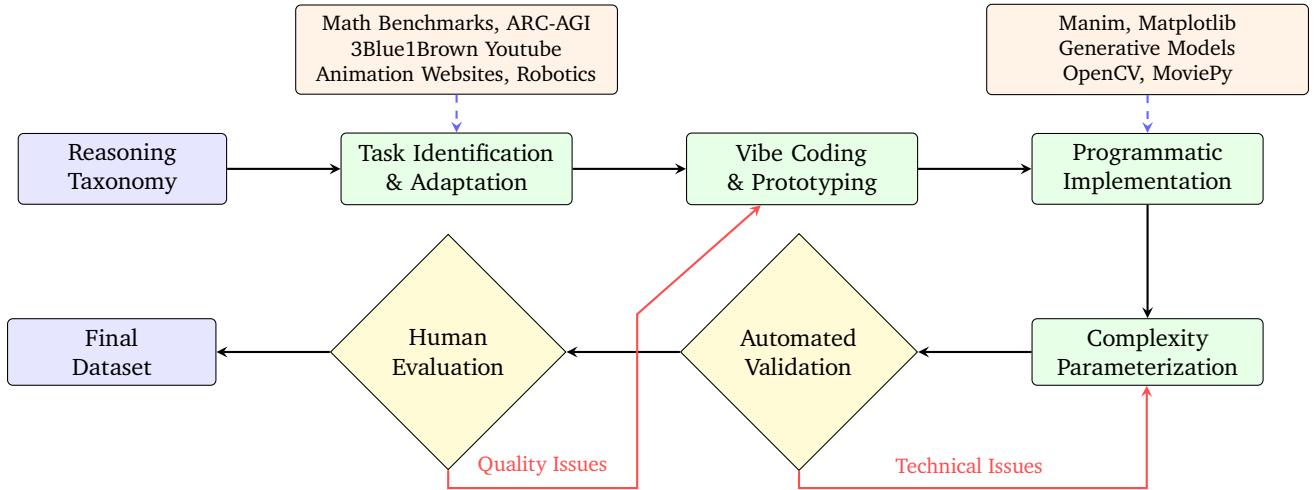


**Figure 4:** Programmatic difficulty scaling in MORSE-500. Each row demonstrates how task complexity can be systematically increased while maintaining the core reasoning category: **(top)** maze navigation tasks with increasing path complexity; **(middle)** abstract shape transformation tasks with more complex geometric patterns and more reasoning steps; **(bottom)** 3D cube visualization with increasing structural complexity and color patterns.

The dataset’s programmatic foundation enables systematic expansion: new instances can be generated with specified difficulty profiles, novel parameter combinations can explore untested reasoning scenarios, and category-specific stress tests can be developed as model capabilities advance, ensuring MORSE-500 remains a relevant evaluation framework for future multimodal reasoning research.

### 2.3. Data Generation and Validation Process

MORSE-500 employs a systematic data generation pipeline that transforms high-quality reasoning tasks into challenging video-based assessments through programmatic generation and rigorous validation. Our approach ensures both scalable content creation and reliable evaluation standards across all reasoning categories.



**Figure 5:** MORSE-500 data generation and validation pipeline. The process flows systematically from reasoning taxonomy through task adaptation, prototyping, and implementation to final validation. External inspiration sources (blue dashed) inform task design and implementation, while quality assurance loops (red) enable iterative refinement at technical and cognitive validation stages.

### 2.3.1. Programmatic Content Creation Pipeline

Our data generation process follows a principled four-stage pipeline designed to create diverse, challenging, and reproducible video reasoning tasks.

**Task Identification and Adaptation.** Beginning with our established six-category reasoning taxonomy, we systematically identify high-quality exemplars from existing benchmarks [Wang et al., 2024a, Chollet et al., 2025, Motamed et al., 2025] and expand them into dynamic video formats. For **abstract reasoning**, we adapt ARC-AGI pattern recognition tasks Chollet [2019] into animated sequences showing rule transformations over time, and included other tasks such as symbolic reasoning with anagram transformation. **Mathematical reasoning** draws inspiration from dynamic visualizations in educational content, particularly the **3Blue1Brown YouTube channel**'s approach to mathematical explanation through animation, adapting static problems from math benchmarks [Lu et al., 2024] into temporal mathematical narratives. **Spatial reasoning** extends traditional mental rotation tasks from [Wang et al., 2024a] into continuous 3D transformations with occlusion and perspective changes. **Physical reasoning** incorporates real-world physics scenarios from established datasets [Motamed et al., 2025] while generating synthetic alternatives using state-of-the-art video generation models for discrimination tasks. **Planning reasoning** leverages maze and frozen-lake environment Brockman et al. [2016], Ivanitskiy et al. [2023], animated rope knots tying database [Grog, 2025], and robotic manipulation datasets [Zhu et al., 2023, Wu et al., 2025, Haldar et al., 2023, Wu et al., 2023, Wang et al., 2023, Bahl et al., 2023], transforming sequential action demonstrations into temporal reasoning challenges. **Temporal reasoning** creates novel sequence understanding tasks through procedural animations and cause-effect chains.

**Programmatic Implementation.** We employ a "vibe coding" approach where domain experts first create conceptual prototypes that capture the essential reasoning challenges, then iteratively refine these into fully functional, parameterized generation scripts. Each reasoning category utilizes specialized libraries:

Manim for mathematical animations, 3D objects and abstract pattern generation, Matplotlib for statistical visualizations, OpenCV for computer vision processing, and MoviePy for video composition and editing. The implementation process involves multiple iterations of prototype development, parameter tuning, and edge case handling to ensure robust generation across the full complexity spectrum.

**Complexity Parameterization.** Each generation script incorporates systematic complexity controls spanning multiple orthogonal dimensions. Entity complexity ranges from 2-3 simple objects to 15+ interconnected elements; reasoning depth spans 1-2 step direct inference to 5+ step multi-hop reasoning chains; distractor density varies from minimal noise to high-distraction environments with 8+ irrelevant elements; temporal complexity progresses from static sequential presentation to dynamic concurrent processes; and visual complexity includes occlusion patterns, perspective changes, and rendering fidelity variations. These parameters enable precise difficulty calibration and systematic scaling as model capabilities advance.

**Content Diversification.** To ensure comprehensive coverage within each category, we generate multiple task variants with independently sampled parameters. This approach balances challenge variety with experimental reproducibility, creating diverse reasoning scenarios while maintaining consistent evaluation standards. The programmatic foundation supports rapid iteration and expansion, enabling the creation of new task variants on demand.

### 2.3.2. Validation and Quality Assurance

To ensure the reliability and quality of MORSE-500, we implemented a rigorous multi-stage validation pipeline encompassing both automated verification and human evaluation.

**Automated Technical Validation.** All generated videos undergo comprehensive automated validation to ensure technical integrity and content completeness. Our validation pipeline systematically checks for: (1) technical quality metrics including resolution consistency (512p minimum) and proper codec encoding; (2) content completeness verification ensuring all necessary visual elements for question answering are present and clearly visible throughout the temporal sequence; and (3) ground-truth label accuracy through automated cross-referencing with generation parameters. This automated process identifies and flags potential issues including rendering artifacts, incomplete animations, and parameter-label mismatches, which are subsequently addressed through iterative script refinement.

**Expert Human Evaluation.** To validate the benchmark’s cognitive validity and eliminate ambiguous cases, we conducted systematic human evaluation by paper authors on randomly sampled data slices. Each evaluator independently assessed video-question pairs for: (1) question clarity and unambiguity, ensuring that questions have single, well-defined correct answers derivable from the visual content; (2) visual-semantic alignment, verifying that all information necessary to answer questions is present and interpretable in the video sequence; and (3) difficulty appropriateness, confirming that tasks are challenging yet solvable for competent human reasoners.

**Iterative Refinement.** Based on validation feedback, we implemented systematic improvements to our generation pipeline. Common issues identified during validation—such as temporal misalignment between question presentation and relevant visual events, insufficient visual contrast for critical elements, and

edge cases in procedural generation—were addressed through targeted script modifications and parameter adjustments. This iterative process ensures that MORSE-500 provides reliable, unambiguous, and cognitively valid assessment of multimodal reasoning capabilities across dynamic visual contexts.

The complete pipeline from task identification to validated content typically requires 2-3 iterations per reasoning category, with the programmatic approach enabling rapid refinement and consistent quality across the entire dataset.

### 3. Experiments

#### 3.1. Settings

**Models and Baselines.** To evaluate the current frontier of multimodal reasoning, we benchmarked a diverse set of vision–language models on MORSE-500. Our selection spans both proprietary and open-source models with varying architectural backbones and pretraining objectives.

**Closed-source models** include Gemini 2.5 Pro [Google, 2025], Google DeepMind’s strongest model with advanced video and visual reasoning capabilities, and other models from Gemini family, including Gemini 2.5 Flash, Gemini 2.0 Flash, Gemini 2.0 Flash-Lite, and Gemini 1.5 Pro[Google, 2024]. And OpenAI o3 [OpenAI, 2025], OpenAI’s strongest LMM with improved temporal and spatial understanding. We also include GPT 4o [Hurst et al., 2024], o1 [Jaech et al., 2024], and o4-mini[OpenAI, 2025].

**Open-source models** encompass a wide spectrum of scale and design. We evaluate multiple model variants of one of the strongest open source model family - Qwen2.5 VL [Bai et al., 2023] of different sized (3B, 7B, 32B, 72B) with and without quantization (AWQ), these model are all with video support. Other video supporting models include Qwen2.5-Omni-7B [Xu et al., 2025], which is able to perform audio understanding and generation tasks beyond vision and language. We also include LLaVA-NeXT-Video-7B [liu, 2024], a video-capable variant of LLaVA; and MiniCPM-o-2\_6 [Yao et al., 2024], a lightweight vision-language model designed for efficiency. For VLMs that only has image support, we include InternVL3 8B [Zhu et al., 2025], a strong multilingual VL baseline; and Gemma 3 [Gemma Team et al., 2025]. These open models vary widely in capability, allowing us to assess how scale, quantization, and architecture impact multimodal reasoning under the same benchmark.

**Evaluation Protocol.** All models were given the original video clips from MORSE-500. For models with native video understanding, we used the entire clip. For image-only models, frames were sampled at 0.5-second intervals (2fps) and with max 32-frame context. We also provide ablation studies on the influence of the fps and max number of frames in section 3.4. All models were prompted with the minimal instruction: "Answer the question in this video." No few-shot examples or format-specific guidance were provided to isolate the models' intrinsic reasoning abilities. Note for Gemini 2.5 series models (2.5 Pro and 2.5 Flash) and OpenAI models (o4-mini, o3, o1, 4o), the api does not yet support video input, so we provide image frames. Videos were downsampled to a maximum side length of 512 pixels (preserving aspect ratio).

**Metrics.** We report accuracy as the primary evaluation metric—the percentage of correctly answered questions over the benchmark. We provided detailed instructions on the output formatting in the video, and

Table 2: Accuracy (%) on MORSE-500 across all six reasoning categories and overall average. Closed-source models are marked in purple, and open-source models are marked in blue.

Model	All	Abstract	Math.	Physical	Planning	Spatial	Temporal
Human	55.4	37.5	45.5	56.3	56.0	73.1	55.2
o3	23.6	23.4	27.4	28.1	5.0	29.6	31.2
o4-mini	22.2	21.9	23.8	29.7	5.0	27.8	28.7
Gemini 2.5 Pro	21.8	18.8	36.9	29.7	3.0	16.7	32.5
o1	19.8	17.2	22.6	28.1	5.0	23.1	26.2
Gemini 2.5 Flash	19.2	9.4	35.7	28.1	1.0	24.1	18.8
Gemini 1.5 Pro	18.8	12.5	21.4	26.6	1.0	26.9	26.2
Qwen2.5 VL 72B	17.8	6.2	21.4	34.4	1.0	22.2	25.0
GPT 4o	17.4	17.2	20.2	34.4	4.0	12.0	25.0
Qwen2.5 VL 32B AWQ	16.8	14.1	23.8	34.4	1.0	15.7	18.8
Qwen2.5 VL 72B AWQ	16.4	12.5	11.9	29.7	2.0	27.8	16.2
Gemini 2.0 Flash	16.0	12.5	29.8	28.1	0.0	13.0	18.8
Qwen2.5 VL 32B	15.6	9.4	19.0	29.7	2.0	16.7	21.2
Gemma 3 27b	14.6	20.3	20.2	25.0	1.0	13.0	15.0
Gemini 2.0 Flash-Lite	14.2	17.2	21.4	21.9	2.0	14.8	12.5
MiniCPM-o 2.6	11.6	4.7	10.7	23.4	1.0	16.7	15.0
Qwen2.5 Omni 7B	11.4	6.2	9.5	21.9	2.0	15.7	15.0
Qwen2.5 VL 7B	11.2	7.8	11.9	25.0	2.0	12.0	12.5
InternVL3 8B	7.8	6.2	6.0	14.1	1.0	11.1	10.0
Qwen2.5 VL 3B	7.6	9.4	3.6	18.8	1.0	9.3	7.5
LLaVA-NeXT-Video 7B	5.0	1.6	11.9	6.2	0.0	5.6	5.0

we extract the answers from the model prediction using a LLM (e.g. Qwen2.5 72B AWQ) and perform string matching for accuracy calculation, following MathVista [Lu et al., 2024].

**Reproducibility.** To ensure transparency and future benchmarking, the full evaluation code, the intermediate evaluation results of different models, and video generation scripts are released on github. The benchmark data is available on huggingface, including all 500 videos, ground-truth answers, and usage instructions.

### 3.2. Quantitative Results

Table 2 presents accuracy across reasoning categories for all evaluated models. Overall performance remains substantially below human-level capabilities, with even the strongest systems averaging below 25% accuracy—a significant gap compared to human performance at 55.4%.

**Proprietary Model Performance.** Among proprietary models, OpenAI’s o3 achieves the highest overall score of 23.6%, demonstrating relatively balanced performance across categories with particular strength in temporal reasoning (31.2%). Gemini 2.5 Pro follows closely at 21.8%, exhibiting notable proficiency in mathematical reasoning (36.9%) and temporal understanding (32.5%), but struggling significantly with abstract reasoning (18.8%) and planning tasks (3.0%). Interestingly, while Gemini 2.5 Flash achieves similar mathematical performance (35.7%), it shows markedly weaker performance in abstract reasoning (9.4%)

and planning (1.0%), suggesting that model scale and optimization strategies significantly impact reasoning capabilities across different cognitive domains.

The performance patterns reveal interesting trade-offs: models optimized for mathematical reasoning tend to excel in structured, rule-based tasks but struggle with open-ended abstract reasoning. Conversely, models with stronger general reasoning capabilities (like o3) show more balanced performance but may sacrifice peak performance in specific domains.

**Open-Source Model Analysis.** The open-source landscape demonstrates a clear scaling relationship between model size and reasoning performance. Among the Qwen2.5 VL family, the 72B model achieves 17.8% overall accuracy, substantially outperforming smaller variants (32B: 16.8%, 7B: 11.2%, 3B: 7.6%). However, quantization effects are mixed: while the 72B AWQ model shows slightly lower overall performance (16.4%) compared to its full-precision counterpart, the 32B AWQ variant actually outperforms the standard 32B model (16.8% vs. 15.6%), suggesting that quantization impacts vary with model scale.

Specialized models show domain-specific strengths: MiniCPM-o 2.6, despite lower overall performance (11.6%), demonstrates competitive physical reasoning capabilities (23.4%), indicating that targeted optimization can yield focused improvements. However, highly specialized models like LLaVA-NeXT-Video 7B, despite being designed for video understanding, achieve only 5.0% overall accuracy, highlighting the gap between video comprehension and video-based reasoning.

**Category-Specific Performance Patterns.** Performance varies dramatically across reasoning categories, revealing systematic weaknesses in current models:

*Mathematical Reasoning:* This category shows the highest performance across most models, with several systems exceeding 20% accuracy. The structured nature of mathematical problems and their prevalence in training data likely contributes to this relative strength.

*Physical Reasoning:* Models demonstrate moderate competency (20-35% for top performers), suggesting that intuitive physics concepts are partially captured in current training paradigms. However, the gap from human performance (56.3%) remains substantial.

*Spatial and Temporal Reasoning:* Performance in these categories is moderate but inconsistent across models, with some showing surprising deficits (e.g., Gemini 2.5 Pro's 16.7% spatial accuracy despite strong mathematical performance).

*Abstract Reasoning:* This category proves most challenging for all models, with even the best performers struggling to exceed 25% accuracy. The poor performance suggests fundamental limitations in pattern recognition, analogical thinking, and rule induction—core components of general intelligence.

*Planning Reasoning:* Perhaps most concerning, planning tasks show near-random performance across all models (0-5% accuracy), indicating a critical gap in multi-step reasoning and goal-directed behavior. This weakness has significant implications for real-world deployment in autonomous systems.

**Implications and Model Limitations.** The uniformly low performance across all reasoning categories, particularly in abstract reasoning and planning, suggests that current multimodal models suffer from fundamental architectural limitations rather than mere training inefficiencies. The inability to perform multi-step reasoning, maintain temporal coherence, and engage in abstract pattern matching indicates that these models may be primarily engaging in sophisticated pattern matching rather than genuine reasoning.

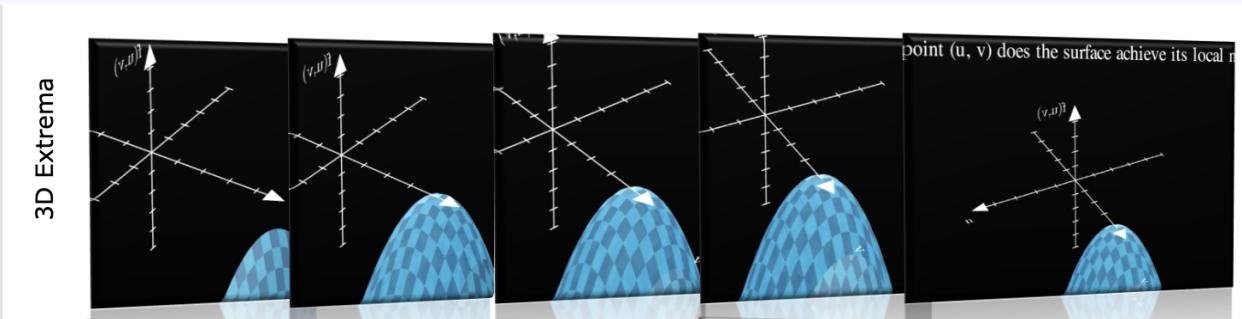
Furthermore, the substantial human-model performance gaps (30+ percentage points in most categories) underscore that achieving human-level multimodal reasoning remains a significant challenge. The particularly poor performance on planning tasks raises questions about the suitability of current models for autonomous decision-making applications.

### 3.3. Qualitative Results

Table 6 illustrates representative model outputs on challenging examples from MORSE-500, and more examples can be found in Appendix C. We observe consistent failure patterns in tasks requiring abstract reasoning and multi-step planning, where even large-scale models falter at integrating temporal cues or executing compositional logic.

Smaller models often rely on superficial visual features, anchoring on salient but irrelevant distractors. In contrast, more capable models—such as Gemini 2.5 Pro and o3—attempt partial reasoning chains but frequently fall short of arriving at the correct solution. These qualitative trends reinforce the quantitative gaps reported earlier and highlight the need for better memory, inference, and grounding in dynamic visual contexts.

**Mathematical Reasoning - 3D Extrema Plot**



**Figure 6:** Example of mathematical reasoning failure across state-of-the-art models. All three models hallucinate the maximum at the origin, whereas the ground truth is  $(0, 4)$ .

**Query:** Answer the question in this video.

**Question text:** At what point  $(u, v)$  does the surface achieve its local maximum?

**Ground truth:** **(0,4)**

**Response from Qwen3-235B-A22B**

To determine the point  $(u, v)$  at which the surface achieves its local maximum, let's analyze the given 3D surface plot step by step.

The surface is a 3D plot with coordinates  $(u, v, z)$ , where  $z$  represents the height of the surface. The goal is to find the point  $(u, v)$  where the surface reaches its local maximum.

The surface appears to have a peak or a highest point. The peak is visually identifiable as the point where the surface rises the most and then begins to descend in all directions.

From the plot: the surface has a distinct peak. The peak is located near the origin of the  $u-v$  plane, but it is not exactly at the origin. The surface slopes downward in all directions from this peak.

The peak appears to be at a point where both  $u$  and  $v$  are close to zero but not exactly zero. Based on the symmetry and the shape of the surface, the peak is likely at a point where  $u$  and  $v$  are small positive or negative values.

The exact coordinates of the peak cannot be determined precisely without numerical data, but visually, the peak is very close to the origin. Therefore, the point  $(u, v)$  where the surface achieves its local maximum is approximately:

**(0, 0)**

#### Response from o3 (summarized)

The model begins by sampling video frames at regular intervals (e.g., 1 fps or 0.5 fps) using OpenCV:

```
import cv2, numpy as np, matplotlib.pyplot as plt
video_path = '/mnt/data/LocalExtreme3DQuestion1.mp4'
cap = cv2.VideoCapture(video_path)
fps = cap.get(cv2.CAP_PROP_FPS)
total = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
duration = total / fps
sample_rate = 30 # 2 fps
frames = []
for i in range(0, total, sample_rate):
    cap.set(cv2.CAP_PROP_POS_FRAMES, i)
    ret, frame = cap.read()
    if ret:
        frames.append((i, frame))
cap.release()
print(len(frames)) # 15 frames
```

It then inspects the sampled frames to locate the overlay text asking, “At what point  $(u, v)$  does the surface achieve its local maximum?”, and attempts to zoom in for clarity. Finally, it visually estimates the peak coordinates.

\*\*Answer:\*\* **(0,0)**

#### Response from Gemini-2.5-Pro-Preview (summarized)

The video displays a 3D surface, which appears to be an elliptic paraboloid opening downwards. The question asks for the point  $(u, v)$  where the surface achieves its local maximum.

The axes are labeled:

- One horizontal axis is labeled ‘u’.
- The other horizontal axis is labeled ‘v’.
- The vertical axis is labeled ‘ $f(u,v)$ ’ (implied by the function notation at the top, though only  $(v,u)f$  is visible).

A local maximum is a point on the surface that is higher than all its immediate neighbors. For a smooth, downward-opening paraboloid like the one shown, the local maximum is unique and occurs at its vertex.

By visually inspecting the graph, the vertex (the highest point) of the paraboloid is located directly above the origin of the  $u-v$  plane. The origin is where the  $u$ -axis and  $v$ -axis intersect, which corresponds to  $u = 0$  and  $v = 0$ .

**Therefore, the surface achieves its local maximum at the point  $(0, 0)$ .**

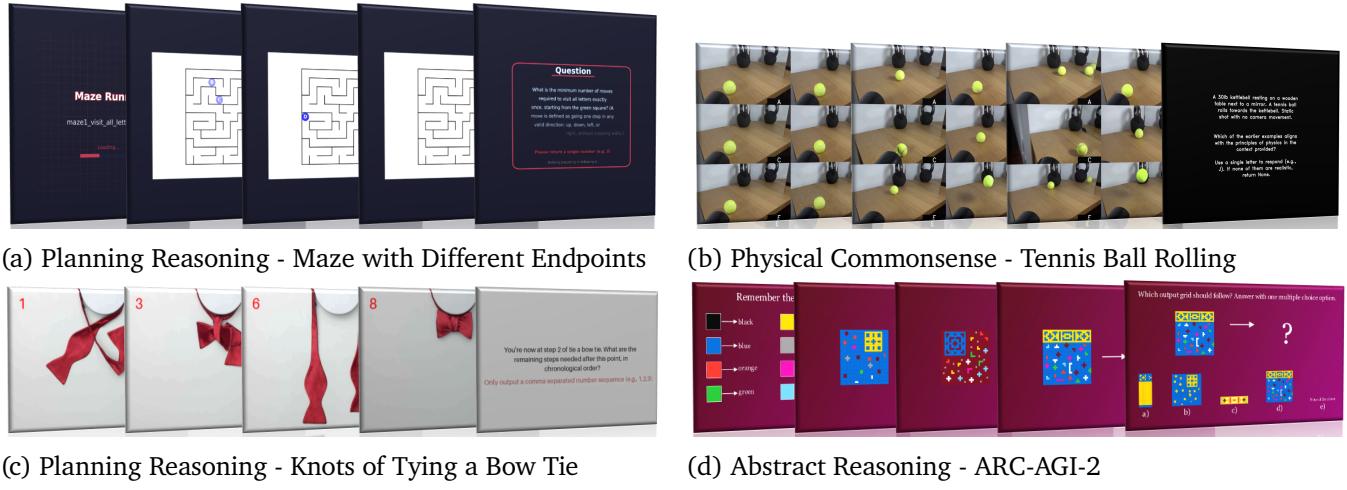
### 3.4. Analysis

**Performance on Challenging Tasks** To illustrate the complexity of the tasks evaluated, a two-by-two grid of example images from each category—Mazes, Rope Knots, Physical Commonsense, and ARC-AGI-2—is presented in 7. These tasks are designed to push the limits of current models, particularly those relying on image-based inputs. For Mazes, we assess the model’s ability to determine the optimal path from start to finish while navigating obstacles, count the number of viable paths, identify holes, and calculate the connected area of holes. The task is intensified by restricting the visible area around the agent, making it dependent on temporal information and increasing difficulty. In Rope Knots, we randomize the sequence of tying steps and challenge the model to predict the remaining steps, deduce the order after partial completion, and reconstruct the full sequence, with added image transformations to heighten complexity. For Physical Commonsense, we generate videos from initial images and scene descriptions using state-of-the-art image-to-video models, blending these with real camera-captured footage. The model must then distinguish the physically realistic video, testing its sensitivity to subtle physical inconsistencies. Finally, for ARC-AGI-2, we adapt the original tasks for vision-language models, presenting before-and-after patterns and either asking the model to select the correct completion image (if any) or count specific colored cells in the solution, posing a significant challenge due to abstract reasoning demands.

Table 3 reveals the performance of different models across these tasks. Humans achieve the highest scores, with 58.3 in Mazes, 53.0 in Rope Knots, 63.6 in Physical Commonsense, and 47.1 in ARC-AGI-2, reflecting their robust reasoning and contextual understanding. In contrast, existing frontier models lag considerably: o3 scores 10.0, 3.8, 13.6, and 14.0 across the respective tasks, while Gemini 2.5 Pro records near-zero performance (0.0 to 4.2). This stark gap underscores that even the most advanced models struggle to match human-level proficiency, particularly in tasks requiring temporal reasoning, physical intuition, and abstract pattern recognition, highlighting a critical area for future model development.

Model	Mazes	Rope Knots	Physical Commonsense	ARC-AGI-2
Human	58.3	53.8	63.6	47.1
o3	10.0	3.8	13.6	14.0
Gemini 2.5 Pro	0.0	1.3	4.2	0.0

Table 3: Performance of top frontier models across 4 challenging tasks.



**Figure 7:** Example videos for challenging tasks: Mazes, Rope Knots, Physical Commonsense, and ARC-AGI-2.

**Impact of FPS and Max Frames on Video-to-Image Frame Sampling for Model Performance** To evaluate image-based models on video inputs, we analyzed how different sampling strategies, specifically varying frames per second (FPS) and maximum number of frames, affect model performance. We tested configurations with  $\text{FPS} \in \{1, 2, 4, 8\}$  and  $\text{max frames} \in \{16, 32, 64, 128\}$  for Gemini 2.5 Flash. As shown in Table 4, the configuration with  $\text{FPS} 2$  and  $32$  max frames achieves the highest overall performance, with an "All" score of 19.2. Increasing FPS and max frames beyond this point, such as to  $\text{FPS} 4$  with 64 frames or  $\text{FPS} 8$  with 128 frames, does not consistently improve performance and can even lead to slight declines (e.g., "All" score drops to 19.0 at  $\text{FPS} 8$ , max 128). This suggests a saturation point where additional frames may introduce noise or redundant information, particularly for tasks like Planning and Spatial reasoning, where scores remain low or inconsistent (e.g., Planning scores range from 1.0 to 4.0). For models with smaller context windows, the  $\text{FPS} 1$ , max 16 frames configuration yields a marginally lower "All" score of 18.4 but outperforms in Physical tasks (34.4), indicating robustness for resource-constrained settings. Based on these findings, we adopt  $\text{FPS} 2$  and max 32 frames as the default configuration, as highlighted in Table 4, balancing performance and computational efficiency.

FPS	Max Frames	All	Abstract	Math	Physical	Planning	Spatial	Temporal
1	16	18.4	17.2	23.8	34.4	4.0	19.4	17.5
2	32	19.2	9.4	35.7	28.1	1.0	24.1	18.8
4	64	19.2	17.2	29.8	26.6	3.0	21.3	21.2
8	128	19.0	15.6	29.8	28.1	1.0	21.3	22.5

**Table 4:** Performance of models with different FPS and max frames settings when converting videos to frames for image-based input. Analyses based on Gemini 2.5 Flash.

**Static vs. Temporal Input.** To investigate how temporal structure affects multimodal reasoning, we conduct an ablation study on the MathVista dataset using the Qwen-2.5 VL model series under three input settings: (1) a single image paired with textual question (the original setting), (2) a sequence of static images simulating temporal progression (with textual questions encoded in images), and (3) a full video input (as a mp3 file format). We construct multiple images from the original image and question pair by putting the

question text on images that are the same size as the original. Then, each of these is either passed into the model separately (2) or stitched together into video frames with 1 fps (3).

Results are summarized in Table 5. These results reveal a consistent decline in performance as input complexity increases. The models performs best with static image-text pairs and degrades when required to reason over temporal sequences. This highlights a critical limitation of current VLMs: while optimized for joint spatial encoding, they remain brittle when handling distributed temporal information.

Table 5: Ablation study on the MathVista dataset showing the effect of different input modalities on Qwen-2.5 VL 7B, 32B<sup>AWQ</sup>, and 72B<sup>AWQ</sup>. Static inputs perform better than temporally distributed ones.

Input Modality	Qwen-2.5 VL 7B	Qwen-2.5 VL 32B	Qwen-2.5 VL 72B
Image + Question Text	62.4	72.1	69.0
Multi-Image Context	57.5	64.2	65.1
Video Input	57.3	60.9	62.8

## 4. Related Work

**Reasoning Benchmarks.** Most existing multimodal reasoning benchmarks operate on static images or short text-image pairs. Early static VQA datasets (e.g., TextVQA [Singh et al., 2019], DocVQA [Mathew et al., 2021], OCR-VQA [Mishra et al., 2019]) evaluate text extraction but largely test surface-level perception. More recent efforts broaden the scope to mathematical (MathVista [Lu et al., 2024], Math-Vision [Wang et al., 2024a], DynaMath [Zou et al., 2024]), commonsense (VisualCOMET [Park et al., 2020]), and multi-disciplinary exam reasoning (MMMU [Yue et al., 2023], ScienceQA [Lu et al., 2022]). Nonetheless these datasets share three structural limitations: (i) they are image-centric and ignore temporal dynamics; (ii) they concentrate on either knowledge retrieval or a single reasoning faculty (e.g., arithmetic) leading to skewed skill coverage; and (iii) their manually curated question banks saturate quickly, offering little head-room for diagnosing new models. Beyond supervised corpora, synthetic abstract-reasoning suites such as ARC-AGI-2 [Chollet et al., 2025] and CLEVRER [Johnson et al., 2017] test compositional and causal inference, yet they remain confined to toy worlds and still images. In sum, a unified benchmark that spans diverse reasoning types and evaluates robustness under temporally unfolding evidence is still absent. MORSE-500 fills this gap by providing programmatically generated video tasks across six reasoning categories with scalable difficulty.

**Video Understanding Benchmarks.** Video-centric datasets have traditionally emphasized descriptive understanding—e.g., action recognition or dense captioning—rather than explicit reasoning. Recent long-context suites such as HourVideo [Chandrasegaran et al., 2024] and LongVideoBench [Wu et al., 2024] introduce hour-long or movie-length clips but probe summarization and retrieval rather than isolating causal or mathematical inference. EMMA [Hao et al., 2025] adds multi-choice questions over short sequences but still relies on crowd-sourced videos with fixed difficulty. Large-scale quantitative benchmarks like Video-MME [Fu et al., 2024] and Video-MMMU [Hu et al., 2025] evaluate factual knowledge in videos; however many tasks can be answered correctly by sampling a single salient frame, blurring the line between perception and reasoning. Similarly, MVBench [Li et al., 2024] focuses on content understanding with limited task diversity. Mementos [Wang et al., 2024b] moves towards temporal reasoning but lacks programmatic control to escalate complexity as systems improve.

By contrast, MORSE-500 synthesizes video clips via deterministic scripts, allowing precise modulation of

motion speed, distractor density, and reasoning depth. This design avoids the “single-frame shortcut” and provides a living benchmark—new, harder instances can be generated on demand to keep pace with rapidly advancing VLMs.

## 5. Conclusion and Discussion

We introduce **MORSE-500**, a programmatically generated, video-centric benchmark that systematically evaluates six core forms of multimodal reasoning: mathematical, abstract, spatial, temporal, physical, and planning. Unlike existing benchmarks, which are often static, narrow in scope, or quickly saturate, MORSE-500 offers fine-grained control over difficulty, scene complexity, and distractor patterns, enabling stress tests that remain relevant as models improve. By embedding questions directly into temporally unfolding video narratives, our benchmark isolates reasoning performance from perception or retrieval shortcuts, providing a more faithful diagnostic lens for genuine multimodal intelligence.

Our evaluation reveals a critical gap in current AI capabilities: even state-of-the-art models lag dramatically behind human performance across all reasoning categories, with particularly concerning deficits in planning tasks (0-5% accuracy). These systematic failures indicate fundamental architectural limitations rather than scaling issues, suggesting that achieving human-level multimodal intelligence requires advances in temporal memory systems, compositional reasoning mechanisms, and hierarchical planning capabilities. MORSE-500’s programmatic foundation positions it as both a diagnostic tool and research catalyst—its scalable difficulty generation prevents benchmark saturation while enabling targeted study of specific reasoning failures. The benchmark’s extensibility to additional modalities and its grounding in cognitive science frameworks make it well-suited for driving the development of next-generation AI systems capable of flexible, robust reasoning in dynamic environments. By releasing the complete framework, we aim to accelerate progress toward AI systems that genuinely understand and reason about our complex world rather than merely matching patterns in static data.

## 6. Limitations and Broader Impact

**Limitations.** While MORSE-500 offers a richly structured and extensible framework for video-based reasoning evaluation, several limitations warrant acknowledgment. First, all videos are generated programmatically or curated from constrained domains, which may limit the visual diversity and realism found in real-world footage. Although we incorporate generative models and curated clips to enhance visual fidelity, the scenarios may not fully reflect the complexity of natural human environments with their inherent unpredictability and contextual richness. Second, our benchmark currently focuses exclusively on visual reasoning without incorporating audio modalities. While questions are embedded within video content, the absence of speech, ambient sounds, or audio-visual synchronization tasks represents a significant gap, as real-world reasoning often requires multimodal integration across sensory channels. Future extensions could incorporate audio cues, spoken questions, or audio-visual reasoning tasks to provide more comprehensive evaluation. Third, while our benchmark focuses on isolating reasoning from perception, complete separation remains challenging. Tasks involving complex visual scenes, occlusion, or visual clutter may conflate reasoning failures with perceptual limitations, making it difficult to attribute poor performance to specific cognitive deficits. Additionally, the programmatic generation approach, while ensuring reproducibility, may inadvertently create artificial regularities that differ from the statistical patterns models encounter in natural training data.

**Broader Impact.** MORSE-500 is designed to rigorously evaluate the reasoning capabilities of large multimodal models (LMMs), particularly in dynamic, temporally grounded contexts. By providing a scalable and reproducible stress-test, MORSE-500 may help identify reasoning deficits in current LMMs, thereby informing safer and more robust deployment in high-stakes domains such as robotics, education, and healthcare. At the same time, misuse of powerful video reasoning models—such as for surveillance, disinformation, or behavioral manipulation—remains a broader societal risk. By releasing MORSE-500 with full transparency, reproducible code, and benchmark harnesses, we aim to promote responsible development and auditing of multimodal AI systems.

## Acknowledgments

This work was supported by DARPA Transfer from Imprecise and Abstract Models to Autonomous Technologies (TIAMAT) 80321, DARPA HR001124S0029-AIQ-FP-019, National Science Foundation NSF-IIS-2147276 FAI, DOD-AFOSR-Air Force Office of Scientific Research under award number FA9550-23-1-0048, National Science Foundation NAIIR240045, ONR MURI program, the National Science Foundation (IIS-2212182), and the NSF TRAILS Institute (2229885). Private support was provided by Capital One Bank, the Amazon Research Award program, Open Philanthropy, and Peraton. We thank Kaiyu Yue, Yuxin Wen, Vasu Singla, Yuhang Zhou and other members from FurongLab and TomLab for the helpful discussions.

## References

- Llava-next: Open large multimodal models. 2024. URL <https://github.com/LLaVA-VL/LLaVA-NeXT>.
- Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022. URL <https://arxiv.org/abs/2204.14198>.
- Shikhar Bahl, Russell Mendonca, Lili Chen, Unnat Jain, and Deepak Pathak. Affordances from human videos as a versatile representation for robotics, 2023. URL <https://arxiv.org/abs/2304.08488>.
- Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*, 2023. URL <https://arxiv.org/abs/2308.12966>.
- Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. URL <https://arxiv.org/abs/1606.01540>.
- John Bissell Carroll. *Human cognitive abilities: A survey of factor-analytic studies.* Number 1. Cambridge university press, 1993. URL <https://www.cambridge.org/core/books/human-cognitive-abilities/F83D5EADF14A453F6350FF3DD39631C8>.
- Keshigeyan Chandrasegaran, Agrim Gupta, Lea M Hadzic, Taran Kota, Jimming He, Cristóbal Eyzaguirre, Zane Durante, Manling Li, Jiajun Wu, and Fei-Fei Li. Hourvideo: 1-hour video-language understanding. *Advances in Neural Information Processing Systems*, 37:53168–53197, 2024. URL <https://arxiv.org/abs/2411.04998>.

- François Chollet. On the measure of intelligence. [arXiv preprint arXiv:1911.01547](https://arxiv.org/abs/1911.01547), 2019. URL <https://arxiv.org/abs/1911.01547>.
- François Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. Arc-agi-2: A new challenge for frontier ai reasoning systems. [arXiv preprint arXiv:2505.11831](https://arxiv.org/abs/2505.11831), 2025. URL <https://arxiv.org/abs/2505.11831>.
- Wei Chow, Jiageng Mao, Boyi Li, Daniel Seita, Vitor Guizilini, and Yue Wang. Physbench: Benchmarking and enhancing vision-language models for physical world understanding. [arXiv preprint arXiv:2501.16411](https://arxiv.org/abs/2501.16411), 2025. URL <https://arxiv.org/abs/2501.16411>.
- Stanislas Dehaene. *The number sense: How the mind creates mathematics*. OUP USA, 2011.
- Jonathan St B. T. Evans and Keith E. Stanovich. Dual-process theories of higher cognition: Advancing the debate. *Perspectives on psychological science*, 8(3):223–241, 2013. doi: 10.1177/1745691612460685. URL <https://doi.org/10.1177/1745691612460685>.
- Chaoyou Fu, Yuhang Dai, Yongdong Luo, Lei Li, Shuhuai Ren, Renrui Zhang, Zihan Wang, Chenyu Zhou, Yunhang Shen, Mengdan Zhang, et al. Video-mme: The first-ever comprehensive evaluation benchmark of multi-modal llms in video analysis. [arXiv preprint arXiv:2405.21075](https://arxiv.org/abs/2405.21075), 2024. URL <https://arxiv.org/abs/2405.21075>.
- Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. [arXiv preprint arXiv:2503.19786](https://arxiv.org/abs/2503.19786), 2025. URL <https://arxiv.org/abs/2503.19786>.
- Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170, 1983. doi: 10.1207/s15516709cog0702\_3. URL [https://doi.org/10.1207/s15516709cog0702\\_3](https://doi.org/10.1207/s15516709cog0702_3).
- Anastasis Germanidis. Introducing gen-3 alpha: A new frontier for video generation. <https://runwayml.com/research/introducing-gen-3-alpha>, 2024. Runway Research blog, Accessed 22 May 2025.
- Google. Our next-generation model: Gemini 1.5. <https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/>, 2024. Accessed: 2025-05-16.
- Google. Gemini 2.5: Our most intelligent ai model. <https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/>, 2025. Accessed: 2025-05-16.
- Google DeepMind. State-of-the-art video and image generation with veo 2 and imagen 3. <https://blog.google/technology/google-labs/video-image-generation-update-december-2024/>, 2024. Accessed 22 May 2025.
- Grog. Animated knots. <https://www.animatedknots.com>, 2025. Accessed: 2025-05-16.
- Hailuo AI. Transform idea to visual with ai. <https://hailuoai.video/>, 2025. Accessed 22 May 2025.
- Siddhant Haldar, Vaibhav Mathur, Denis Yarats, and Lerrel Pinto. Watch and match: Supercharging imitation with regularized optimal transport, 2023. URL <https://arxiv.org/abs/2206.15469>.

Yunzhuo Hao, Jiawei Gu, Huichen Will Wang, Linjie Li, Zhengyuan Yang, Lijuan Wang, and Yu Cheng. Can mllms reason in multimodality? emma: An enhanced multimodal reasoning benchmark. In Proceedings of the 42nd International Conference on Machine Learning (ICML 2025), 2025. URL <https://arxiv.org/abs/2501.05444>.

Keith J. Holyoak and Robert G. Morrison, editors. The Oxford handbook of thinking and reasoning. Oxford University Press, 2013. ISBN 9780199730013. URL <https://academic.oup.com/edited-volume/34559>.

Kairui Hu, Penghao Wu, Fanyi Pu, Wang Xiao, Yuanhan Zhang, Xiang Yue, Bo Li, and Ziwei Liu. Videommu: Evaluating knowledge acquisition from multi-discipline professional videos. arXiv preprint arXiv:2501.13826, 2025. URL <https://arxiv.org/abs/2501.13826>.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. arXiv preprint arXiv:2410.21276, 2024. URL <https://arxiv.org/abs/2410.21276>.

Michael Igorevich Ivanitskiy, Rusheb Shah, Alex F. Spies, Tilman Räuker, Dan Valentine, Can Rager, Lucia Quirke, Chris Mathwin, Guillaume Corlouer, Cecilia Diniz Behn, and Samy Wu Fung. A configurable library for generating and manipulating maze datasets, 2023. URL <https://arxiv.org/abs/2309.10498>.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. arXiv preprint arXiv:2412.16720, 2024. URL <https://arxiv.org/abs/2412.16720>.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2901–2910, 2017. doi: 10.1109/CVPR.2017.311. URL [https://openaccess.thecvf.com/content\\_cvpr\\_2017/html/Johnson\\_CLEVR\\_A\\_Diagnostic\\_CVPR\\_2017\\_paper.html](https://openaccess.thecvf.com/content_cvpr_2017/html/Johnson_CLEVR_A_Diagnostic_CVPR_2017_paper.html).

Kling AI. Kling ai: Next-generation ai creative studio. <https://klingai.com/>, 2025. Accessed 22 May 2025.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In Proceedings of the 40th International Conference on Machine Learning (ICML 2023), volume 202 of Proceedings of Machine Learning Research, pages 19109–19133. PMLR, jul 2023. URL <https://arxiv.org/abs/2301.12597>.

Kunchang Li, Yali Wang, Yinan He, Yizhuo Li, Yi Wang, Yi Liu, Zun Wang, Jilan Xu, Guo Chen, Ping Luo, et al. Mvbench: A comprehensive multi-modal video understanding benchmark. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 22195–22206, 2024. doi: 10.1109/CVPR52733.2024.02049. URL [https://openaccess.thecvf.com/content/CVPR2024/html/Li\\_MVBench\\_A\\_Comprehensive\\_Multi-modal\\_Video\\_Understanding\\_Benchmark\\_CVPR\\_2024\\_paper.html](https://openaccess.thecvf.com/content/CVPR2024/html/Li_MVBench_A_Comprehensive_Multi-modal_Video_Understanding_Benchmark_CVPR_2024_paper.html).

Pan Lu, Michel Galley, Hao Cheng, Kai-Wei Chang, and Jianfeng Gao. Learn from science textbooks: Retrieval-augmented science question answering. arXiv preprint arXiv:2207.05275, 2022. URL <https://arxiv.org/abs/2207.05275>.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations (ICLR 2024)*, 2024. URL <https://arxiv.org/abs/2310.02255>.

Manim Community. Manim – mathematical animation framework, 2025. URL <https://zenodo.org/records/14699705>.

Minesh Mathew, Dimosthenis Karatzas, and C. V. Jawahar. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2200–2209, 2021. doi: 10.1109/WACV48630.2021.00224. URL [https://openaccess.thecvf.com/content/WACV2021/html/Mathew\\_DocVQA\\_A\\_Dataset\\_for\\_VQA\\_on\\_Document\\_Images\\_WACV\\_2021\\_paper.html](https://openaccess.thecvf.com/content/WACV2021/html/Mathew_DocVQA_A_Dataset_for_VQA_on_Document_Images_WACV_2021_paper.html).

Michael McCloskey, Allyson Washburn, and Linda Felch. Intuitive physics: the straight-down belief and its origin. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 9(4):636–649, 1983. doi: 10.1037/0278-7393.9.4.636. URL <https://doi.org/10.1037/0278-7393.9.4.636>.

Ankush Mishra, Shashank Shekhar, Ajeet Kumar Singh, and Anirban Chakraborty. Ocr-vqa: Visual question answering by reading text in images. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 947–952. IEEE, 2019.

Akira Miyake, Naomi P. Friedman, Michael J. Emerson, Alexander H. Witzki, Amy Howerter, and Tor D. Wager. The unity and diversity of executive functions and their contributions to complex “frontal lobe” tasks: A latent variable analysis. *Cognitive psychology*, 41(1):49–100, 2000. doi: 10.1006/cogp.1999.0734. URL <https://doi.org/10.1006/cogp.1999.0734>.

Saman Motamed, Laura Culp, Kevin Swersky, Priyank Jaini, and Robert Geirhos. Do generative video models understand physical principles? *arXiv preprint arXiv:2501.09038*, 2025. URL <https://arxiv.org/abs/2501.09038>.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. URL <https://arxiv.org/abs/2303.08774>.

OpenAI. Sora is here. <https://openai.com/index/sora-is-here/>, 2024. Accessed 22 May 2025.

OpenAI. Introducing o3 and o4-mini. <https://openai.com/index/introducing-o3-and-o4-mini/>, 2025. Accessed: 2025-05-16.

Jae Sung Park, Dong Huk Kim, Rowan Zellers, Yonatan Bisk, Ani Kembhavi, and Yejin Choi. Visualcomet: Reasoning about the dynamic context of a still image. In *European Conference on Computer Vision (ECCV)*, pages 508–524. Springer, 2020. URL <https://arxiv.org/abs/2004.10796>.

Shi Qiu, Shaoyang Guo, Zhuo-Yang Song, Yunbo Sun, Zeyu Cai, Jiashen Wei, Tianyu Luo, Yixuan Yin, Haoxu Zhang, Yi Hu, et al. Phybench: Holistic evaluation of physical perception and reasoning in large language models. *arXiv preprint arXiv:2504.16074*, 2025. URL <https://arxiv.org/abs/2504.16074>.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023. URL <https://arxiv.org/abs/2303.17580>.

Roger N. Shepard and Jacqueline Metzler. Mental rotation of three-dimensional objects. *Science*, 171(3972):701–703, 1971. doi: 10.1126/science.171.3972.701. URL <https://doi.org/10.1126/science.171.3972.701>.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning (CoRL 2022)*, 2023. URL <https://proceedings.mlr.press/v205/shridhar23a.html>. arXiv preprint year 2022.

Amanpreet Singh, Vivek Natarajan, Yu Jiang, Xinlei Chen, Marcus Rohrbach, Dhruv Batra, and Devi Parikh. Textvqa: Towards image-text reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8317–8326, 2019. doi: 10.1109/CVPR.2019.00852. URL [https://openaccess.thecvf.com/content\\_CVPR\\_2019/html/Singh\\_Towards\\_VQA\\_Models\\_That\\_Can\\_Read\\_CVPR\\_2019\\_paper.html](https://openaccess.thecvf.com/content_CVPR_2019/html/Singh_Towards_VQA_Models_That_Can_Read_CVPR_2019_paper.html). Consolidated entry for TextVQA CVPR paper.

Wan Team, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025. URL <https://arxiv.org/abs/2503.20314>.

Chen Wang, Linxi Fan, Jiankai Sun, Ruohan Zhang, Li Fei-Fei, Danfei Xu, Yuke Zhu, and Anima Anandkumar. Mimicplay: Long-horizon imitation learning by watching human play, 2023. URL <https://arxiv.org/abs/2302.12422>.

Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Houxing Ren, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Measuring multimodal mathematical reasoning with math-vision dataset. *Advances in Neural Information Processing Systems*, 37:95095–95169, 2024a. URL <https://arxiv.org/abs/2402.14804>.

Xiyao Wang, Yuhang Zhou, Xiaoyu Liu, Hongjin Lu, Yuancheng Xu, Feihong He, Jaehong Yoon, Taixi Lu, Gedas Bertasius, Mohit Bansal, et al. Mementos: A comprehensive benchmark for multimodal large language model reasoning over image sequences. *arXiv preprint arXiv:2401.10529*, 2024b. URL <https://arxiv.org/abs/2401.10529>.

Haoning Wu, Dongxu Li, Bei Chen, and Junnan Li. Longvideobench: A benchmark for long-context interleaved video-language understanding. *Advances in Neural Information Processing Systems*, 37:28828–28857, 2024. URL <https://arxiv.org/abs/2407.15754>.

Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, nov 2023. ISSN 1573-7527. doi: 10.1007/s10514-023-10139-z. URL <https://doi.org/10.1007/s10514-023-10139-z>.

Kun Wu, Chengkai Hou, Jiaming Liu, Zhengping Che, Xiaozhu Ju, Zhuqin Yang, Meng Li, Yinuo Zhao, Zhiyuan Xu, Guang Yang, Shichao Fan, Xinhua Wang, Fei Liao, Zhen Zhao, Guangyu Li, Zhao Jin, Lecheng Wang, Jilei Mao, Ning Liu, Pei Ren, Qiang Zhang, Yaoxu Lyu, Mengzhen Liu, Jingyang He, Yulin Luo, Zeyu Gao, Chenxuan Li, Chenyang Gu, Yankai Fu, Di Wu, Xingyu Wang, Sixiang Chen, Zhenyu Wang, Pengju An, Siyuan Qian, Shanghang Zhang, and Jian Tang. Robomind: Benchmark on multi-embodiment intelligence normative data for robot manipulation, 2025. URL <https://arxiv.org/abs/2412.13877>.

Kun Xiang, Heng Li, Terry Jingchen Zhang, Yinya Huang, Zirong Liu, Peixin Qu, Jixi He, Jiaqi Chen, Yu-Jie Yuan, Jianhua Han, et al. Seephys: Does seeing help thinking?—benchmarking vision-based physics reasoning. *arXiv preprint arXiv:2505.19099*, 2025. URL <https://arxiv.org/abs/2505.19099>.

Jin Xu, Zhifang Guo, Jinzheng He, Hangrui Hu, Ting He, Shuai Bai, Keqin Chen, Jialin Wang, Yang Fan, Kai Dang, et al. Qwen2. 5-omni technical report. [arXiv preprint arXiv:2503.20215](https://arxiv.org/abs/2503.20215), 2025. URL <https://arxiv.org/abs/2503.20215>.

Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, et al. Minicpm-v: A gpt-4v level mllm on your phone. [arXiv preprint arXiv:2408.01800](https://arxiv.org/abs/2408.01800), 2024. URL <https://arxiv.org/abs/2408.01800>.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. [arXiv preprint arXiv:2311.16502](https://arxiv.org/abs/2311.16502), 2023. URL <https://arxiv.org/abs/2311.16502>.

Jeffrey M. Zacks, Nicole K. Speer, Khena M. Swallow, Todd S. Braver, and Jeremy R. Reynolds. Event segmentation in perception and memory. *Trends in Cognitive Sciences*, 11(2):80–86, 2007. doi: 10.1016/j.tics.2006.12.001. URL <https://doi.org/10.1016/j.tics.2006.12.001>.

Haotian Zhang, Jingyang Li, Zhiwei Li, Zhang Kanchor, Zhengkai Liu, Chen-Yu Lee, Chunyuan Su, Chunyuan Li, Percy Liang, and Ahmed H. Awadallah. Multimodal chain-of-thought reasoning in language models. [arXiv preprint arXiv:2302.00923](https://arxiv.org/abs/2302.00923), 2023. URL <https://arxiv.org/abs/2302.00923>.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Yuchen Duan, Hao Tian, Weijie Su, Jie Shao, Zhangwei Gao, Erfei Cui, Yue Cao, Yangzhou Liu, Weiye Xu, Hao Li, Jiahao Wang, Han Lv, Dengnian Chen, Songze Li, Yinan He, Tan Jiang, Jiapeng Luo, Yi Wang, Conghui He, Botian Shi, Xingcheng Zhang, Wenqi Shao, Junjun He, Yingtong Xiong, Wenwen Qu, Peng Sun, Penglong Jiao, Lijun Wu, Kaipeng Zhang, Huirong Deng, Jiaye Ge, Kai Chen, Limin Wang, Min Dou, Lewei Lu, Xizhou Zhu, Tong Lu, Dahua Lin, Yu Qiao, Jifeng Dai, and Wenhui Wang. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. [arXiv preprint arXiv:2504.10479](https://arxiv.org/abs/2504.10479), 2025. URL <https://arxiv.org/abs/2504.10479>.

Yifeng Zhu, Abhishek Joshi, Peter Stone, and Yuke Zhu. Viola: Imitation learning for vision-based manipulation with object proposal priors, 2023. URL <https://arxiv.org/abs/2210.11339>.

Chengke Zou, Xingang Guo, Rui Yang, Junyu Zhang, Bin Hu, and Huan Zhang. Dynamath: A dynamic visual benchmark for evaluating mathematical reasoning robustness of vision language models. [arXiv preprint arXiv:2411.00836](https://arxiv.org/abs/2411.00836), 2024. URL <https://arxiv.org/abs/2411.00836>.

## Contents

<b>A Benchmark Overview &amp; Multi-modal Reasoning Taxonomy</b>	<b>27</b>
<b>B Example Question and Code</b>	<b>29</b>
B.1 Abstract Reasoning . . . . .	29
B.2 Mathematical Reasoning . . . . .	34
B.3 Physical Reasoning . . . . .	38
B.4 Planning Reasoning . . . . .	45
B.5 Spatial Reasoning . . . . .	50
B.6 Temporal Reasoning . . . . .	54
<b>C More Qualitative Results</b>	<b>59</b>
C.1 Abstract Reasoning . . . . .	59
C.2 Mathematical Reasoning . . . . .	64
C.3 Physical Reasoning . . . . .	66
C.4 Planning Reasoning . . . . .	74
C.5 Spatial Reasoning . . . . .	77
C.6 Temporal Reasoning . . . . .	81

## A. Benchmark Overview & Multi-modal Reasoning Taxonomy

We construct MORSE-500 around six major categories of multi-modal reasoning, each designed with specific task instantiations and complexity controls, as detailed in Table A.1.

Our approach combines established benchmarks with novel adaptations to create challenging multi-modal scenarios that require both visual understanding and temporal reasoning.

**Task Design Methodology.** For each reasoning category, we employ a systematic approach to task creation that balances coverage and difficulty. We adapt existing datasets where appropriate (e.g., maze navigation [Ivanitskiy et al., 2023, Brockman et al., 2016], ARC-AGI [Chollet, 2019, Chollet et al., 2025]), introduce environmental challenges (e.g., fog effects, visual noise), and curate real-world scenarios (e.g., robotic manipulation sequences and rope knot tying procedures). Complexity is controlled through multiple dimensions specific to each category, including structural parameters (number of objects, plan length), environmental factors (visual noise, temporal irregularity), and cognitive demands (rule depth, interaction complexity).

**Novel Adaptations.** Several categories feature innovative task designs. For planning reasoning, we introduce fog effects to standard maze and FrozenLake environments [Brockman et al., 2016] and curate action sequence recognition tasks from robotic manipulation videos [Zhu et al., 2023, Wu et al., 2025, Haldar et al., 2023, Wu et al., 2023, Wang et al., 2023, Bahl et al., 2023]. Physical reasoning incorporates a novel real-vs-generated video discrimination task that leverages the improving quality of video generation models. We use the videos in Physics IQ Benchmark Motamed et al. [2025] as real footage, we use image-to-video models, including Sora [OpenAI, 2024], Runnway Gen-3 [Germanidis, 2024], Kling 1.6 [Kling AI, 2025], Hailuo AI [Hailuo AI, 2025], Wan 2.1 [Wan Team et al., 2025], and Veo2 [Google DeepMind, 2024] to generate 5s videos conditioned on one video frame and textual description on the object and camera motion. Abstract reasoning repurposes ARC-AGI patterns into multiple-choice and free-form visual reasoning questions based on color patterns and spatial arrangements.

### Reasoning Category Description

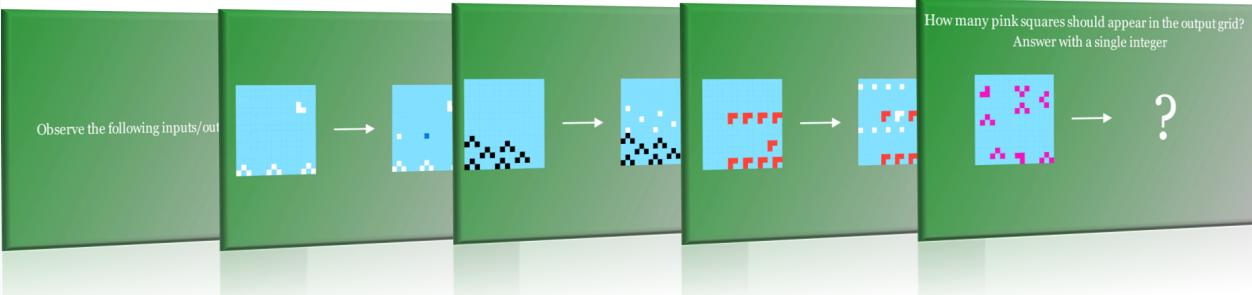
Abstract Reasoning (12.8%)	It targets <i>pattern recognition</i> , <i>logical inference</i> , and <i>symbolic reasoning</i> through adapted benchmarks and novel task designs. For logical reasoning, we repurpose ARC-AGI 2 patterns into multiple-choice questions and free-form responses based on color patterns, spatial arrangements, and cell counts, requiring rule induction from minimal examples and pattern extrapolation under abstract transformations. For symbolic reasoning, we design novel tasks including anagram word transformations, symbolic equation solving, and visual-textual symbol mapping challenges. Complexity is controlled by number of visual elements (4-25 cells), rule depth (1-4 nested transformations), color diversity (2-10 colors), symbolic abstraction level, and cross-modal symbol correspondence requirements.
Mathematical Reasoning (16.8%)	It evaluates <i>arithmetic operations</i> , <i>algebraic relations</i> , and <i>quantitative comparisons</i> through visual and textual integration. Tasks include dynamic word problems with chart interpretation, visual equation solving with geometric constraints, proportional comparisons across multiple data modalities, and geometric reasoning requiring spatial-numerical synthesis. Complexity is controlled through variable count (2-8 unknowns), operation depth (1-4 nested operations), visual noise levels, and cross-modal information density.
Physical Reasoning (12.8%)	It tests understanding of <i>object dynamics</i> and <i>causal interactions</i> governed by physical laws through simulation and real-world discrimination. Tasks involve predicting structural collapse in block towers, estimating force effects on object motion, forecasting collision trajectories, and a novel real-vs-generated video discrimination task where models must identify authentic physics from increasingly sophisticated generated alternatives. Complexity is determined by interaction complexity (1-8 interacting objects), diversity of physical principles (gravity, friction, momentum, elasticity), and realism of generated distractors.
Planning Reasoning (20.0%)	It emphasizes <i>multi-step</i> , <i>goal-directed reasoning</i> through adapted environments and real-world scenarios. Tasks include maze navigation with fog effects (adapted from standard maze datasets), FrozenLake traversal under partial observability, robotic manipulation sequence recognition from curated online videos, and rope-tying action ordering tasks. We test action sequencing, goal inference, and sequential decision-making with complexity defined by plan length (3-15 steps), environmental uncertainty (fog density, partial observability), constraint density, and branching factor (2-6 alternative paths).
Spatial Reasoning (21.6%)	It tests understanding of <i>object relationships</i> , <i>spatial transformations</i> , and <i>3D reasoning</i> across multiple viewpoints and reference frames. Tasks include mental rotation with occlusion handling, multi-view inference requiring perspective integration, spatial pathfinding through complex 3D environments, and relative positioning under dynamic transformations. Complexity is determined by number of objects (3-12), degree of transformation (0°-360° rotations), scene dimensionality (2D/3D), and presence of visual distractors.
Temporal Reasoning (16.0%)	It assesses <i>sequence understanding</i> and <i>causal inference</i> over time through multi-frame visual narratives and process documentation. Tasks include event reordering from shuffled image sequences, duration comparison across parallel processes, future-state prediction in dynamic scenes, and cause-effect relationship identification in temporal chains. Difficulty is influenced by temporal irregularity (non-uniform time intervals), number of concurrent events (1-5), sequence length (4-20 frames), and presence of temporal red herrings.

Table A.1: Definitions, task instantiations, and proportions of six multimodal reasoning categories in MORSE-500. Each category incorporates specific complexity controls and novel adaptations to create challenging multimodal scenarios.

## B. Example Question and Code

### B.1. Abstract Reasoning

#### Abstract Reasoning - ARC-AGI Video Question



**Query:** Answer the question in this video.

#### Question Text:

How many pink squares should appear in the output grid?

Answer with only one multiple choice option.

**Ground truth: 21**

#### Code

The python code summarization is pasted below, the full code is available on [github](#).

```

1 from manim import *
2 import json, random, copy
3 import os
4 from pathlib import Path
5 from collections import Counter
6 from itertools import chain
7
8 class ARCSpace(Scene):
9
10     def __init__(self, data_path, p_type):
11         super().__init__()
12         self.p_type = p_type
13         self.DATA_PATH = data_path # path to ARC json file
14
15         # ----- Visual constants -----
16         self.CELL_SIZE = 0.35
17         self.MAX_GRID = 3.0 # max side length after scaling for training grids
18         self.TOP_SCALE = 0.8 # additional shrink factor for test (top) grids
19         self.COLOR_TABLE = [
20             BLACK, # 0 - background / empty
21             ManimColor.from_hex("#0074D9"), # 1 - blue
22             ManimColor.from_hex("#FF4136"), # 2 - orange
23             ManimColor.from_hex("#2ECC40"), # 3 - green
24             ManimColor.from_hex("#FFDC00"), # 4 - yellow
25             ManimColor.from_hex("#AAAAAA"), # 5 - light gray
26             ManimColor.from_hex("#F012BE"), # 6 - pink
27             ManimColor.from_hex("#7FDBFF"), # 7 - light blue
28             ManimColor.from_hex("#870C25"), # 8 - dark red
29             WHITE, # 9 - white
30         ]

```

```

31     self.NUM_TO_STR = [
32         "black",
33         "blue",
34         "orange",
35         "green",
36         "yellow",
37         "light gray",
38         "pink",
39         "light blue",
40         "dark red",
41         "white",
42     ]
43     self.TRAIN_STAY = 1.5
44     self.TRANSITION = 0.5
45     self.TEST_BIG_STAY = 1.5 # how long to hold full-size test pair
46     self.TEST_STAY = 4.0 # after options appear
47
48 # -----
49 # Helpers
50 #
51 def grid_to_vgroup(self, grid):
52     """Convert a 2-D list of ints into a VGroup of colored squares."""
53     print(grid)
54     vg = VGroup()
55     for r in range(len(grid)):
56         for c in range(len(grid[r])):
57             val = grid[r][c]
58             val = val % 10
59             color = self.COLOR_TABLE[val]
60             sq = Square(side_length=self.CELL_SIZE, stroke_width=0)
61             sq.set_fill(color, opacity=1)
62             # top-left origin mapping
63             x = (c + 0.5) * self.CELL_SIZE
64             y = -(r + 0.5) * self.CELL_SIZE
65             sq.move_to([x, y, 0])
66             vg.add(sq)
67     # center on origin then return
68     vg.move_to(ORIGIN)
69     return vg
70
71 def scale_grid(self, vg, max_side):
72     """Scale *vg* so its larger dimension equals *max_side*."""
73     sf = min(max_side / vg.width, max_side / vg.height)
74     vg.scale(sf)
75     return vg
76
77 def load_task(self):
78     with open(self.DATA_PATH, "r") as fp:
79         task = json.load(fp)
80     train_pairs = [(s["input"], s["output"]) for s in task["train"]]
81     test_input = task["test"][0]["input"]
82     test_output = task["test"][0]["output"]
83     return task, train_pairs, test_input, test_output
84
85 def color_perturb(self, grid, p_changes=0.1):
86     g = copy.deepcopy(grid)
87     rows, cols = len(g), len(g[0])
88     n_changes = int(p_changes * rows * cols)
89     for _ in range(n_changes):
90         allowed = set()
91         while len(allowed) == 0:
92             allowed = set()
93             r, c = random.randrange(rows), random.randrange(cols)
94             orig = g[r][c]
95             dirs = [(-1, 0), (1, 0), (0, -1), (0, 1)] # up, down, left, right
96
97             for dr, dc in dirs:
98                 if 0 <= r+dr < rows and 0 <= c+dc < cols and grid[r+dr][c+dc] != orig:
99                     allowed.add(grid[r+dr][c+dc])
100
101     g[r][c] = random.choice(list(allowed))
102
103     return g
104
105 def show_colors(self, colors, names):
106
107     cols = VGroup()
108     all_squares = VGroup()
109     all_arrows = VGroup()

```

```

110     all_labels = VGroup()
111     for start in range(0, len(colors), 4):
112
113         squares = VGroup(
114             *[
115                 Square(1.0).set_fill(col, 1).set_stroke(color=WHITE, width=2)
116                 for col in colors[start : start + 4]
117             ]
118         )
119         squares.arrange(DOWN, buff=0.5, aligned_edge=LEFT).to_edge(LEFT, buff=2)
120         labels = VGroup(
121             *[Text(nn, font_size=32) for nm in names[start : start + 4]]
122         )
123         for sq, lbl in zip(squares, labels):
124             lbl.next_to(sq, RIGHT, buff=1.2)
125         arrows = VGroup(
126             *[
127                 Arrow(
128                     start=sq.get_right(),
129                     end=lbl.get_left(),
130                     buff=0.05,
131                     stroke_width=4,
132                 )
133                 for sq, lbl in zip(squares, labels)
134             ]
135         )
136         col_group = VGroup(squares, arrows, labels)
137         cols.add(col_group)
138         all_squares.add(*squares)
139         all_arrows.add(*arrows)
140         all_labels.add(*labels)
141
142     cols.arrange(buff=1.7, aligned_edge=UP).move_to(ORIGIN).scale_to_fit_width(
143         0.9 * config.frame_width
144     )
145
146     # Title
147     title = Text("Remember the following color names", font_size=40)
148     title.to_edge(UP)
149
150     # Animation sequence
151     self.play(Write(title))
152     self.play(
153         Succession(
154             FadeIn(all_squares),
155             AnimationGroup(*[GrowArrow(ar) for ar in all_arrows], lag_ratio=0.1),
156             AnimationGroup(*[Write(label) for label in all_labels]),
157         ),
158         run_time=1.5,
159     )
160     self.wait(2)
161     self.play(FadeOut(title, cols))
162     self.wait(0.5)
163
164     # ----- Main construct -----
165     def construct(self):
166         self.add(
167             Rectangle(height=config.frame_height, width=config.frame_width)
168             .set_color(color_gradient(random.sample(self.COLOR_TABLE, 2), 5))
169             .set_opacity(0.7)
170         )
171         self.show_colors(self.COLOR_TABLE, self.NUM_TO_STR)
172
173         prompt = Text(
174             "Observe the following inputs/outputs", color=WHITE, font_size=36
175         ).move_to(ORIGIN)
176         self.play(FadeIn(prompt), run_time=0.5)
177         self.wait(1.5)
178         self.play(FadeOut(prompt), run_time=0.5)
179         self.wait(1)
180         task_raw, train_pairs, test_input, test_output = self.load_task()
181
182         arrow_proto = Arrow(LEFT, RIGHT, color=WHITE, buff=0.2)
183         train_L = LEFT * 3
184         train_R = RIGHT * 3
185
186         # ----- Training slideshow -----
187         for inp, out in train_pairs:
188             lgrid = self.scale_grid(self.grid_to_vgroup(inp), self.MAX_GRID).move_to(

```

```

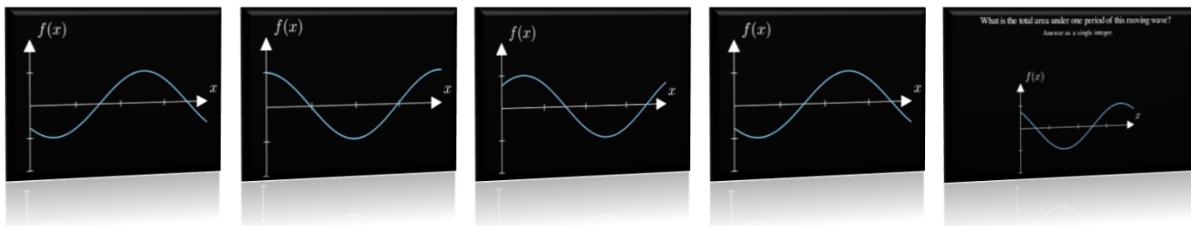
189         train_L
190     )
191     rgrid = self.scale_grid(self.grid_to_vgroup(out), self.MAX_GRID).move_to(
192         train_R
193     )
194     arr = arrow_proto.copy()
195     self.play(
196         FadeIn(lgrid, shift=DOWN * 0.2),
197         FadeIn(arr),
198         FadeIn(rgrid, shift=UP * 0.2),
199         run_time=self.TRANSITION,
200     )
201     self.wait(self.TRAIN_STAY)
202     self.play(
203         FadeOut(lgrid), FadeOut(arr), FadeOut(rgrid), run_time=self.TRANSITION
204     )
205
206 # ----- Test: show full-size pair -----
207 L_big_anchor = LEFT * 3
208 R_big_anchor = RIGHT * 3
209 test_in_big = self.scale_grid(
210     self.grid_to_vgroup(test_input), self.MAX_GRID
211 ).move_to(L_big_anchor)
212
213 blank_grid = [[0 for _ in row] for row in test_output]
214 blank_big = self.scale_grid(
215     self.grid_to_vgroup(blank_grid), self.MAX_GRID
216 ).move_to(R_big_anchor)
217 q_big = Text("?", font_size=160, color=WHITE).move_to(blank_big)
218 arrow_big = arrow_proto.copy()
219
220 self.play(
221     FadeIn(test_in_big),
222     FadeIn(arrow_big),
223     FadeIn(q_big),
224     run_time=self.TRANSITION,
225 )
226 self.wait(self.TEST_BIG_STAY)
227
228 ca = Counter(chain.from_iterable(test_input))
229 cb = Counter(chain.from_iterable(test_output))
230 changed = {n for n in set(ca) | set(cb) if ca[n] != cb[n]} - {0, 9}
231 color = random.choice(list(changed))
232 color_str = self.NUM_TO_STR[color]
233 title = f"How many {color_str} squares should appear in the output grid?\nAnswer with a
single integer"
234
235 lines = title.split("\n")
236
237 para = Paragraph(*lines, alignment="center", font_size=36, line_spacing=0.8)
238 para.to_edge(UP)
239 if para.width > 0.9 * config.frame_width:
240     para.scale_to_fit_width(config.frame_width * 0.9)
241 self.play(Write(para), run_time=1)
242
243 self.answer = sum(row.count(color) for row in test_output)
244 self.question_text = title.replace("\n", " ")
245
246
247 if __name__ == "__main__":
248     N_EXAMPLES = 20
249     p_type = "count" # {mc, count}
250     path_name = f"arcagi2_{p_type}"
251
252     os.makedirs(f"media/videos/1080p60/{path_name}/questions", exist_ok=True)
253     os.makedirs(f"media/videos/1080p60/{path_name}/solutions", exist_ok=True)
254     os.makedirs(f"media/videos/1080p60/{path_name}/question_text", exist_ok=True)
255
256     folder = Path("arcagi2")
257     random.seed(1)
258     paths = random.sample(list(folder.iterdir()), N_EXAMPLES)
259     for path in paths:
260         config.output_file = f"{path_name}/questions/{path.stem}"
261         scene = ARCScape(path, p_type)
262         scene.render()
263         with open(
264             f"media/videos/1080p60/{path_name}/solutions/{path.stem}.txt", "w"
265         ) as f:
266             f.write(str(scene.answer))

```

```
267     with open(
268         f"media/videos/1080p60/{path_name}/question_text/{path.stem}.txt", "w"
269     ) as f:
270         f.write(scene.question_text)
```

## B.2. Mathematical Reasoning

### Mathematical Reasoning - Area Under Fluctuating Sine Wave



**Query:** Answer the question in this video.

#### Question Text:

What is the total area under one period of this moving wave?

Please answer as a single integer.

**Ground truth: 4**

#### Code

The python code summarization is pasted below, the full code is available on github.

```

1 # question_wave_area.py
2 from manim import *
3 import numpy as np
4 import random
5 import os
6
7 class QuestionWaveArea(Scene):
8     def construct(self):
9         # ensure questions_text exists
10        os.makedirs("questions_text", exist_ok=True)
11
12        # 1) Randomize wave parameters
13        A = random.randint(1, 5)           # amplitude
14        B = random.choice([1, 2])          # frequency
15        C0 = random.uniform(0, 2 * np.pi) # initial phase offset
16        period = 2 * np.pi / B
17
18        # 2) Set up phase tracker for animation
19        phase = ValueTracker(0.0)
20        speed = 4.0 # units per second
21
22        # 3) Set up axes with ticks every 1/B * pi maybe, but x-range is one period
23        x_min, x_max = 0, period
24        axes = Axes(
25            x_range=[x_min, x_max, period/4],
26            y_range=[-A - 1, A + 1, 1],
27            x_length=6, y_length=4,
28            tips=True
29        ).to_edge(DOWN)
30        axes_labels = axes.get_axis_labels(x_label="x", y_label="f(x)")
31
32        # 4) Plot and animate the wave via updater
33        graph = axes.plot(lambda t: A * np.sin(B * (t - phase.get_value()) + C0),
34                           x_range=[x_min, x_max], color=BLUE)
35        def update_graph(mob, dt):
36            phase.increment_value(speed * dt)
37            new_graph = axes.plot(lambda t: A * np.sin(B * (t - phase.get_value()) + C0),
38                           x_range=[x_min, x_max], color=BLUE)

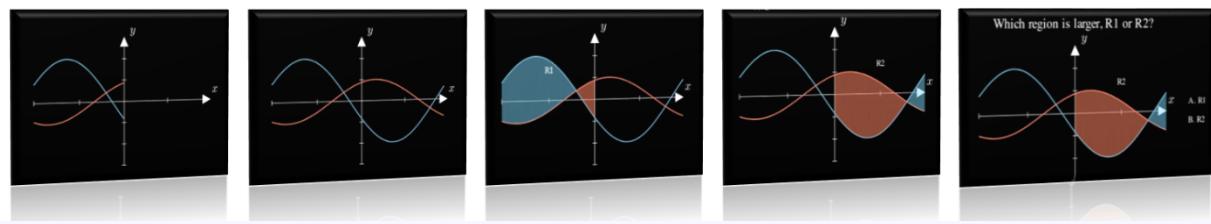
```

```

39         mob.become(new_graph)
40 graph.add_updater(update_graph)
41
42 self.add(axes, axes_labels, graph)
43 self.wait(1)
44
45 # 5) Compute the total area under one period
46 # Integral of |f(x)| dx over one period = 4A/B
47 correct = int(4 * A / B)
48
49 # 6) Save question prompt and answer
50 prompt = ("What is the total area under one period of this moving wave?"
51           )
52 with open("questions_text/question.txt", "w") as qf:
53     qf.write(prompt + "\nAnswer as a single integer.")
54 with open("answer.txt", "w") as af:
55     af.write(f"{correct}\n")
56
57 # 7) Display question prompt on-screen (two lines)
58 q1 = Text(
59     "What is the total area under one period of this moving wave?",
60     font_size=32
61 ).to_edge(UP)
62 q2 = Text(
63     "Answer as a single integer.",
64     font_size=24
65 ).next_to(q1, DOWN, buff=0.2)
66 self.play(Write(q1), Write(q2))
67 self.wait(2)
68
69 # 8) Clean up
70 graph.remove_updater(update_graph)
71 phase.clear_updaters()
72 self.wait(1)

```

## Mathematical Reasoning - Region Area Comparison



**Query:** Answer the question in this video.

### Question Text:

Which region is larger, R1 or R2?

**Ground truth:** R2

### Code

The python code summarization is pasted below, the full code is available on [github](#).

```

1 from manim import *
2 import random
3 import numpy as np
4
5 class RegionCompareQuestion(Scene):
6     def construct(self):
7         # 1) Define two wave functions f and g
8         A1, B1, C1 = random.randint(1,3), random.randint(1,3), random.uniform(0, 2*np.pi)
9         A2, B2, C2 = random.randint(1,3), random.randint(1,3), random.uniform(0, 2*np.pi)
10        f = lambda x: A1 * np.sin(B1*x + C1)
11        g = lambda x: A2 * np.cos(B2*x + C2)
12
13        # 2) Setup domain and axes
14        x_min, x_max = -PI, PI
15        axes = Axes(
16            x_range=[x_min, x_max, PI/2],
17            y_range=[-(max(A1,A2)+1), max(A1,A2)+1, 1],
18            x_length=8, y_length=5,
19            tips=True
20        )
21        labels = axes.get_axis_labels("x", "y")
22        self.play(Create(axes), Write(labels), run_time=1)
23
24        # 3) Plot f (blue) and g (red)
25        graph_f = axes.plot(lambda t: f(t), x_range=[x_min, x_max], color=BLUE)
26        graph_g = axes.plot(lambda t: g(t), x_range=[x_min, x_max], color=RED)
27        self.play(Create(graph_f), Create(graph_g), run_time=2)
28
29        # 4) Shade R1 over [-pi, 0]
30        xs1 = np.linspace(x_min, 0, 200)
31        polys1 = VGroup()
32        for i in range(len(xs1)-1):
33            x0, x1 = xs1[i], xs1[i+1]
34            y0f, y1f = f(x0), f(x1)
35            y0g, y1g = g(x0), g(x1)
36            mid = (f((x0+x1)/2) - g((x0+x1)/2))
37            if mid > 0:
38                lower0, upper0 = y0g, y0f
39                lower1, upper1 = y1g, y1f
40                color = BLUE
41            else:
42                lower0, upper0 = y0f, y0g
43                lower1, upper1 = y1f, y1g

```

```

44         color = RED
45         poly = Polygon(
46             axes.c2p(x0, lower0),
47             axes.c2p(x0, upper0),
48             axes.c2p(x1, upper1),
49             axes.c2p(x1, lower1),
50             fill_color=color, fill_opacity=0.5, stroke_opacity=0
51         )
52         polys1.add(poly)
53     self.play(Create(polys1), run_time=2)
54     r1_label = Text("R1", font_size=24, color=WHITE).next_to(
55         axes.c2p((x_min+0)/2, max(A1,A2)/2), UP, buff=0.2
56     )
57     self.play(Write(r1_label))
58
59 # 5) Shade R2 over [0, pi] (fresh)
60 xs2 = np.linspace(0, x_max, 200)
61 polys2 = VGroup()
62 for i in range(len(xs2)-1):
63     x0, x1 = xs2[i], xs2[i+1]
64     y0f, y1f = f(x0), f(x1)
65     y0g, y1g = g(x0), g(x1)
66     mid = (f((x0+x1)/2) - g((x0+x1)/2))
67     if mid > 0:
68         lower0, upper0 = y0g, y0f
69         lower1, upper1 = y1g, y1f
70         color = BLUE
71     else:
72         lower0, upper0 = y0f, y0g
73         lower1, upper1 = y1f, y1g
74         color = RED
75     poly = Polygon(
76         axes.c2p(x0, lower0),
77         axes.c2p(x0, upper0),
78         axes.c2p(x1, upper1),
79         axes.c2p(x1, lower1),
80         fill_color=color, fill_opacity=0.5, stroke_opacity=0
81     )
82     polys2.add(poly)
83 # Fade out R1 and its label, then draw R2
84 self.play(FadeOut(polys1), FadeOut(r1_label), Create(polys2), run_time=2)
85 r2_label = Text("R2", font_size=24, color=WHITE).next_to(
86     axes.c2p((0+x_max)/2, max(A1,A2)/2), UP, buff=0.2
87 )
88 self.play(Write(r2_label))
89
90 # 6) Compute areas
91 area1 = np.trapz(np.abs(f(xs1) - g(xs1)), xs1)
92 area2 = np.trapz(np.abs(f(xs2) - g(xs2)), xs2)
93 correct = "R1" if area1 > area2 else "R2"
94
95 # 7) Show question
96 question = Text("Which region is larger, R1 or R2?", font_size=36).to_edge(UP)
97 self.play(Write(question))
98
99 # 8) Show choices
100 choices = VGroup(
101     Text("A. R1", font_size=24),
102     Text("B. R2", font_size=24)
103 ).arrange(DOWN, aligned_edge=LEFT, buff=0.5) \
104     .next_to(axes, RIGHT, buff=1)
105 self.play(Write(choices))
106
107 # 9) Save answer
108 with open("answer.txt","w") as out:
109     out.write(f"Region: {correct}\n")
110
111 self.wait(2)

```

### B.3. Physical Reasoning

#### Physical Reasoning - Teapot Rotating



**Query:** Answer the question in this video.

**Question Text:**

From the scenarios previewed, which video aligns with observable physical interactions?

A teapot is placed on a rotating display that rotates clockwise. Static shot with no camera movement.

Choose a letter (e.g., S). If none of them are realistic, return None.

**Ground truth:** E

#### Code

The python code summarization is pasted below, the full code is available on [github](#).

```

1 import csv
2 import random
3 import argparse
4 from pathlib import Path
5 from typing import List, Optional, Tuple
6 import cv2
7 import numpy as np
8 from moviepy.editor import VideoFileClip, CompositeVideoClip, ColorClip
9 import textwrap
10
11
12 class VideoGridGenerator:
13     """A class to generate video grids with physics questions"""
14
15     def __init__(self, target_width: int = 1920, target_fps: int = 30):
16         self.target_width = target_width
17         self.target_fps = target_fps
18         self.question_templates = [
19             "Which of the earlier videos reflects realistic physics in this
situation?\n\n{prompt}\n\nAnswer with a single letter (e.g., H). If none of them are realistic,
return None.",
20             "From the clips you viewed, which one accurately simulates natural motion/behavior
here?\n\n{prompt}\n\nRespond using one letter (e.g., M). If none of them are realistic, return
None.",
21             "Which previously shown video demonstrates plausible physics for this
event?\n\n{prompt}\n\nSubmit one letter (e.g., K). If none of them are realistic, return None."]

```

```

22     "Which of the earlier examples exhibits scientifically valid
23 motion/behavior?\n\n{prompt}\n\nReply with a single letter (e.g., P). If none of them are
24 realistic, return None.", 
25     "Which previously shown video adheres to the laws of physics in this
26 experiment?\n\n{prompt}\n\nAnswer with a single letter (e.g., J). If none of them are realistic,
27 return None.", 
28     "{prompt}\n\nWhich of the earlier videos aligns with real-world physics in this
29 scenario?\n\nRespond with a single letter (e.g., K). If none of them are realistic, return None.", 
30     "{prompt}\n\nFrom the clips viewed earlier, which one adheres to the laws of physics in
31 this situation?\n\nChoose the correct letter (e.g., L). If none of them are realistic, return
32 None.", 
33     "{prompt}\n\nWhich demonstrated video exhibits physical plausibility for the scenario
34 above?\n\nIndicate your answer as a single letter (e.g., M).", 
35     "{prompt}\n\nConsidering the scene described, which of the clips shown prior follows
36 realistic physical principles?\n\nProvide your answer in a single letter (e.g., Q). If none of
37 them are realistic, return None.", 
38     "{prompt}\n\nAmong the options previewed earlier, which video is consistent with
39 real-world physics in this context?\n\nSubmit your answer as a single letter (e.g., R). If none of
40 them are realistic, return None.", 
41     ]
42
43 def get_video_properties(self, video_path: str) -> Tuple[float, float, int, int]:
44     """Get video properties using MoviePy"""
45     try:
46         clip = VideoFileClip(video_path)
47         fps = clip.fps
48         duration = clip.duration
49         width, height = clip.size
50         clip.close()
51         return fps, duration, width, height
52     except Exception as e:
53         print(f"Error reading video {video_path}: {e}")
54         raise
55
56 def add_letter_to_video(self, video_clip: VideoFileClip, letter: str) -> VideoFileClip:
57     """Add a letter marker in a black strip at the bottom right of the video"""
58     def add_letter(frame):
59         frame_copy = frame.copy()
60
61         # Create a black strip at the bottom of the frame
62         strip_height = 60
63         strip_width = 250
64
65         # Calculate position for bottom right placement
66         x_start = frame_copy.shape[1] - strip_width
67         y_start = frame_copy.shape[0] - strip_height
68
69         # Create black rectangle
70         cv2.rectangle(frame_copy, (x_start, y_start),
71                       (frame_copy.shape[1], frame_copy.shape[0]),
72                       (0, 0, 0), -1)
73
74         # Calculate text position to center in the black strip
75         text_size = cv2.getTextSize(letter, cv2.FONT_HERSHEY_DUPLEX, 2, 2)[0]
76         text_x = x_start + (strip_width - text_size[0]) // 2
77         text_y = y_start + (strip_height + text_size[1]) // 2
78
79         # Add shadow for better visibility
80         cv2.putText(frame_copy, letter, (text_x+2, text_y+2),
81                     cv2.FONT_HERSHEY_DUPLEX, 2, (50, 50, 50), 3, cv2.LINE_AA)
82
83         # Add the letter in white
84         cv2.putText(frame_copy, letter, (text_x, text_y),
85                     cv2.FONT_HERSHEY_DUPLEX, 2, (255, 255, 255), 2, cv2.LINE_AA)
86
87         return frame_copy
88
89     return video_clip.fl_image(add_letter)
90
91 def wrap_text(self, text: str, max_width: int = 40) -> str:
92     """Wrap text to fit within a maximum width"""
93     lines = []
94     for paragraph in text.split('\n'):
95         wrapped_lines = textwrap.wrap(paragraph, width=max_width)
96         lines.extend(wrapped_lines)
97     # Add an empty line between paragraphs if there's another paragraph coming
98     if paragraph != text.split('\n')[-1]:
99         lines.append('')
100    return '\n'.join(lines)

```

```

89
90     def add_ending_question(self, final_clip: VideoFileClip, question_text: str,
91                             fade_duration: float = 3) -> CompositeVideoClip:
92         """Add a fading question frame at the end of the video"""
93         wrapped_question = self.wrap_text(question_text)
94
95     def make_question_frame(t):
96         # Create a black frame
97         frame = np.zeros((final_clip.h, final_clip.w, 3), dtype=np.uint8)
98
99         # Split the text into lines
100        lines = wrapped_question.split('\n')
101
102        # Calculate positions for centered text
103        y_position = final_clip.h // 2 - (len(lines) * 50) // 2
104
105        # Add each line of text
106        for line in lines:
107            # Get text size
108            text_size = cv2.getTextSize(
109                line, cv2.FONT_HERSHEY_DUPLEX, 1.5, 2)[0]
110
111            # Calculate x position to center text
112            x_position = (final_clip.w - text_size[0]) // 2
113
114            # Add shadow
115            cv2.putText(
116                frame, line, (x_position+2, y_position+2),
117                cv2.FONT_HERSHEY_DUPLEX, 1.5, (50, 50, 50), 4, cv2.LINE_AA
118            )
119
120            # Add main text
121            cv2.putText(
122                frame, line, (x_position, y_position),
123                cv2.FONT_HERSHEY_DUPLEX, 1.5, (255, 255, 255), 2, cv2.LINE_AA
124            )
125
126            # Move to next line
127            y_position += 60
128
129            # Add fade-in effect
130            if t < fade_duration/2:
131                alpha = t / (fade_duration/2)
132                return (frame * alpha).astype(np.uint8)
133
134        return frame
135
136    # Create a ColorClip with the question text
137    question_clip = ColorClip(
138        size=(final_clip.w, final_clip.h),
139        color=[0, 0, 0],
140        duration=fade_duration
141    )
142
143    # Override the make_frame method to use our custom text function
144    question_clip.make_frame = make_question_frame
145    question_clip = question_clip.set_fps(final_clip.fps)
146    question_clip = question_clip.set_start(final_clip.duration)
147
148    # Combine with the main video
149    final_video = CompositeVideoClip(
150        [final_clip, question_clip],
151        size=final_clip.size
152    )
153
154    final_video = final_video.set_duration(final_clip.duration + fade_duration)
155    return final_video
156
157    def calculate_grid_layout(self, num_videos: int) -> Tuple[int, int]:
158        """Calculate optimal grid layout for given number of videos"""
159        if num_videos <= 2:
160            return num_videos, 1
161        elif num_videos <= 4:
162            return 2, 2
163        elif num_videos <= 6:
164            return 2, 3
165        elif num_videos <= 9:
166            return 3, 3
167        else:

```

```

168     # For more than 9 videos, use a more general approach
169     cols = int(np.ceil(np.sqrt(num_videos)))
170     rows = int(np.ceil(num_videos / cols))
171     return cols, rows
172
173 def create_video_grid(self, video_paths: List[str], output_path: str,
174                       ending_question: str, solutions_dir: Optional[Path] = None,
175                       questions_text_dir: Optional[Path] = None,
176                       num_videos: Optional[int] = None) -> None:
177     """Create a video grid with specified number of videos"""
178
179     # Limit number of videos if specified
180     if num_videos is not None and len(video_paths) > num_videos:
181         video_paths = video_paths[:num_videos]
182
183     if not video_paths:
184         raise ValueError("No video paths provided")
185
186     print(f"Creating grid with {len(video_paths)} videos")
187
188     # Load video clips
189     original_clips = []
190     for path in video_paths:
191         try:
192             clip = VideoFileClip(path)
193             original_clips.append(clip)
194         except Exception as e:
195             print(f"Error loading video {path}: {e}")
196             continue
197
198     if not original_clips:
199         raise ValueError("No valid video clips could be loaded")
200
201     # Find the minimum duration among all clips
202     min_duration = min(clip.duration for clip in original_clips)
203
204     # Trim all clips to the same duration
205     clips = [clip.subclip(0, min_duration) for clip in original_clips]
206
207     # Calculate the grid layout
208     num_videos_actual = len(clips)
209     num_columns, num_rows = self.calculate_grid_layout(num_videos_actual)
210
211     # Calculate the width for each video clip
212     clip_width = self.target_width // num_columns
213
214     # Resize all clips while preserving aspect ratio
215     resized_clips = []
216     for clip in clips:
217         resized_clip = clip.resize(width=clip_width)
218         resized_clips.append(resized_clip)
219
220     # Find the correct answer (video with 'full' in the name)
221     correct_answer = self._find_correct_answer(video_paths)
222
223     # Save the correct answer and question text
224     self._save_answer_and_question(output_path, correct_answer, ending_question,
225                                   solutions_dir, questions_text_dir)
226
227     # Add letter markers to each clip
228     marked_clips = [
229         self.add_letter_to_video(clip, chr(65 + i))
230         for i, clip in enumerate(resized_clips)
231     ]
232
233     # Calculate grid dimensions
234     row_heights = self._calculate_row_heights(resized_clips, num_columns, num_rows)
235     total_height = sum(row_heights)
236     final_size = (self.target_width, total_height)
237
238     # Create composite clips arrangement
239     composite_clips = self._arrange_clips_in_grid(marked_clips, num_columns, num_rows,
240                                                   clip_width, row_heights)
241
242     # Create the final composite video
243     background = ColorClip(size=final_size, color=(0, 0, 0), duration=min_duration)
244     background = background.set_fps(self.target_fps)
245
246     final_grid = CompositeVideoClip([background] + composite_clips, size=final_size)

```

```

247     final_grid = final_grid.set_fps(self.target_fps)
248
249     # Add the ending question with fade
250     final_clip = self.add_ending_question(final_grid, ending_question)
251
252     # Write the result to a file
253     try:
254         final_clip.write_videofile(output_path, codec='libx264')
255         print(f"Successfully created video: {output_path}")
256     except Exception as e:
257         print(f"Error writing video file: {e}")
258         raise
259     finally:
260         # Clean up resources
261         self._cleanup_clips(original_clips + resized_clips + marked_clips +
262                             [final_grid, final_clip])
263
264     def _find_correct_answer(self, video_paths: List[str]) -> Optional[str]:
265         """Find the correct answer based on video filename containing 'full'"""
266         for i, path in enumerate(video_paths):
267             if 'full' in Path(path).name.lower():
268                 return chr(65 + i) # Convert to letter (A, B, C, etc.)
269         return None
270
271     def _save_answer_and_question(self, output_path: str, correct_answer: Optional[str],
272                                  ending_question: str, solutions_dir: Optional[Path],
273                                  questions_text_dir: Optional[Path]) -> None:
274         """Save the correct answer and question text to files"""
275         if correct_answer and solutions_dir:
276             output_filename = Path(output_path).stem
277             answer_file = solutions_dir / f"{output_filename}.txt"
278             try:
279                 with open(answer_file, 'w') as f:
280                     f.write(correct_answer)
281             except Exception as e:
282                 print(f"Error saving answer file: {e}")
283
284         if questions_text_dir:
285             output_filename = Path(output_path).stem
286             question_file = questions_text_dir / f"{output_filename}.txt"
287             try:
288                 with open(question_file, 'w') as f:
289                     f.write(ending_question)
290             except Exception as e:
291                 print(f"Error saving question file: {e}")
292
293     def _calculate_row_heights(self, resized_clips: List[VideoFileClip],
294                               num_columns: int, num_rows: int) -> List[int]:
295         """Calculate the height of each row in the grid"""
296         row_heights = []
297         num_videos = len(resized_clips)
298
299         for r in range(num_rows):
300             row_start_idx = r * num_columns
301             row_end_idx = min(row_start_idx + num_columns, num_videos)
302
303             if row_end_idx > row_start_idx:
304                 row_clips = resized_clips[row_start_idx:row_end_idx]
305                 row_height = max(clip.h for clip in row_clips)
306                 row_heights.append(row_height)
307             else:
308                 row_heights.append(0)
309
310         return row_heights
311
312     def _arrange_clips_in_grid(self, marked_clips: List[VideoFileClip],
313                               num_columns: int, num_rows: int, clip_width: int,
314                               row_heights: List[int]) -> List[VideoFileClip]:
315         """Arrange clips in a grid layout"""
316         composite_clips = []
317         current_y = 0
318         num_videos = len(marked_clips)
319
320         for row in range(num_rows):
321             row_start_idx = row * num_columns
322             row_end_idx = min(row_start_idx + num_columns, num_videos)
323             row_height = row_heights[row]
324
325             if row_end_idx <= row_start_idx:

```

```

326         break
327
328     # Handle the last row with potentially fewer videos
329     videos_in_row = row_end_idx - row_start_idx
330     if videos_in_row == 1 and num_columns > 1:
331         # Center single video in last row
332         center_x = (self.target_width - clip_width) // 2
333         clip = marked_clips[row_start_idx]
334         composite_clips.append(clip.set_position((center_x, current_y)))
335     else:
336         # Regular row arrangement
337         for i in range(row_start_idx, row_end_idx):
338             col = i - row_start_idx
339             clip = marked_clips[i]
340             composite_clips.append(clip.set_position((col * clip_width, current_y)))
341
342     current_y += row_height
343
344     return composite_clips
345
346 def _cleanup_clips(self, clips: List) -> None:
347     """Clean up video clips to free memory"""
348     for clip in clips:
349         try:
350             if hasattr(clip, 'close'):
351                 clip.close()
352         except:
353             pass
354
355     def generate_question(self, prompt: str) -> str:
356         """Generate a random question using the prompt"""
357         template = random.choice(self.question_templates)
358         return template.format(prompt=prompt)
359
360
361 def load_descriptions(csv_path: str) -> List[List[str]]:
362     """Load descriptions from CSV file"""
363     try:
364         with open(csv_path, 'r', encoding='utf-8') as f:
365             reader = csv.reader(f)
366             return list(reader)
367     except Exception as e:
368         print(f"Error loading descriptions from {csv_path}: {e}")
369     return []
370
371
372 def create_output_directories(base_dirs: List[str]) -> List[Path]:
373     """Create output directories if they don't exist"""
374     paths = []
375     for dir_name in base_dirs:
376         path = Path(dir_name)
377         path.mkdir(exist_ok=True)
378         paths.append(path)
379     return paths
380
381
382 def main():
383     parser = argparse.ArgumentParser(description='Generate physics question videos from input
384     videos')
385     parser.add_argument('--descriptions', default='descriptions/descriptions.csv',
386                         help='Path to descriptions CSV file')
387     parser.add_argument('--video-root', default='videos_generated',
388                         help='Root directory containing generated videos')
389     parser.add_argument('--frames-root', default='frames_selected',
390                         help='Root directory containing selected frames')
391     parser.add_argument('--num-videos', type=int,
392                         help='Number of videos to include in each grid (optional)')
393     parser.add_argument('--target-width', type=int, default=1920,
394                         help='Target width for output videos')
395     parser.add_argument('--target-fps', type=int, default=30,
396                         help='Target FPS for output videos')
397     parser.add_argument('--shuffle', action='store_true',
398                         help='Shuffle video order randomly')
399
400     args = parser.parse_args()
401
402     # Load prompt data
403     prompt_data = load_descriptions(args.descriptions)
404     if not prompt_data:
405

```

```

404     print("No prompt data loaded. Exiting.")
405     return
406
407     # Create output directories
408     questions_dir, questions_text_dir, solutions_dir = create_output_directories([
409         'questions', 'questions_text', 'solutions'
410     ])
411
412     # Initialize video grid generator
413     generator = VideoGridGenerator(target_width=args.target_width,
414                                     target_fps=args.target_fps)
415
416     # Process images and generate videos
417     image_root = Path(args.frames_root)
418     images = sorted(image_root.glob("*.jpg"))
419     video_root = Path(args.video_root)
420
421     for image in images:
422         try:
423             prefix = image.stem.split('_')[0]
424             video_paths = video_root.glob(f"{prefix}*.{mp4}")
425             video_paths = [str(v) for v in sorted(video_paths)]
426
427             if not video_paths:
428                 print(f"No videos found for prefix {prefix}")
429                 continue
430
431             # Shuffle videos if requested
432             if args.shuffle:
433                 random.shuffle(video_paths)
434
435             # Get prompt for this index
436             idx = int(prefix)
437             if idx >= len(prompt_data):
438                 print(f"No prompt data for index {idx}")
439                 continue
440
441             prompt = prompt_data[idx][1]
442             question = generator.generate_question(prompt)
443
444             # Generate output path
445             video_output_path = questions_dir / f"physics_iq_{prefix}.mp4"
446
447             # Create the video grid
448             generator.create_video_grid(
449                 video_paths=video_paths,
450                 output_path=str(video_output_path),
451                 ending_question=question,
452                 solutions_dir=solutions_dir,
453                 questions_text_dir=questions_text_dir,
454                 num_videos=args.num_videos
455             )
456
457         except Exception as e:
458             print(f"Error processing {image}: {e}")
459             continue
460
461     if __name__ == "__main__":
462         main()
463
464     # # bash# Use only 3 videos per grid
465     # python script.py --num-videos 3
466
467     # Shuffle videos and use 6 videos per grid
468     # python script.py --num-videos 6 --shuffle
469
470     # # Custom resolution and fps
471     # python script.py --target-width 1280 --target-fps 24
472
473     # # Use all available videos (original behavior)
474     # python script.py

```

## B.4. Planning Reasoning

**Planning Reasoning - Frozen Lake Path Selection**

**Question:** Answer the question in this video.

**Question Text:**  
Which of the listed move sequences carries the agent from Start to Goal without stepping onto a hole?

(A) ↓↓↓↓ ⇒ ⇒ ↓ ⇒ ⇒ ↓↓ ⇒ ↓↓  
 (B) ⇒ ⇒ ↓↓ ⇒ ⇒ ⇒ ↓↓↓ ⇒ ↓ ⇒ ⇒  
 (C) ↓↓↓ ⇒ ↓↓ ⇒ ⇒ ⇒ ↓ ⇒ ↓↓ ⇒  
 (D) ⇒ ⇒ ⇒ ↓ ⇒ ⇒ ↓ ⇒ ↓↓↓ ⇒ ↓↓  
 (E) ↓↓ ⇒ ↓↓↓↓ ⇒ ↓↓↓ ⇒ ⇒

Please return a single letter (e.g. A). Nothing preceding or following it.

**Ground truth: D**

### Code

The python code summarization is pasted below, the full code is available on [github](#).

```

1 # =====
2 # FOGGY FROZEN LAKE MAZE GENERATION
3 # =====
4
5 def generate_maze_question(difficulty_params):
6     """
7         Generate a spatial reasoning question with controllable difficulty
8
9     Difficulty Parameters:
10     - maze_size: Grid dimensions (4x4 to 20x20)
11     - visibility_range: How far agent can see (1-5 cells)
12     - num_holes: Number of obstacles/holes (0 to size/4)
13     - num_options: Multiple choice options (3-8)
14     - path_length: Minimum path length required
15     """
16
17     # STEP 1: Generate Base Environment

```

```

18     maze_size = difficulty_params['maze_size']           # 4-20 (larger = harder)
19     num_holes = difficulty_params['num_holes']          # 0-size/4 (more = harder)
20
21     grid = create_random_map(size=maze_size)
22     for _ in range(num_holes):
23         place_hole_randomly(grid)
24
25     env = FrozenLakeEnvironment(grid)
26     start = (0, 0)                                     # Top-left corner
27     goal = (maze_size-1, maze_size-1) # Bottom-right corner
28
29
30     # STEP 2: Apply Fog-of-War (Limited Visibility)
31     visibility_range = difficulty_params['visibility_range'] # 1-5 (less = harder)
32
33     def apply_fog(frame, agent_position, visibility_range):
34         """Hide cells beyond visibility distance from agent"""
35         foggy_frame = frame.copy()
36
37         for each_cell in grid:
38             distance = manhattan_distance(each_cell, agent_position)
39             if distance > visibility_range:
40                 apply_gray_fog(foggy_frame, each_cell)
41
42     return foggy_frame
43
44
45     # STEP 3: Generate Exploration Animation
46     def explore_maze():
47         """Agent explores maze with limited visibility"""
48         frames = []
49         agent_pos = start
50         visited_cells = set()
51
52         while not fully_explored():
53             # Find unvisited neighbors within visibility
54             unvisited = find_unvisited_neighbors(agent_pos, visibility_range)
55
56             if unvisited:
57                 # Move to random unvisited neighbor
58                 next_pos = random.choice(unvisited)
59                 agent_pos = move_to(next_pos)
60             else:
61                 # Use BFS to find path to nearest unvisited area
62                 path = find_path_to_unvisited_area(agent_pos, visited_cells)
63                 agent_pos = follow_path(path)
64
65             visited_cells.add(agent_pos)
66
67             # Render frame with fog applied
68             frame = render_environment(env, agent_pos)
69             foggy_frame = apply_fog(frame, agent_pos, visibility_range)
70             frames.append(foggy_frame)
71
72     return frames
73
74
75     # STEP 4: Find Valid Solutions
76     def find_shortest_paths():
77         """Use BFS to find all shortest paths from start to goal"""
78         queue = [(start, [])] # (position, path_so_far)
79         visited = {start: 0}
80         shortest_paths = []
81         min_length = infinity
82
83         while queue:
84             pos, path = queue.pop_front()
85
86             if pos == goal:
87                 if len(path) < min_length:
88                     min_length = len(path)
89                     shortest_paths = [path]
90                 elif len(path) == min_length:
91                     shortest_paths.append(path)
92                 continue
93
94             for action in [LEFT, DOWN, RIGHT, UP]:
95                 next_pos = apply_action(pos, action)
96

```

```

97         if is_valid_position(next_pos) and not_hole(next_pos):
98             new_path = path + [action]
99             if next_pos not in visited or visited[next_pos] >= len(new_path):
100                 visited[next_pos] = len(new_path)
101                 queue.append((next_pos, new_path))
102
103     return shortest_paths
104
105
106 # STEP 5: Generate Distractor Paths
107 def generate_wrong_paths(correct_length, num_distractors):
108     """Create plausible but incorrect paths of similar length"""
109     wrong_paths = []
110
111     while len(wrong_paths) < num_distractors:
112         # Generate random path of same length
113         path = [random_action() for _ in range(correct_length)]
114
115         # Check if path fails to reach goal
116         if not reaches_goal(path) and is_plausible(path):
117             wrong_paths.append(path)
118
119     return wrong_paths
120
121
122 # STEP 6: Create Multiple Choice Question
123 num_options = difficulty_params['num_options']      # 3-8 (more = harder)
124
125 correct_paths = find_shortest_paths()
126 correct_path = random.choice(correct_paths)
127 wrong_paths = generate_wrong_paths(len(correct_path), num_options - 1)
128
129 # Shuffle options
130 all_options = wrong_paths + [correct_path]
131 random.shuffle(all_options)
132 correct_answer = find_index(correct_path, all_options)
133
134 # Convert to human-readable format
135 action_symbols = {LEFT: " ", DOWN: " ", RIGHT: " ", UP: " "}
136
137 question_text = "Which path leads to the goal without falling into holes?"
138 options_text = ""
139 for i, path in enumerate(all_options):
140     letter = chr(65 + i) # A, B, C, ...
141     path_symbols = " ".join([action_symbols[action] for action in path])
142     options_text += f"({letter}) {path_symbols}\n"
143
144
145 # STEP 7: Create Professional Animation
146 def create_animation(frames, question_text, options_text):
147     """Generate video with intro, maze exploration, and question display"""
148
149     total_frames = (
150         intro_frames +                      # Title and setup
151         len(frames) +                     # Maze exploration
152         outro_frames                      # Question and options
153     )
154
155     for frame_num in range(total_frames):
156         if frame_num < intro_frames:
157             render_intro_with_title()
158         elif frame_num < intro_frames + len(frames):
159             maze_frame_idx = frame_num - intro_frames
160             render_maze_frame(frames[maze_frame_idx])
161         else:
162             render_question_and_options(question_text, options_text)
163
164
165 # STEP 8: Save Output Files
166 question_name = f"maze_sz{maze_size}_vis{visibility_range}_holes{num_holes}"
167
168 # Save video as MP4
169 video_path = f"questions/{question_name}.mp4"
170 create_professional_animation(
171     frames=explore_maze(),
172     output_path=video_path,
173     question_text=question_text,
174     options_text=options_text
175 )

```

```

176     # Save ground truth answer as TXT
177     solution_path = f"solutions/{question_name}.txt"
178     with open(solution_path, "w") as f:
179         f.write(chr(65 + correct_answer))  # Single letter: A, B, C, etc.
180
182     # Save question text as TXT
183     full_question_text = f"{question_text}\n{n=options_text}\nPlease return a single letter (e.g. A)"
184     question_text_path = f"questions_text/{question_name}.txt"
185     with open(question_text_path, "w") as f:
186         f.write(full_question_text)
187
188     # RETURN COMPLETE QUESTION PACKAGE
189     return {
190         'video_path': video_path,
191         'solution_path': solution_path,
192         'question_text_path': question_text_path,
193         'question_text': question_text,
194         'options': options_text,
195         'correct_answer': chr(65 + correct_answer),
196         'metadata': {
197             'maze_size': maze_size,
198             'visibility_range': visibility_range,
199             'num_holes': num_holes,
200             'path_length': len(correct_path),
201             'difficulty_score': calculate_difficulty_score(difficulty_params)
202         }
203     }
204
205
206 # =====
207 # DIFFICULTY CONTROL MECHANISMS
208 # =====
209
210 def calculate_difficulty_score(params):
211     """
212     Combine multiple factors into overall difficulty score (0-100)
213     """
214     base_score = 0
215
216     # Maze size contribution (0-30 points)
217     size_factor = min(30, (params['maze_size'] - 4) * 2)
218
219     # Visibility contribution (0-25 points, inverse relationship)
220     visibility_factor = max(0, 25 - params['visibility_range'] * 5)
221
222     # Obstacles contribution (0-20 points)
223     obstacle_density = params['num_holes'] / (params['maze_size'] ** 2)
224     obstacle_factor = min(20, obstacle_density * 100)
225
226     # Path complexity contribution (0-15 points)
227     path_factor = min(15, params.get('path_length', 0) - params['maze_size'])
228
229     # Options contribution (0-10 points)
230     options_factor = max(0, (params['num_options'] - 3) * 2)
231
232     total_score = (size_factor + visibility_factor + obstacle_factor +
233                     path_factor + options_factor)
234
235     return min(100, total_score)
236
237
238 # =====
239 # EXAMPLE DIFFICULTY CONFIGURATIONS
240 # =====
241
242 DIFFICULTY_PRESETS = {
243     'easy': {
244         'maze_size': 6,
245         'visibility_range': 3,
246         'num_holes': 2,
247         'num_options': 4,
248         'expected_difficulty': 25
249     },
250
251     'medium': {
252         'maze_size': 10,
253         'visibility_range': 2,
254         'num_holes': 8,

```

```
255     'num_options': 5,
256     'expected_difficulty': 55
257 },
258
259 'hard': {
260     'maze_size': 15,
261     'visibility_range': 1,
262     'num_holes': 20,
263     'num_options': 7,
264     'expected_difficulty': 85
265 }
266 }
267
268 # Usage Example:
269 if __name__ == "__main__":
270     # Create necessary directories
271     os.makedirs("questions", exist_ok=True)
272     os.makedirs("solutions", exist_ok=True)
273     os.makedirs("questions_text", exist_ok=True)
274
275     # Generate question with medium difficulty
276     question_package = generate_maze_question(DIFFICULTY_PRESETS['medium'])
277
278     print(f"Generated files:")
279     print(f"    Video: {question_package['video_path']}")
280     print(f"    Answer: {question_package['solution_path']}")
281     print(f"    Question: {question_package['question_text_path']}")
282     print(f"    Difficulty: {question_package['metadata']['difficulty_score']}/100")
283
284     # Files saved:
285     # - questions/maze_sz10_vis2_holes8.mp4      (video with question)
286     # - solutions/maze_sz10_vis2_holes8.txt      (single letter answer)
287     # - questions_text/maze_sz10_vis2_holes8.txt (full question text)
```

## B.5. Spatial Reasoning

### Spatial Reasoning - Matching the Missing Shape



**Query:** Answer the question in this video.

#### Question Text:

Observe the following structure. Which shape matches the missing cubes?

Answer with only one multiple choice option.

**Ground truth: D**

#### Code

The python code summarization is pasted below, the full code is available on [github](#).

```

1 from manim import *
2 import random
3 import json
4 import math
5 import os
6
7 class Cubes(ThreeDScene):
8     def __init__(self,
9                  p_type,
10                 grid_size,
11                 p_removed,
12                 cfg_path='../templates/prisms.json',
13                 max_iters=25,
14                 ):
15         super().__init__()
16         self.p_type = p_type
17         self.grid_size = grid_size
18         self.total = math.prod(grid_size)
19         self.n_removed = int(self.total * p_removed)
20         self.max_iters = max_iters
21         with open(cfg_path, "r") as f:
22             self.cfg = json.load(f)
23
24     def construct(self):
25         # Add background
26         bg = (
27             Rectangle(height=config.frame_height, width=config.frame_width)
28             .set_color()
29

```

```

30         color_gradient([random_bright_color(), random_bright_color()], 5)
31     )
32     .set_opacity(0.6)
33 )
34 self.add_fixed_in_frame_mobjects(bg)
35
36 VALID_COLORS = {"blue": BLUE, "red": RED, "green": GREEN, "yellow": YELLOW}
37
38 # Show opening prompt
39 prompt = Text(
40     "Observe the following structure", color=WHITE, font_size=36
41 ).move_to(ORIGIN)
42
43 self.play(FadeIn(prompt), run_time=0.5)
44 self.wait(1.5)
45 self.play(FadeOut(prompt), run_time=0.5)
46 self.wait(1)
47
48 # Create initial grid of cubes
49 rows, cols, depth = self.grid_size
50
51 cubes_vgroup = VGroup()
52 cube_list = [
53     [[None for z in range(depth)] for y in range(cols)] for z in range(rows)
54 ]
55 colors = [
56     [[None for z in range(depth)] for y in range(cols)] for z in range(rows)
57 ]
58 unique_colors = set()
59
60 for x in range(rows):
61     for y in range(cols):
62         for z in range(depth):
63             color = random.choice(list(VALID_COLORS.keys()))
64             cube = Cube(side_length=0.75)
65             cube.set_fill(color=VALID_COLORS[color], opacity=1)
66             cube.set_stroke(color=VALID_COLORS[color], width=2)
67             cube.x_idx = x
68             cube.y_idx = y
69             cube.z_idx = z
70             cube.shift(0.75 * (x * RIGHT + y * UP + z * OUT))
71
72             cube_list[x][y][z] = cube
73             colors[x][y][z] = color
74             cubes_vgroup.add(cube)
75             unique_colors.add(color)
76
77 max_height = config.frame_height * 0.6 # 5% buffer top and bottom
78
79 if cubes_vgroup.height > max_height:
80     cubes_vgroup.scale(max_height / cubes_vgroup.height)
81 cubes_vgroup.move_to(ORIGIN)
82
83 # Create and start rotating cube structure
84 cubes_vgroup.rotate(45 * DEGREES, axis=UP)
85 cubes_vgroup.rotate(20 * DEGREES, axis=RIGHT)
86 cubes_vgroup.add_updater(
87     lambda m, dt: m.rotate(1 * dt, axis=UP, about_point=ORIGIN)
88 )
89 self.play(Write(cubes_vgroup), runtime=1)
90
91 self.wait(1)
92
93 # Randomly remove cubes
94 heights = [[depth - 1 for j in range(cols)] for i in range(rows)]
95 iters = 0
96 cubes_to_remove = set()
97 idxs_to_remove = set()
98 done = False
99 while iters < self.max_iters and not done:
100     for row in range(rows):
101         for col in range(cols):
102             old_height = heights[row][col]
103             if random.uniform(0, 1) < 0.5:
104                 new_height = random.randint(0, heights[row][col])
105             else:
106                 new_height = old_height
107
108             heights[row][col] = new_height

```

```

109         for layer in range(old_height, new_height - 1, -1):
110             cubes_to_remove.add(cube_list[row][col][layer])
111             idxs_to_remove.add((row, col, layer))
112             if len(cubes_to_remove) == self.n_removed:
113                 done = True
114                 break
115             if done:
116                 break
117             if done:
118                 break
119             iters += 1
120
121     if not done:
122         raise ValueError(f"Failed to find valid configuration in {iters} attempts")
123
124     # Fade out cubes with updater
125     vt = ValueTracker(10)
126     for cube in cubes_to_remove:
127         cube.add_updater(lambda m: m.set_opacity(vt.get_value() / 10))
128     self.play(vt.animate.set_value(0), run_time=1.5, rate_func=smooth)
129     for cube in cubes_to_remove:
130         cube.clear_updaters()
131     cubes_vgroup.remove(*cubes_to_remove)
132
133     unique_colors = list(unique_colors)
134     color = random.choice(unique_colors)
135     N = random.randint(1, 4)
136
137     # Create random variants to actual missing shape
138     correct = idxs_to_remove
139     all_idxs = [
140         (i, j, k)
141         for i in range(rows)
142         for j in range(cols)
143         for k in range(depth)
144     ]
145     avail_to_add = set(all_idxs) - correct
146
147     variants = []
148     while len(variants) < 3:
149         # random numbers to add and remove
150         na = random.randint(
151             1, min(len(avail_to_add), max(1, len(correct))) // 4
152         )
153         nr = (
154             random.randint(1, min(len(correct), max(1, len(correct) - 1)) // 4)
155             if len(correct) > 1
156             else 1
157         )
158         to_add = set(random.sample(avail_to_add, na))
159         to_remove = set(random.sample(correct, nr))
160         variant = (correct - to_remove) | to_add
161         if variant != correct and variant not in variants:
162             variants.append(variant)
163
164     options = [correct] + variants
165     random.shuffle(options)
166     labels = ["A", "B", "C", "D"]
167
168     # Draw each option as a mini-diagram at the bottom
169     option_groups = VGroup()
170     positions = [LEFT * 3, LEFT, RIGHT, RIGHT * 3]
171     for pos, inds, lbl in zip(positions, options, labels):
172         mini = VGroup()
173         for i, j, k in inds:
174             c = Cube(side_length=0.2)
175             c.set_fill(colors[i][j][k], opacity=1)
176             c.set_stroke(colors[i][j][k], width=1)
177             c.shift(i * 0.2 * RIGHT + j * 0.2 * UP + k * 0.2 * OUT)
178             mini.add(c)
179         mini.scale(0.8)
180         mini.move_to(pos + DOWN * 3)
181         mini.rotate(-45, axis=UP)
182         mini.rotate(20, axis=RIGHT)
183         mini.add_updater(lambda m, dt: m.rotate(1.57 * dt, axis=UP))
184
185         label = Text(lbl).scale(0.2).next_to(mini, UP)
186         option_groups.add(VGroup(mini, label))
187

```

```

188     option_groups.scale_to_fit_height(config.frame_height * 0.4)
189     option_groups.scale_to_fit_width(config.frame_width * 0.9)
190
191     option_groups.to_edge(DOWN)
192     self.answer = labels[options.index(correct)]
193
194     self.wait(3)
195
196     # Show answer page title
197     self.play(*[FadeOut(mob) for mob in self.mobjects if mob != bg])
198     title_text = random.choice(self.cfg["text"][self.p_type])
199     title_text = title_text.replace("<C>", color)
200     title_text = title_text.replace("<N>", str(N))
201     lines = title_text.split("\n")
202
203     para = Paragraph(*lines, alignment="center", font_size=36, line_spacing=0.8)
204     para.move_to(ORIGIN)
205     if para.width > 0.9 * config.frame_width:
206         para.scale_to_fit_width(config.frame_width * 0.9)
207     self.add_fixed_in_frame_mobjects(para)
208     self.play(Write(para), run_time=1.5)
209
210     self.play(para.animate.to_edge(UP, buff=0.2 * config.frame_height))
211
212     self.wait(0.5)
213     self.play(Write(option_groups, run_time=1))
214
215     self.wait(3)
216     self.question_text = "Observe the following structure. {}".format(
217         title_text.replace("\n", " "))
218
219
220
221 def create_problem(grid_size, p_type, p_removed, path, file_name):
222     config.output_file = f"{path}/questions/{file_name}"
223     scene = Cubes(p_type, grid_size, p_removed)
224     scene.render()
225     with open(f"media/videos/1080p60/{path}/solutions/{file_name}.txt", "w") as f:
226         f.write(str(scene.answer))
227     with open(f"media/videos/1080p60/{path}/question_text/{file_name}.txt", "w") as f:
228         f.write(scene.question_text)
229
230
231 if __name__ == "__main__":
232     os.makedirs("media/videos/1080p60/cubes/questions", exist_ok=True)
233     os.makedirs("media/videos/1080p60/cubes/solutions", exist_ok=True)
234     os.makedirs("media/videos/1080p60/cubes/question_text", exist_ok=True)
235
236     create_problem((5, 5, 3), "missing_shape", 0.45, "cubes", "missing_shape") # Change grid params
as needed

```

## B.6. Temporal Reasoning

### Temporal Reasoning - Dominoes



**Query:** Answer the question in this video.

#### Question Text:

How many dominoes fell before the fourth green domino?

**Ground truth: 16**

### Code

The python code summarization is pasted below, the full code is available on github.

```

1 from manim import *
2
3 # List your question texts here (one per video)
4 domino_question_texts = [
5     "What event caused the dominoes to change direction?",
6     "Did the direction change occur before or after the 30th domino fell?",
7     "Insert your third question here",
8     "Insert your fourth question here",
9 ]
10
11 from manim import *
12 import random
13
14 class domino1(Scene):
15     def construct(self):
16         # 1) Question text
17         question_text = "How many dominoes were red?"
18         question = Text(question_text, font_size=24).to_edge(UP)
19
20
21         # 2) Parameters
22         num_dominoes = 30
23         turn_index = 20
24         domino_width = 0.2
25         domino_height = 0.6
26         spacing = 0.3
27
28         # 3) Precompute which 7 positions will be red
29         red_positions = set(random.sample(range(num_dominoes), 7))
30
31         # 4) Build and display dominoes one by one
32         dominoes = []
33         for i in range(num_dominoes):
34             # determine position on screen
35             x = -6 + i * spacing
36             y = 0

```

```

37     # choose color
38     color = RED if i in red_positions else random.choice([
39         BLUE, GREEN, YELLOW, PURPLE, ORANGE, TEAL, PINK
40     ])
41
42     # create domino
43     domino = Rectangle(
44         width=domino_width,
45         height=domino_height,
46         fill_opacity=1,
47         color=color
48     ).move_to([x, y, 0])
49     dominoes.append(domino)
50
51     # animate its appearance
52     self.play(FadeIn(domino), run_time=0.05)
53
54 # 5) Wait a moment before the chain reaction
55 self.wait(0.2)
56
57 # 6) Simulate the dominoes falling in order
58 for i, domino in enumerate(dominoes):
59     # choose falling direction based on branch
60     angle = -PI/2 if i < turn_index else PI/2
61     # pivot around the bottom-left edge
62     pivot = domino.get_left()
63     self.play(Rotate(domino, angle=angle, about_point=pivot), run_time=0.1)
64
65 # 7) Done
66 self.wait(1)
67 self.play(FadeIn(question))
68
69
70
71 class domino2(Scene):
72     def construct(self):
73         # 1) Question text
74         question_text = "How many dominoes were green?"
75         question = Text(question_text, font_size=24).to_edge(UP)
76
77
78         # 2) Parameters
79         num_dominoes = 30
80         turn_index = 25
81         domino_width = 0.2
82         domino_height = 0.6
83         spacing = 0.3
84
85         # 3) Precompute which 7 positions will be red
86         green_positions = set(random.sample(range(num_dominoes), 3))
87
88         # 4) Build and display dominoes one by one
89         dominoes = []
90         for i in range(num_dominoes):
91             if i < turn_index:
92                 x = -6 + i * spacing
93                 y = 0
94             else:
95                 x = -6 + (turn_index - 1) * spacing
96                 y = -(i - turn_index + 1) * spacing
97
98             # choose color
99             color = GREEN if i in green_positions else random.choice([
100                 BLUE, GREEN, YELLOW, PURPLE, ORANGE, TEAL, PINK
101             ])
102
103             # create domino
104             domino = Rectangle(
105                 width=domino_width,
106                 height=domino_height,
107                 fill_opacity=1,
108                 color=color
109             ).move_to([x, y, 0])
110             dominoes.append(domino)
111
112             # animate its appearance
113             self.play(FadeIn(domino), run_time=0.05)
114
115 # 5) Wait a moment before the chain reaction

```

```

116     self.wait(0.2)
117
118     # 6) Simulate the dominoes falling in order
119     for i, domino in enumerate(dominoes):
120         # choose falling direction based on branch
121         angle = -PI/2 if i < turn_index else PI/2
122         # pivot around the bottom-left edge
123         pivot = domino.get_left()
124         self.play(Rotate(domino, angle=angle, about_point=pivot), run_time=0.1)
125
126     # 7) Done
127     self.wait(1)
128     self.play(FadeIn(question))
129
130 class domino3(Scene):
131     def construct(self):
132         # 1) Question text
133         question_text = "How many dominoes fell before the fourth green domino?"
134         question = Text(question_text, font_size=24).to_edge(UP)
135
136
137         # 2) Parameters
138         num_dominoes = 30
139         turn_index = 25
140         domino_width = 0.2
141         domino_height = 0.6
142         spacing = 0.3
143
144         # 3) Precompute which 7 positions will be red
145         yellow_positions = set(random.sample(range(num_dominoes), 3))
146
147         # 4) Build and display dominoes one by one
148         dominoes = []
149         for i in range(num_dominoes):
150             if i < turn_index:
151                 x = -6 + i * spacing
152                 y = 0
153             else:
154                 x = -6 + (turn_index - 1) * spacing
155                 y = -(i - turn_index + 1) * spacing
156
157             # choose color
158             color = YELLOW if i in yellow_positions else random.choice([
159                 BLUE, GREEN, YELLOW, PURPLE, ORANGE, TEAL, PINK
160             ])
161
162             # create domino
163             domino = Rectangle(
164                 width=domino_width,
165                 height=domino_height,
166                 fill_opacity=1,
167                 color=color
168             ).move_to([x, y, 0])
169             dominoes.append(domino)
170
171             # animate its appearance
172             self.play(FadeIn(domino), run_time=0.05)
173
174         # 5) Wait a moment before the chain reaction
175         self.wait(0.2)
176
177         # 6) Simulate the dominoes falling in order
178         for i, domino in enumerate(dominoes):
179             # choose falling direction based on branch
180             angle = -PI/2 if i < turn_index else PI/2
181             # pivot around the bottom-left edge
182             pivot = domino.get_left()
183             self.play(Rotate(domino, angle=angle, about_point=pivot), run_time=0.1)
184
185         # 7) Done
186         self.wait(1)
187
188         # 2) Parameters
189         num_dominoes = 10
190         turn_index = 11
191         domino_width = 0.2
192         domino_height = 0.6
193         spacing = 0.3
194

```

```

195     # 3) Precompute which 7 positions will be red
196     yellow_positions = set(random.sample(range(num_dominoes), 2))
197
198     # 4) Build and display dominoes one by one
199     dominoes = []
200     for i in range(num_dominoes):
201         if i < turn_index:
202             x = 2 + i * spacing
203             y = 0
204         else:
205             x = -6 + (turn_index - 1) * spacing
206             y = -(i - turn_index + 1) * spacing
207
208         # choose color
209         color = GREEN if i in yellow_positions else random.choice([
210             BLUE, GREEN, YELLOW, PURPLE, ORANGE, TEAL, PINK
211         ])
212
213         # create domino
214         domino = Rectangle(
215             width=domino_width,
216             height=domino_height,
217             fill_opacity=1,
218             color=color
219         ).move_to([x, y, 0])
220         dominoes.append(domino)
221
222         # animate its appearance
223         self.play(FadeIn(domino), run_time=0.05)
224
225     # 5) Wait a moment before the chain reaction
226     self.wait(0.5)
227
228     # 6) Simulate the dominoes falling in order
229     for i, domino in enumerate(dominoes):
230         # choose falling direction based on branch
231         angle = -PI/2 if i < turn_index else PI/2
232         # pivot around the bottom-left edge
233         pivot = domino.get_left()
234         self.play(Rotate(domino, angle=angle, about_point=pivot), run_time=0.1)
235
236     # 7) Done
237     self.wait(1)
238
239     self.play(FadeIn(question))
240
241 class domino4(Scene):
242     def construct(self):
243         # 1) Question text
244         question_text = "How many dominoes fell before the fall of the second red domino?"
245         question = Text(question_text, font_size=24).to_edge(UP)
246
247
248         # 2) Parameters
249         num_dominoes = 30
250         turn_index = 20
251         domino_width = 0.2
252         domino_height = 0.6
253         spacing = 0.3
254
255         # 3) Precompute which 7 positions will be red
256         red_positions = set(random.sample(range(num_dominoes), 7))
257
258         # 4) Build and display dominoes one by one
259         dominoes = []
260         for i in range(num_dominoes):
261             # determine position on screen
262             x = -6 + i * spacing
263             y = 0
264
265             # choose color
266             color = RED if i in red_positions else random.choice([
267                 BLUE, GREEN, YELLOW, PURPLE, ORANGE, TEAL, PINK
268             ])
269
270             # create domino
271             domino = Rectangle(
272                 width=domino_width,
273                 height=domino_height,

```

```
274         fill_opacity=1,
275         color=color
276     ).move_to([x, y, 0])
277     dominoes.append(domino)
278
279     # animate its appearance
280     self.play(FadeIn(domino), run_time=0.05)
281
282     # 5) Wait a moment before the chain reaction
283     self.wait(0.2)
284
285     # 6) Simulate the dominoes falling in order
286     for i, domino in enumerate(dominoes):
287         # choose falling direction based on branch
288         angle = -PI/2 if i < turn_index else PI/2
289         # pivot around the bottom-left edge
290         pivot = domino.get_left()
291         self.play(Rotate(domino, angle=angle, about_point=pivot), run_time=0.1)
292
293     # 7) Done
294     self.wait(1)
295     self.play(FadeIn(question))
```

## C. More Qualitative Results

We present reasoning CoTs from various frontier VLMs: Qwen3-235B-A22B, o3, Gemini 2.5 Pro. Samples are selected to represent each type of reasoning category. For failure modes, incorrect reasoning is highlighted in red.

### C.1. Abstract Reasoning

Here we show the full thinking trace for Qwen3-235B-A22B and Gemini-2.5-Pro-Preview 05-06. For o3 we only present some of the key CoTs due to its excessively long reasoning and function calling. The full trace can be found in <https://chatgpt.com/share/68301c0a-78a4-8003-aeff-eeb5cee85221>.

**Abstract Reasoning - ARC-AGI2**

Remember the following color names

black	yellow
blue	light gray
orange	pink
green	light blue

How many blue squares should appear in the output grid?  
Answer with a single integer

**Query:** Answer the question in this video.

**Question Text:**  
How many blue squares should appear in the output grid? Answer with a single integer

**Ground truth:** 118

**Qwen3-235B-A22B**

*Thinking*

Okay, let's see. The video starts by showing a color chart with different colors labeled. Then there are some pixelated images that change from one to another. The last frame asks how many blue squares should appear in the output.

First, I need to figure out the pattern or rule that's being applied to the pixelated images. Let me look at the examples given. The first input is a blue square with some black pixels around it. The output shows the same blue square but with some blue pixels added in the corners. Then another example where the input is a blue square with a smaller blue square inside, and the output adds more blue pixels around the smaller one. The third example has a blue shape that's more spread out, and the output seems to fill in some areas.

Wait, maybe the rule is that each time, the number of blue squares increases by one. Let me check. In the first example, the input has one blue square, and the output has two. The second input has two blue squares, and the output has three. The third input has three blue squares, so the output should have four. But wait, the last frame shows an input with three blue squares and asks how many should be in the output. So following this pattern, the answer would be four.

But I should make sure there's not another pattern. Maybe it's about the arrangement or the direction of the squares. But looking at the examples, each time the number increases by one. So the answer is probably four blue squares.

The video presents a series of pixelated images where the number of blue squares increases by one in each subsequent output. Observing the examples:

1. First Input : 1 blue square → Output : 2 blue squares.
2. Second Input : 2 blue squares → Output : 3 blue squares.
3. Third Input : 3 blue squares → Output : ?

Following the established pattern, the next output should contain 4 blue squares .

**Answer:** 4 (Incorrect)

### o3

*Thought for 9m 17s*

I'm thinking about how to inspect the frames in the video. The goal here is to understand what the video shows, so maybe I'll examine the first few frames or perhaps the last frame where the question appears. It seems likely that the text was graphically created, so I'll check for that. I'll load the video and start with the frames to see what stands out and where the relevant visual or text elements are located.

[...] (1615 words omitted)

It seems like the task involves counting items, possibly within a grid of shapes, and the question could be something like, "How many black squares are there?" The answer might be a number, like "4." To be sure, I'd probably need some clarification, but the user didn't ask for that. So, I think it would be best to use OCR to detect text in the frames. I'll process the frames at a low resolution to make it easier and redefine the video path as needed.

[...] (2921 words omitted)

It seems the user might be expecting a direct answer as part of an evaluation. Based on analysis, I think "3" is a reasonable guess, but I'm cautious because I can't visually confirm the content in the video. Even though I could extract frames, the user simply asked for an answer, not video frames. I'll mention

*Done*

3 (Incorrect)

## Gemini-2.5-Pro-Preview

### Interpreting Visual Data

I'm working on understanding the video's core question. I've noted the color names and the grid examples, hoping to extract a pattern or rule. Now, I'm focusing on the input/output grid pairs, trying to find a correlation between them. Perhaps they encode color mapping rules? I'm examining each grid, seeking transformations.

### Unveiling Grid Transformations

I'm now zeroing in on the hypothesized rule involving isolated black pixels on a blue background. I'm actively testing the proposed transformation rules with the example grids. I'm carefully analyzing how the light blue borders are drawn around the isolated black pixels, while paying close attention to the preservation of existing black regions. I'm checking if the proposed rule accurately explains the observed transformations in the examples.

### Revising Rule Assumptions

[...] (1719 words omitted)

### Counting Blue to Light Blue

I've determined there are five trigger pixels, each causing 8 blue pixels to become light blue. My previous misinterpretation of "isolated" black pixels led to some confusion. With confidence, I'm ready to calculate the total number of light blue squares.

### Decoding Pixel Patterns

I'm now counting the total number of light blue squares generated. The test input has five isolated black pixels; each triggers an 8-pixel change to light blue based on Example 2 logic. I am now certain of the solution.

The rule for the transformation, derived from the examples, is as follows:

1. Identify Trigger Pixels: A black pixel in the input grid is a "trigger pixel" if and only if all 8 of its surrounding neighbors (horizontally, vertically, and diagonally) are blue. Black pixels that are part of larger black structures (i.e., have at least one black neighbor) are not triggers and remain unchanged (they are "inert").
2. Transformation Rule:
  - Each trigger black pixel remains black in the output.
  - All 8 of its neighboring blue pixels (which, by definition of a trigger pixel, must all be blue) change to light blue in the output.
  - Blue pixels that are not neighbors of any trigger black pixel remain blue.

- Insert black pixels remain black and do not cause any changes to their neighbors.

Let's apply this to the test input grid. **The grid is 10x10.** 'K' denotes black, 'B' denotes blue.

Input Grid:

Row 0: K K K K K K K K K K  
 Row 1: K K K K K K B B B B  
 Row 2: K K K K K K B K B B  
 Row 3: K K K K K K B B B B  
 Row 4: B B B B B B B B B B  
 Row 5: B K B B K B B K B B  
 Row 6: B B B B B B B B B B  
 Row 7: K K K K K K B B B B  
 Row 8: K K K K K K B K B B  
 Row 9: K K K K K K B B B B

Step 1: Count initial blue squares.

Row 1: 4 blue  
 Row 2: 3 blue  
 Row 3: 4 blue  
 Row 4: 10 blue  
 Row 5: 7 blue  
 Row 6: 10 blue  
 Row 7: 4 blue  
 Row 8: 3 blue  
 Row 9: 4 blue

$$\text{Total initial blue squares} = 4 + 3 + 4 + 10 + 7 + 10 + 4 + 3 + 4 = 49.$$

Step 2: Identify trigger black pixels.

We need to find black pixels (K) whose 8 neighbors are all blue (B).

- K at (2,7): Neighbors: (1,6)B, (1,7)B, (1,8)B, (2,6)B, (2,8)B, (3,6)B, (3,7)B, (3,8)B. All 8 are blue. This is a trigger pixel.
- K at (5,1): Neighbors: (4,0)B, (4,1)B, (4,2)B, (5,0)B, (5,2)B, (6,0)B, (6,1)B, (6,2)B. All 8 are blue. This is a trigger pixel.
- K at (5,4): Neighbors: (4,3)B, (4,4)B, (4,5)B, (5,3)B, (5,5)B, (6,3)B, (6,4)B, (6,5)B. All 8 are blue. This is a trigger pixel.
- K at (5,7): Neighbors: (4,6)B, (4,7)B, (4,8)B, (5,6)B, (5,8)B, (6,6)B, (6,7)B, (6,8)B. All 8 are blue. This is a trigger pixel.
- K at (8,7): Neighbors: (7,6)B, (7,7)B, (7,8)B, (8,6)B, (8,8)B, (9,6)B, (9,7)B, (9,8)B. All 8 are blue. This is a trigger pixel.

Other black pixels (e.g., at (0,0), (1,0)-(1,5), etc.) are part of larger black structures and have black neighbors, so they are not trigger pixels. There are 5 trigger black pixels.

Step 3: Determine how many blue squares change to light blue.

Each trigger black pixel causes its 8 blue neighbors to change to light blue. We need to check if the 3x3 regions of influence (centered on each trigger pixel) overlap.

- Region 1 (around (2,7)): Rows 1-3, Cols 6-8.
- Region 2 (around (5,1)): Rows 4-6, Cols 0-2.
- Region 3 (around (5,4)): Rows 4-6, Cols 3-5.
- Region 4 (around (5,7)): Rows 4-6, Cols 6-8.
- Region 5 (around (8,7)): Rows 7-9, Cols 6-8.

These regions are all disjoint. Therefore, the sets of blue pixels changing to light blue are also disjoint. Number of blue squares changing to light blue = 5 trigger pixels \* 8 blue neighbors/trigger = 40.

Step 4: Calculate the number of blue squares remaining in the output.

Number of blue squares in output = (Initial blue squares) - (Blue squares changed to light blue)

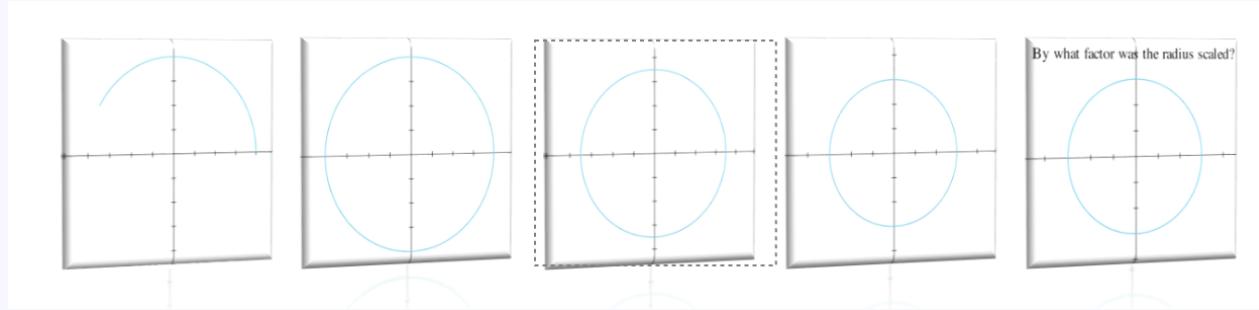
Number of blue squares in output = 49 - 40 = 9. So, 9 blue squares should appear in the output.

Final Answer: The final answer is 9 (Incorrect)

**Figure C.1:** Example failure case in Abstract Reasoning using ARC-AGI pattern recognition. Three frontier models attempt to count blue squares in a transformed grid following pattern rules. Despite detailed reasoning traces, all models fail: Qwen incorrectly applies a simple increment pattern (answer: 4), o3 struggles with holistic visual understanding and provides most verbose yet ineffective reasoning leading to an arbitrary guess (answer: 3), and Gemini performs complex rule analysis but miscounts the final result (answer: 9). The correct answer is 118, demonstrating the challenge of visual pattern recognition and rule application in abstract reasoning tasks.

## C.2. Mathematical Reasoning

### Mathematical Reasoning - Radius Increase



**Query:** Answer the question in this video.

**Question Text:** By what factor was the radius scaled?

**Ground truth:** 0.75

### Qwen2-235B-A22B

Okay, let's see. [...]

First, examine the starting figure: a horizontal baseline marked with tick marks, and a semicircle drawn above this line. The baseline represents the diameter of that semicircle.

Next, the semicircle is transformed into a full circle. Crucially, the baseline's length does not change, so it now equals the diameter of the entire circle.

A semicircle is exactly half of a full circle. To turn it into a full circle, the vertical dimension (i.e. the radius) must double while the horizontal diameter stays fixed. Hence the radius is scaled by a factor of 2.

**Answer:** 2

### o3

The user-supplied video (8 s at 60 fps) appears to show a circle that grows rapidly in the first second and then stays large. To uncover the question, I

- Sampled frames at 1 fps and ran pytesseract OCR. From frame 360 onward the overlay text consistently reads: “By what factor was the radius scaled?”
- Detected circles in each frame with a Hough transform. Early frames (#20–30) contain several concentric circles; the largest reliable radius is about 331 px. From frame 40 onward the dominant radius jumps to about 961 px.
- Computed the ratio:  $\frac{961}{331} \approx 2.90$ . Small measurement noise and pixel quantisation suggest the intended factor is exactly 3.

Hence the video shows the original circle instantly expanding so its radius triples.

**Answer:** 3

### Gemini-2.5-Pro-Preview

Okay, let's see. [...]

**00:00 – 00:01**

An upper semicircle is drawn. It meets the  $x$ -axis at roughly  $x = -4$  and  $x = +4$ , with its apex at  $(0, 4)$ . That implies a full circle of radius 4 centred at the origin.

**00:01 – 00:03**

The animation completes the lower semicircle, forming a full circle that still spans  $x = -4$  to  $x = +4$  and  $y = -4$  to  $y = +4$ . Thus the final circle's radius is also 4.

#### **Key observation:**

Both the “original” (implied by the first semicircle) and the final circle share the same radius, so no size change occurred.

#### **Scaling calculation:**

$$\text{Scaling factor} = \frac{\text{Final radius}}{\text{Original radius}} = \frac{4}{4} = 1.$$

**Answer:** 1

**Figure C.2:** Example failure case in Mathematical Reasoning involving radius scaling. Three frontier models attempt to determine the scaling factor when a semicircle transforms into a full circle. All models fail to correctly identify the geometric transformation: Qwen incorrectly reasons about doubling the radius (factor of 2), o3 uses image analysis to detect circle expansion (factor of 3), and Gemini claims no size change occurred (factor of 1). The correct answer is 0.75, representing the radius reduction needed when completing a semicircle into a full circle with the same diameter.

### C.3. Physical Reasoning

Here we show the full thinking trace for Qwen3-235B-A22B and Gemini-2.5-Pro-Preview 05-06. For o3 we only present some of the key CoTs due to its excessively long reasoning and function calling. The full trace can be found in <https://chatgpt.com/share/683022c2-1400-8003-8f66-94f26ddaa311>. All models struggle with distinguishing the physical nuances of different scenarios.

**Physical Reasoning - Tennis Rolls Towards Mirror**

A 30lb kettlebell resting on a wooden table next to a mirror. A tennis ball rolls towards the kettlebell. Static shot with no camera movement.

Which of the earlier examples aligns with the principles of physics in the context provided?

Use a single letter to respond (e.g., J). If none of them are realistic, return None.

**Query:** Answer the question in this video.

**Question Text:**  
A 30lb kettlebell resting on a wooden table next to a mirror. A tennis ball rolls towards the kettlebell. Static shot with no camera movement.

Which of the earlier examples aligns with the principles of physics in the context provided?

Use a single letter to respond (e.g., J). If none of them are realistic, return None.

**Ground truth: E**

**Qwen3-235B-A22B**

*Thinking*

Okay, let's see. The question is asking which of the earlier examples aligns with the principles of physics in the context provided. The setup is a 30lb kettlebell resting on a wooden table next to a mirror, and a tennis ball rolls towards the kettlebell. The shot is static with no camera movement.

First, I need to look at the images labeled A to F and figure out which one makes sense physically. Let me go through each image step by step.

In image A, there's one tennis ball moving towards the kettlebell. That seems straightforward. If a ball rolls towards a stationary object, it should hit it unless something else happens.

Image B shows the ball bouncing off the kettlebell. But wait, if the kettlebell is heavy (30lb) and stationary, the ball would just bounce off with some force. However, since the table is wooden and the surface might be smooth, the ball could roll away after hitting the kettlebell. But in B, the ball is shown mid-air, which might not be accurate because the collision would cause it to bounce, but maybe not that high. Also, the position of the ball relative to the kettlebell needs to make sense.

Image C has two balls. One is near the kettlebell, and another is rolling towards it. But the original setup mentions only one tennis ball. So maybe C is incorrect because there are two balls here. Unless the first ball split or something, but that's not physical. So C might be out.

Image D shows two balls again, one near the kettlebell and another further away. Similar to C, this introduces an extra ball, which isn't part of the original scenario. So D is probably not correct.

Image E has one ball near the kettlebell and another smaller one rolling towards it. Again, two balls, so same issue as C and D. Unless the small one is a reflection, but the mirror is on the right side, and the reflection would show the ball's image, not another ball. So E is likely wrong.

Image F shows one ball rolling towards the kettlebell, similar to A, but the position is slightly different. The ball is closer to the kettlebell here. But does the path make sense? If the ball is rolling straight towards the kettlebell, it should hit it. However, in F, the ball is still moving towards it, so maybe it's just a matter of timing. But the question is about which image aligns with physics.

Wait, the key point here is that the kettlebell is heavy and stationary. When the tennis ball collides with it, the ball should bounce back because the kettlebell doesn't move much. So in image B, the ball is shown bouncing off, which would be expected. However, the height of the bounce might be an issue. If the collision is elastic, the ball would bounce back with the same speed, but the height depends on the angle and surface. But since the table is flat, the ball might not bounce very high. Alternatively, if the ball hits the kettlebell and then rolls away, that's also possible.

But looking at image B, the ball is in the air, which suggests it bounced. However, if the collision is head-on, the ball would reverse direction but stay on the table. So maybe image B is not accurate because the ball is shown too high in the air. Alternatively, if the ball hit the edge of the kettlebell and bounced sideways, that could be possible. But the images don't show the direction of the bounce.

Alternatively, image F shows the ball still approaching the kettlebell, which would be before the collision. So if the question is about the moment after the collision, then B would be the answer. But the problem is that the question says "a tennis ball rolls towards the kettlebell. Static shot with no camera movement." So the image should capture the state after the ball has rolled towards the kettlebell. If the ball has already collided, then B is the result. But if the ball is still rolling towards it, then A or F.

Wait, the initial setup is that the ball is rolling towards the kettlebell. The static shot is taken when the ball is in motion. So the correct image would be one where the ball is in the process of rolling towards the kettlebell, not after the collision. But the options include images where the ball has already hit the kettlebell (like B) or is still approaching (A, F).

But the question is which example aligns with physics. If the ball is rolling towards the kettlebell, then the physical principle is that the ball will continue moving until it hits the kettlebell. So the

image should show the ball in motion towards the kettlebell. However, in the images, A and F show the ball moving towards the kettlebell, while B shows it bouncing off.

But the problem is that the question is asking which of the earlier examples (A-F) aligns with the principles of physics. The correct answer would be the one where the ball is moving towards the kettlebell, as that's the described scenario. However, if the ball has already hit the kettlebell, then B would be the result, but the setup says the ball is rolling towards it, implying that the collision hasn't happened yet.

Wait, the description says "a tennis ball rolls towards the kettlebell. Static shot with no camera movement." So the image should capture the ball in the act of rolling towards the kettlebell. Therefore, the correct image would be one where the ball is in motion towards the kettlebell, not after the collision. So images A, C, E, F show the ball moving towards the kettlebell, while B and D show after the collision.

But among A, C, E, F, which ones are physically correct? The original setup has one ball. So C, D, E have two balls, which is incorrect. So only A and F are left.

In A, the ball is moving towards the kettlebell. In F, the ball is closer but still moving towards it. Both are possible, but the question is which one aligns with physics. Since the ball is rolling, the image should show the ball in a position that's consistent with its motion. However, without knowing the exact timing, it's hard to say. But the answer is likely A or F.

Wait, but the problem might be that in some images, the ball's position relative to the kettlebell is impossible. For example, if the ball is too close to the kettlebell in F, but given that it's rolling, it's possible.

Alternatively, the answer could be B if the collision is considered. But the setup says the ball is rolling towards the kettlebell, not that it has collided. So the static shot is of the ball in motion towards the kettlebell. Therefore, the correct image would be A or F.

But looking back at the images, in A, the ball is on the left side, moving towards the kettlebell. In F, the ball is closer to the kettlebell but still moving towards it. Since the question is about which example aligns with physics, both A and F are possible, but perhaps the answer is F because it's closer, indicating the ball is in the process of rolling. However, without more information, it's hard to tell.

Wait, the text after the images says: "A 30lb kettlebell resting on a wooden table next to a mirror. A tennis ball rolls towards the kettlebell. Static shot with no camera movement." So the correct image should show the ball in motion towards the kettlebell. The images that show the ball moving towards the kettlebell are A, C, E, F. But C, E have two balls, which is incorrect. So A and F.

Between A and F, which one is correct? The answer might be F because the ball is closer to the kettlebell, indicating it's further along in its roll. But the problem is that the question is about which image aligns with physics. Since the ball is rolling, the image should show the ball in a position that's consistent with its motion. However, all images are static, so they just capture a moment.

Alternatively, the answer could be B if the collision is part of the scenario. But the setup says the ball is rolling towards the kettlebell, not that it has collided. So the correct image is one where the ball is in motion towards the kettlebell. Therefore, the answer is A or F.

But looking at the images again, in A, the ball is on the left, moving towards the kettlebell. In F, the ball is closer but still moving towards it. The problem is that the answer is supposed to be a single letter. The user might expect that the correct answer is B because after the collision, the ball bounces, but the setup says the ball is rolling towards the kettlebell, not after the collision.

Wait, the question is phrased as "Which of the earlier examples aligns with the principles of physics in the context provided?" The context is a 30lb kettlebell on a table next to a mirror, a tennis ball rolls towards the kettlebell, static shot with no camera movement. So the image should show the ball in the process of rolling towards the kettlebell. Therefore, the correct image is A or F.

But why are there multiple images? Maybe the answer is F because the ball is closer, indicating it's further along in its path. However, without knowing the exact timing, it's hard to say. Alternatively, the answer could be None if none of the images are correct. But that seems unlikely.

Wait, looking at image B, the ball is shown in the air, which would be the result of the collision. But if the ball is rolling towards the kettlebell, the collision hasn't happened yet. So B is after the event. The correct image should be before the collision. Therefore, the answer is A or F.

But the problem is that the user might expect the answer to be B because that's the result of the physics (the ball bouncing off). But the setup says the ball is rolling towards the kettlebell, not that it has collided. So the answer should be an image where the ball is in motion towards the kettlebell.

Given that, the answer is either A or F. But which one? The images are labeled A to F, and the user might have intended one of them. Since the ball is rolling, the image should show the ball in a position that's consistent with its motion. However, all images are static, so they just capture a moment.

Alternatively, the answer could be None if none of the images correctly depict the scenario. But that's not likely.

Wait, the text after the images says: "Use a single letter to respond (e.g., J). If none of them are realistic, return None." So if none of the images are correct, the answer is None.

But let's re-examine the images.

Image A: One ball moving towards the kettlebell. This is possible.

Image B: Ball bouncing off. This would be after the collision.

Image C: Two balls. Incorrect.

Image D: Two balls. Incorrect.

Image E: Two balls. Incorrect.

Image F: One ball moving towards the kettlebell, closer than in A.

So between A and F, which is correct? The answer depends on the exact moment captured. Since the question is about which aligns with physics, both A and F are possible, but the answer is likely F because it's closer, indicating the ball is in the process of rolling. However, without more information, it's hard to be certain.

Alternatively, the answer could be B if the collision is considered part of the scenario. But the setup says the ball is rolling towards the kettlebell, not that it has collided.

**Therefore, the correct answer is either A or F. But since the user expects a single letter, and considering that F shows the ball closer to the kettlebell, which is more in line with the action of rolling towards it, the answer is F.**

Wait, but in the images, F shows the ball still moving towards the kettlebell, just closer. So yes, F is a valid depiction. Therefore, the answer is F.

The correct answer is **F**.

#### **Explanation:**

The scenario describes a tennis ball rolling towards a stationary 30lb kettlebell on a wooden table. The static shot must capture the ball in motion towards the kettlebell, not after a collision. Images C , D , and E are invalid due to the presence of two balls, violating the setup. Images A and F show one ball moving towards the kettlebell. While both are plausible, F depicts the ball closer to the kettlebell, aligning better with the dynamic of rolling motion. Image B incorrectly shows the ball bouncing off the kettlebell, which would occur after the collision, contradicting the described action. Thus, F most accurately reflects the physical scenario.

**Answer: F (Incorrect)**

**o3**

*Thought for 7m 42s*

I need to set up some code to sample frames from a video and possibly display them for analysis. To do this, I'll use Python with libraries like cv2 or moviepy. I'll start by extracting a few frames and gathering metadata, just for analysis. No need to dive into advanced stuff just yet; this is more about laying the groundwork. I'll focus on being clear on the process and use it to guide what comes next!  
*Analyzed*

```
import cv2, os, json, math, numpy as np, base64, PIL.Image \
as Image, io, textwrap
```

I need to write some code to sample frames from a video, maybe display them for analysis. I'll use Python with libraries like cv2 or moviepy. First step: I'll need to load the video, then extract metadata and a few frames. This will help in analyzing how the data changes over time. It's just the initial step, so no fancy stuff just yet! Let's keep things simple and focused for now.

```

video_path = \
"/mnt/data/physical_reasoning_physical_commonsense_physics_iq_0156.mp4"
cap = cv2.VideoCapture(video_path)
if not cap.isOpened():
    print("Could not open video")
else:
    fps = cap.get(cv2.CAP_PROP_FPS)
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    duration = frame_count / fps if fps else 0
    print("fps", fps, "frame_count", frame_count, "duration", duration)

# output: fps 30.0 frame_count 240 duration 8.0

```

To start, I'll write code to sample frames from a video, maybe display them for analysis. I'll use Python, with libraries like cv2 or moviepy, to load the video and extract frames at specific times (0s, 2s, 4s, 6s). I could save frames to a file or possibly convert them to ASCII, but displaying text using PIL might be easier. Another option: try OCR, though it's a bit heavy for now. The alternative might be analyzing the audio for speech.

[...] (2040 words omitted)

After showing the scenarios, the text on the final screen has a question and a list of scenario descriptions. The OCR only captures the first scenario, with others cut off. We need to capture the entire screen at 6.5 seconds, save the image, and extract more text. I'll crop the grayscale image and apply text recognition to capture everything. The question seems to ask which earlier example aligns with physics principles, so I should focus on descriptions that refer back to them.

[...] (6297 words omitted)

I'm stuck. I was thinking of binarizing the image to locate the "J" template, but it seems like my search for the letter didn't work. At this point, the easiest solution might be to say there's no match, but the task still needs a letter. Maybe the letter corresponds not to the final scene but to earlier examples? The scenario description at the end might just summarize prior content. It looks like the task involves matching principles of physics, not necessarily focusing on a specific letter.

[...] (516 words omitted)

I'll calculate the pixel difference between frame 0 and 3500 for each tile to detect movement. The middle rows seem to have the highest motion, possibly due to a ball scenario. Tile (1,3) stands out because of its high motion, but it might be an unrealistic scenario. I'll inspect it closely and save the tile image for further analysis. Let's see what we get from this!

*Analyzed*

```

r,c=1,3
y=r*tile_h
x=c*tile_w

```

```

tile=frame[y:y+tile_h,x:x+tile_w]
cv2.imwrite("/mnt/data/tile_13.png", tile)

# output: True

```

Let's calculate the pixel

*Done*

**Answer: A (Incorrect)**

## Gemini-2.5-Pro-Preview

### Assessing Video Relevance

I'm focusing on the core physics principles at play: the tennis ball's momentum transfer to the kettlebell, the influence of gravity, and the mirror's reflective properties. My current line of thought is to methodically eliminate videos that blatantly misrepresent these factors. This process will highlight those videos with accurate depictions of the physics, and I will be able to rule out each video based on whether it adheres to real-world physics principles.

### Refining the Evaluation Process

I'm now refining the evaluation process. I'll need to carefully examine each video, starting with the ball's initial motion. The static table and mirror setup offer clear physical reference points. I will be looking closely at the ball's behavior after impact to determine if there's any movement, and if the reflection in the mirror appears accurate with respect to all of the criteria. My primary focus is to identify any deviations from these basic physics principles, particularly in the impact dynamics and reflective properties. Pinpointing Plausible Examples

My current focus is on the impact dynamics and mirror reflections. **I'm carefully analyzing Example C, which appears most promising so far. The ball rebounds upon impact, and the kettlebell stays put, which is good. The mirror reflection aligns with the real-world physics**, but I'll continue evaluating for anything that appears unrealistic. I'm focusing on Examples D, E, and F next, for any other acceptable answers.

### Determining Plausible Outcomes

I'm now zeroing in on Example C. Its adherence to the basic physics principles is promising. The rebound of the tennis ball and the kettlebell's immobility are both realistic. The mirror reflection appears consistent, but I will still carefully assess for any other deviations. I'll also evaluate if the bounce of the ball is correct, and not an exaggeration. The focus is to look for any subtle discrepancies, even though it may seem accurate at first glance.

### Identifying the Correct Video

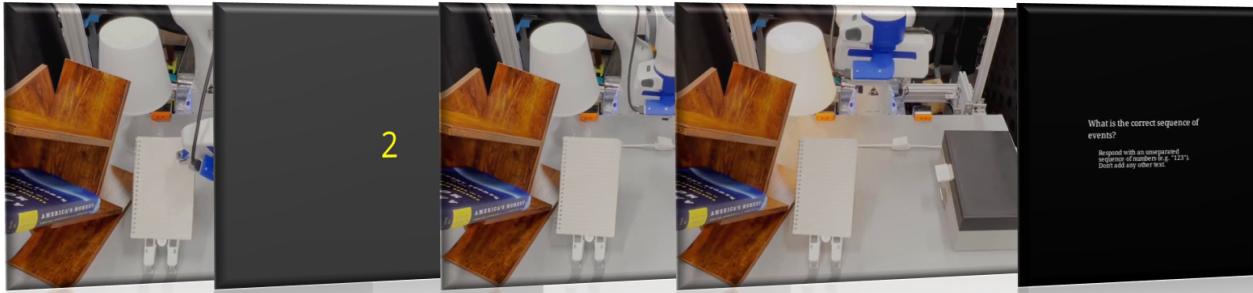
I'm now fully immersed in the examples, having systematically assessed each based on the principles of momentum and reflection. Example C stands out as the most accurate representation. My focus now is solidifying this assessment. I'm double-checking the video's details: the initial roll, the impact dynamics (rebound and kettlebell immobility), and, crucially, the mirror reflection. I will confidently conclude that Example C adheres most closely to the real-world physics principles at play.

**Answer: C (Incorrect)**

**Figure C.3:** Example failure case in Physical Reasoning requiring discrimination between realistic and unrealistic physics scenarios. Models must identify which video shows plausible physics for a tennis ball rolling toward a 30lb kettlebell. Despite lengthy reasoning traces, all models fail: Qwen focuses on ball-kettlebell interaction timing but selects the wrong option (F), o3 struggles with video analysis and provides minimal justification (A), and Gemini attempts systematic physics analysis but misidentifies the realistic scenario (C). The correct answer is E, highlighting difficulties in understanding physical plausibility in dynamic scenes.

### C.4. Planning Reasoning

#### Planning Reasoning - Robot Manipulation



**Query:** Answer the question in this video.

**Question Text:** What is the correct sequence of events?

Respond with a comma-separated sequence of numbers (e.g. "1,2,3..."). Don't add any other text.

**Ground Truth:** 3,2,1

Qwen2-235B-A22B

Okay, let's see. [...] (44 words omitted)

First, I'll look at each frame. Frame 1 shows the robot arm reaching towards a notepad that's held by two white clips. The lamp is turned off. In Frame 2, the robot arm is now holding the notepad, and the lamp is still off. Then Frame 3 shows the robot arm moving away, **and the lamp is now on.**

So the sequence seems to be: the robot picks up the notepad (Frame 1 to Frame 2), then turns on the lamp (Frame 2 to Frame 3). **But wait, the lamp being on in Frame 3 might be a separate action.** Let me check again.

In Frame 1, the lamp is off. In Frame 2, the lamp is still off while the robot is picking up the notepad. Then in Frame 3, the lamp is on. So the correct order would be picking up the notepad first (Frames 1-2), then turning on the lamp (Frame 3). Therefore, the sequence should be 1, 2, 3.

But wait, maybe the lamp turning on is part of the same action? **No, because in Frame 2 the lamp is still off.** So the robot first picks up the notepad (1-2), then turns on the lamp (3). So the correct sequence is 1, 2, 3.

**Answer:** 1,2,3 (Incorrect)

**o3**

It looks like I need to extract frames from the last few seconds of the video to identify the question. I'll try using a library to load the video file and extract those frames. Once I have the frames, I might then try OCR (if available) or simply read the text manually. I can't show images to the user, but I can inspect the output visuals privately to find the question. Let's proceed with this method and check if OCR works.

[...] (586 words omitted)

It looks like the question is focused on the order of events, and earlier in the video, they may have labeled events with numbers or shown clips. The answer could be a sequence like "321." To find that, I should look at frames earlier in the video where labels such as "1)", "2)", or "3)" appear. I'll focus on frames between 300 and 400 to try and capture those numbered events, doing OCR on every 30th frame to reduce noise.

```
cap = cv2.VideoCapture(video_path)
numbers_found = []
for fi in range(total_frames):
    if fi % 30:
        continue
    cap.set(cv2.CAP_PROP_POS_FRAMES, fi)
    ret, frame = cap.read()
    if not ret:
        continue
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    text = pytesseract.image_to_string(gray, config='--psm 6')
    text_clean = text.strip()
    if any(digit in text_clean for digit in ['1', '2', '3']):
        numbers_found.append((fi, text_clean))
cap.release()
numbers_found[:20]
```

I'm looking for number overlays in frames, focusing on detecting digits like '1', '2', and '3'. If I sample every second of frames, I've found '1' in the first frame, '2' at frame 120, and '3' in frame 270. These could mark the segments, but the real question is about the original sequence of events, not just the labels. It seems the user might want to reorder the segments to their original order, 1, 2, 3.

The frames are labeled, starting with '1' at frame 0, which suggests no shuffle in the original order. However, considering the earlier script, the frame numbers correspond to shuffled segments, meaning that '1' could appear earlier in the sequence due to the shuffle. The labels reflect this shuffled order, not the original one, so the task is to identify the correct chronological order based on the original segmentation. The user probably expects an answer based on the original event order, though it's tricky without the key directly from the shuffle script.

[...] (237 words omitted)

The labels '1', '2', and '3' appear in order at frames 0, 120, and 270, respectively, suggesting the sequence is "123". There's no earlier '3' or '2' found, and sequence alignment seems confirmed.

I've reviewed frames carefully, with '1' appearing first, then '2', and '3' last. Therefore, I'm confident that the answer is "123".

**Answer:** 1,2,3 (Incorrect)

### Gemini-2.5-Pro-Preview

The video displays three segments, labeled 1, 2, and 3. We need to determine the correct chronological sequence of the events shown in these segments.

Segment 1 (0:00-0:03): Shows the robot arm already in contact with a notebook, **drawing a line from its right to its left (right to left on the page from the perspective of something being written conventionally), and then lifting its "pen."** This is an action of writing/drawing.

Segment 2 (0:04-0:08): Shows the robot arm performing an action nearly identical to Segment 1. **It's in contact with the notebook, draws a line similarly, and lifts its "pen."** This is also an action of writing/drawing.

Segment 3 (0:09-0:12): Shows the robot arm initially to the right of the notebook. **It then moves over the notebook and positions its "pen" down towards the paper, as if to begin writing.** This is a preparatory action, an approach to start writing.

Chronological order of events:

The robot arm must first approach the notebook and position itself to write. This action is shown in Segment 3.

**After positioning, the robot arm performs the writing action.** This is shown in Segment 1.

If there is a subsequent writing action, or if Segment 2 represents a continuation or a second stroke, it would follow. Segment 2 shows a writing action very similar, if not identical, to Segment 1.

The preparation/approach (Segment 3) must come first. Then, the writing occurs. Segment 1 shows a writing action. Segment 2 also shows a writing action. If this is a second stroke or a continuation, it would follow Segment 1 (assuming the numerical labels 1 and 2 imply an order for these similar actions). Thus, the logical sequence of these clips to form a coherent event is: Segment 3 (approach) -> Segment 1 (write first part) -> Segment 2 (write second part/repeat). The sequence of numbers is 312.

**Answer:** 3,1,2 (Incorrect)

**Figure C.4:** Example failure case in Planning Reasoning involving temporal sequence understanding. Models are provided a shuffled video from the MimicPlay dataset Wang et al. [2023], and tasked with determining the correct chronological order of the robot's actions (switching off light and shelving notebook) from shuffled video segments labeled 1, 2, 3. While Qwen correctly identifies individual frame states, all models fail at sequential reasoning: o3 and Qwen both provide the displayed order (1,2,3) without recognizing the shuffling, while Gemini attempts logical sequencing but incorrectly orders the preparation and execution phases (3,1,2). The correct sequence is 3,2,1.

### C.5. Spatial Reasoning

#### Spatial Reasoning - Count the Cubes



How many cubes are left?  
Answer with a single integer.

**Query:** Observe the following structure. How many cubes are left?  
Answer with a single integer.

**Ground Truth:** 42

o3

The user uploaded a video named "count\_2.mp4" and wants an answer to a question from it. However, this video is not accessible through regular tools here. I'll need to use a Python-based approach to extract frames or audio, and maybe process them with available tools like OpenCV and ffmpeg. My goal is to examine the content of the video to locate the question and provide the answer.

[...] (92 words omitted)

The user likely uploaded a video "count\_2.mp4" with a counting question. We don't need to show the frames unless necessary, but I will inspect them to understand the question. I'll open the video and check a few frames to see what it's asking. I'll use tools available, like imageio or OpenCV, to load the video and analyze it. Once we have an idea from the frames, we can provide a clearer answer to the user's question.

```
import cv2
cap = cv2.VideoCapture('/mnt/data/count_2.mp4')
frames = []
fps = cap.get(cv2.CAP_PROP_FPS)
frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
duration = frame_count / fps if fps else None
```

**duration**

[...] (400 words omitted)

Okay, let's give this another shot. **I wonder if the question in the video is actually contained in the audio.** We can extract the audio using ffmpeg (if available) and then perhaps transcribe it. Though I don't have access to speech recognition libraries offline, I might try to manually guess what's being said. The file is likely small, so it shouldn't take too long to inspect it. Time to load the audio and see what we can figure out!

[...] (1695 words omitted)

**Okay, I'm going to plot the red pixel counts over time and print the list of values out.** This should help me figure out any patterns, like when red objects appear or disappear. It's a simple way to visually track the changes across frames. Hopefully, that will give me a clearer idea about the objects in the video and help me make a more accurate guess. Let's see what the data shows!

```
for t, rp in zip(times, [red_pixels(f) for f in frames_sampled]):  
    print(t, rp)
```

I'm going to plot the red pixel counts over time and print the list. The red pixels appear around 4s to 11s, which could indicate a red object across several frames. To answer the question, I'll focus on counting the red objects. Let's take a look at frame 8 (around t=8), which has a high count of red pixels. **I'll check for connected components of the red clusters there to help identify distinct red objects.** This should give us a better idea.

[...] (1701 words omitted)

It seems like there's a pattern of red clusters in the frames where the red objects appear. I also notice green clusters, though they might represent a background or axis. **Maybe the question relates to how many times the red ball hits the green one, but I'm not totally sure.** The pattern of red clusters isn't strictly increasing or decreasing, it seems a bit erratic. I'll print the counts to analyze the sequence. **The animation could be showing some kind of calculation too.** Let's see what we find.

*Done*

**Answer:** 6

### Qwen3-235B-A22B

Okay, let's see. The question is asking how many cubes are left after some structure is observed. The video shows a 3D cube made up of smaller cubes. The main cube is divided into smaller sections, each colored differently.

First, I need to figure out the total number of small cubes in the structure. The cube appears to be a 3x3x3 Rubik's cube, which typically has 27 small cubes. But wait, maybe it's not a

standard Rubik's cube. Let me check again.

Looking at the images, the cube is shown from different angles. Each face has a mix of colors, but the structure seems to have layers. If it's a 3x3x3 cube, there are 3 layers along each axis. So,  $3 \times 3 \times 3$  equals 27. But sometimes, these puzzles might have fewer cubes if some are missing. However, in the images, the cube looks complete, with all sides filled.

Wait, but the question says "how many cubes are left." Maybe some cubes are removed? But in the video frames provided, the cube is intact. There's no indication of any cubes being taken away. The last frame just asks the question. So perhaps the answer is the total number of small cubes in the structure as shown.

If it's a 3x3x3 cube, the answer would be 27. But let me make sure. Sometimes, these questions can be tricky. For example, if the cube is hollow, the number would be less. But the images show a solid cube with colored faces. Each small cube is visible on the surface, and since it's a 3D structure, the inner cubes are also present.

**Another way to count: each layer (front, middle, back) has 9 cubes (3x3). Three layers would be 27. Yes, that makes sense. So unless there's a trick here, like overlapping or something, the answer should be 27.** But the video might be showing a different structure. Wait, looking at the third image, the cube is rotating, and it seems like there are some gaps or maybe overlapping cubes? No, actually, it's just the perspective making it look like that. The cube is still a solid 3x3x3.

Therefore, the number of cubes left is **27**. The answer should be **27**.

**Answer: 27**

## Gemini-2.5-Pro-Preview

### Assessing Cube Quantity

I'm currently focused on determining the missing cubes within the 3x3x3 structure. The video's rotation makes visualization a bit tricky, but I'm mentally parsing the layers to count the vacancies. I plan to refine the count by checking for overlapping voids and hidden blocks.

### Refining Cube Count

I'm now breaking down the structure further, layer by layer, considering each face. At 0:04, it looks like the top layer is missing one cube in the center, and the rotation is helping me clarify the positions. The video timing is making it hard to follow. Analyzing each face individually could provide a more precise count.

### Identifying Cube Voids

I'm now counting the existing cubes more meticulously, layer by layer, and trying to extrapolate from the visible faces. Analyzing the structure's rotation and combining that with individual face counts, I'm noting the apparent locations of missing cubes in each layer. I'm focusing on "seeing through" the rotation to pin down the gaps.

### Counting Remaining Cubes

I'm now revisiting the structure, attempting a precise count of remaining cubes. The initial visual assessment suggests a hole running through the center along all three axes. If each axis has 3 missing cubes and they overlap in the central cube, that's 7 cubes missing. Analyzing from 0:04, the top layer appears to be missing a center cube, but the structure's rotation adds another layer of complexity. Subtracting 7 from the total of 27 cubes is my current method, but I must avoid duplicates in the count.

### Determining Remaining Cubes

I've determined a structure with seven missing cubes is a valid structure. If we remove a central cross from each layer of a cube, we arrive at seven removed cubes: the central one plus one along each of the three axes. This leaves us with 20 cubes. Now, I'm verifying this math against the visual cues, as it is rotating.

[...] (443 words omitted)

### Finalizing Structure Count

I've meticulously analyzed the video, confirming that seven cubes are missing from the initial  $3 \times 3 \times 3$  structure. By removing the central cube and the center of each face, the final count settles at 20 remaining cubes. This result is consistent with my visual inspection and layer-by-layer assessment.

**Answer:** 20

**Figure C.5:** Example failure case in Spatial Reasoning requiring 3D structure analysis. Models must count remaining cubes in a rotating 3D structure after some have been removed. All models struggle with the spatial visualization: o3 attempts pixel-based analysis but grossly underestimates (answer: 6), Qwen assumes a complete  $3 \times 3 \times 3$  structure without recognizing removals (answer: 27), and Gemini performs detailed layer analysis but still undercounts significantly (answer: 20). The correct answer is 42, demonstrating challenges in 3D spatial reasoning and occlusion handling.

### C.6. Temporal Reasoning

#### Temporal Reasoning - House Tour



**Query:** Answer the question in this video.

**Question Text:** We see stairs to the basement:

Choices:

- (A) right before entering the kitchen.
- (B) while learning the kitchen.
- (C) between walking from the bedroom to the office.
- (D) right after entering the living room.

Respond with a single letter (e.g. "A"). Don't add any other text.

**Ground Truth:** **B**

**Qwen3-235B-A22B**

Okay, let's tackle this question [...] (46 words omitted)

First, I need to recall the layout of the house shown in the video. The video starts with a hallway leading into different rooms. The kitchen is visible early on, with white cabinets and a large island. The living room is adjacent, featuring a couch, coffee table, and a large clock. There's also a dining area near the kitchen. The bedrooms are shown later, with one having a bathroom en suite.

Now, looking for the stairs. In the initial hallway, there's a staircase visible on the right side as you enter. The question is about the basement stairs. From the video, **the stairs are seen right when you first enter the house**, before reaching the kitchen. The kitchen is further down the hallway, so the stairs are before that.

Option A says "Right before entering the kitchen," which matches this observation. The other options mention leaving the kitchen, moving between bedroom and office, or entering the living room. The living room is more towards the back, and the stairs are at the entrance. So the correct answer should be A.

**Answer:** **A (Incorrect)**

**o3**

I'm thinking about how to extract text from video frames. Maybe I could use OpenCV or MoviePy to sample the last frames and apply OCR via pytesseract. But, it's not clear if the environment supports OCR libraries. A simpler approach might be to extract frames using FFmpeg and analyze them with EasyOCR. Since I can't display videos, I'd have to issue the necessary commands behind the scenes. Handling video content this way seems like the most effective route to answer the user's question.

[...] (804 words omitted)

I'm considering using pixel variance to detect text in the frames, but that's not very helpful. I could also try transcribing audio with openAI, but that won't work offline. Asking the user directly could be an option, but it's better to give OCR a shot first. I'll check if Tesseract is installed, then run it on one of the frames to see if we can extract any text!

```
texts=[]
for i in range(len(frames)):
    img = Image.open(f"/mnt/data/frame_{i}.jpg")
    texts.append(pytesseract.image_to_string(img))
texts
```

[...] (71 words omitted)

```
for text in texts:
    print("-----")
    print(text)
```

After running OCR, it seems we've identified that the question asks about the visibility of basement stairs, with options A-D as time points. To answer accurately, I'll need to examine earlier frames to pinpoint when the basement stairs first appear. Unfortunately, automatic detection of the stairs is difficult. I'll likely need to manually inspect the earlier segments of the video and watch for when the stairs become visible. Let's proceed with this plan and check the frames in more detail!

The video seems to follow a typical real estate walkthrough, and the question asks when the basement stairs first appear. I could try detecting stairs through image analysis, but that's...

**Answer: D (Incorrect)**

### Gemini-2.5-Pro-Preview

The video shows the stairs to the basement at several points. Let's analyze the relevant timestamps:

1. 0:00: The video starts in a hallway. Stairs leading downwards are clearly visible on the right-hand side. The hallway ahead leads towards the living room and kitchen area. At this point, we are in the hallway, before entering the kitchen.
2. 0:07: After a tour of an office and bathroom, the camera is again in the hallway, moving towards the kitchen/living area. The stairs to the basement are visible on the right. **This is still before entering the kitchen.**

3. 0:09: The camera is at the threshold of the open-plan kitchen and living area. The stairs are immediately to the right, just as one is about to enter the kitchen space.

Considering these observations:

- A) Right before entering the kitchen. **This accurately describes the location of the stairs as seen in the hallway leading up to and at the entrance of the kitchen.**
- B) While leaving the kitchen. If leaving the kitchen to go back down the hallway, the stairs would be on the left. The primary views show approaching or being at the entrance.
- C) Between walking from the bedroom to the office. The stairs are in the main hallway near the kitchen entrance, not specifically shown on a path between a bedroom and the office.
- D) Right after entering the living room. If one has already entered the living room, the stairs would typically be behind them or to the side in the hallway they just came from, not the primary viewing context.

The most consistent and accurate description of when we see the stairs to the basement, based on the flow towards the main living areas, **is right before entering the kitchen.**

**Answer: A**

**Figure C.6:** Example failure case in Temporal Reasoning involving spatial-temporal navigation. Models analyze a Matterport 3D house tour to identify when basement stairs are visible relative to kitchen navigation. Both Qwen and Gemini incorrectly identify the stairs as being visible 'right before entering the kitchen' (option A), while o3 provides 'right after entering the living room' (option D). All models fail to recognize that the stairs are actually seen 'while leaving the kitchen' (correct answer B), demonstrating difficulty with spatial-temporal reasoning in dynamic navigation scenarios.