VISUAL PROMPT ENGINEERING FOR VIDEO MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

In the age of foundation models, a model is only as good as its prompt. For this reason, prompt engineering has become an essential technique for improving language model performance. Since video models are currently becoming foundation models for visual tasks (e.g., visual reasoning), we here ask whether they similarly benefit from *visual* prompt engineering: automatically modifying the task image to improve model performance. For example, for a visual physics reasoning task ("Where does the ball land, after passing a set of obstacles?"), an abstract sketch-like scene can be turned into a photorealistic version with a simple call to an image editing model. We find that visual prompt engineering, or VIPE for short, improves video reasoning performance across tasks. In fact, for video models, visual prompt engineering can be even more effective than classic text-based prompt engineering or test-time scaling. Ultimately, just as text-based prompt engineering systematically improves language model performance, visual prompt engineering can serve as a simple, compute-efficient approach to elicit better visual reasoning performance from video models. Example videos on our project page.

1 INTRODUCTION

Prompt engineering has become a crucial component for improving language model performance (Khattab et al., 2022; White et al., 2023; Giray, 2023; Reynolds & McDonell, 2021; Khattab et al., 2024; Yuksekgonul et al., 2024). By carefully optimizing the text input, one can systematically elicit better outputs without modifying the model itself. Indeed, prompt design significantly affects model accuracy across a range of tasks, from question answering to code generation (Marvin et al., 2023; Meskó, 2023). This is based on the observation that whenever a model's input space is rich enough, there is an opportunity to optimize that input for better downstream performance. Is this principle limited to language models, or does it extend to other modalities, such as the visual domain?

Generative video models have been proposed as foundation models for visual reasoning (Wiedemer et al., 2025; Yang et al., 2024; Acuaviva et al., 2025; Wang et al., 2026a; Zeller et al., 2026; Newman et al., 2026; Guo et al., 2026; Wang et al., 2026b; Li et al., 2026; Cheng et al., 2026; Tong et al., 2026). Recent works have demonstrated that they can tackle surprisingly challenging tasks, from maze solving to logic puzzles and physics simulations (Wiedemer et al., 2025). While far from perfect, their abilities are rapidly improving. Unlike language models, which receive only text input, video models accept a *text* prompt (describing what should happen) and a *visual* prompt (typically an image that becomes the first frame of the generated video). The model then generates a video that depicts a solution attempt for the task.

Yet, while text prompts have been extensively studied and optimized for LLMs and VLMs (White et al., 2023; Reynolds & McDonell, 2021; Wang et al., 2023a; Gu et al., 2023), the visual input to video models has been treated as fixed. This asymmetry is surprising. For visual reasoning tasks, the image is arguably the dominant conditioning signal: it accounts for many tokens and defines the visual context, spatial layout, objects, and constraints that the model must reason over. Prior work has explored modifying visual inputs for image classifiers, for instance by overlaying shapes onto input images (Shtedritski et al., 2023; Yao et al., 2024), learning pixel-level perturbations for CLIP (Bahng et al., 2022; Zeller et al., 2024), or inserting learnable tokens into vision transformers (Jia et al., 2022). Prompting via inpainting (Bar et al., 2022; Wang et al., 2023b), visual cloze tasks (Li et al., 2025), or visual instructions (Lin et al., 2025) has also been explored. However, engineering the visual input to improve video model performance remains unexplored.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

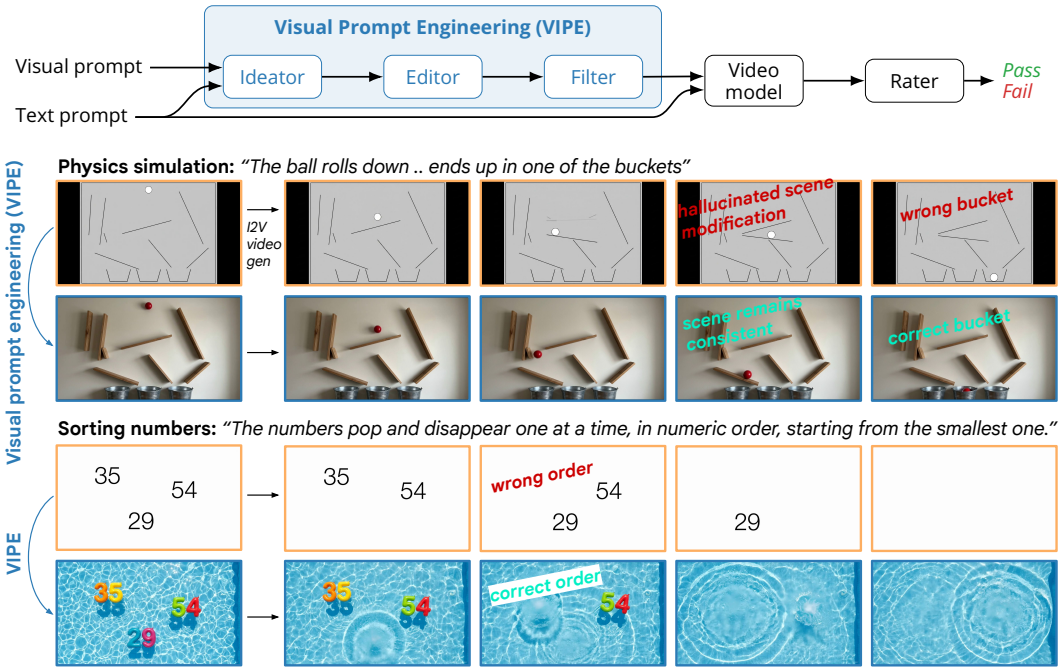


Figure 1: **Visual prompt engineering (VIPE)** starts with a visual prompt (e.g., a ball-and-ramp sketch image) and a text prompt (e.g., "The ball rolls down [...] ends up in one of the buckets"). Instead of directly generating a video, an *ideator* proposes variants of the task (e.g., replacing the sketch ball with a realistic billiard ball, and the sketch obstacles with wooden shelves). The task variants are then generated with an image *editor* (e.g., Nano Banana), followed by an optional *filter* for quality control (e.g., a VLM). After completing visual prompt engineering, the video model generates a video for the modified scenario, which is then scored as a pass/fail. See videos on our anonymous project page.

We address this gap by asking: *Can we improve visual reasoning capabilities of video models by transforming the input task image?* We call this *visual prompt engineering*, or VIPE. Concretely, VIPE transforms the visual prompt (e.g., replacing an abstract ball-and-ramp sketch with a photorealistic scene of billiard balls rolling down wooden shelves) using an image editing model, and then passes the transformed image to the video model for reasoning (e.g., generating a visual prediction for where the ball would roll). The underlying task remains identical; only its visual presentation changes, with the goal of providing a visual context that the model can reason over more effectively.

Our **main contributions** are as follows:

- We introduce visual prompt engineering (VIPE), a simple approach that consistently improves visual reasoning capabilities of video models.
- We show that VIPE is easily automated and an effective test-time scaling strategy.
- We demonstrate that refining the visual context is often more effective for video models than optimizing text instructions.
- We reveal that video models systematically prefer photorealistic context over abstract inputs: sketch-based benchmarks can significantly underestimate model capabilities.

2 Visual Prompt Engineering (VIPE)

Definition of visual prompt engineering for video models We define a task sample as a pair (x_i, t_i), where $x_i \in \mathcal{X}$ is a visual prompt (here: always an image) and $t_i \in \mathcal{T}$ is a text prompt. A video model attempts to solve each task sample by generating a video. Visual prompt engineering

(VIPE) aims to improve downstream reasoning performance by replacing an original image x_i with a variant v_i^* without altering the task's underlying logic.

Step 1: Ideator Conditioned on an ideation prompt $t_{\text{ideate}} \in \mathcal{T}$ and k task samples, the ideator (say, a human or a language model) $I : (\mathcal{X} \times \mathcal{T})^k \times \mathcal{T} \to \mathcal{T}$ describes a visual edit in natural language as

$$t_{\text{edit}} \sim I\left(X, t_{\text{ideate}}\right), \quad \text{where} \quad X = \{(x_i, t_i)\}_{i=1}^{k}. \tag{1}$$

The edit instructions t_{edit} are only constrained by the ideation prompt t_{ideate}, which ensures that the core task remains unchanged, e.g., by specifying what scene elements may not change.

Step 2: Editor Let $E : \mathcal{X} \times \mathcal{T} \to \mathcal{X}$ be a general image editor (in our case an image editing model; though in principle this could also be a human), conditioned on edit instructions t_{edit}. For an original prompt image x_i, we sample m candidate variants

$$V_i = \{v_{i,j}\}_{j=1}^{m}, \quad \text{where} \quad v_{i,j} \sim E\left(x_i, t_{\text{edit}}\right). \tag{2}$$

Step 3: Filter (optional) To select the highest quality variant, let $S : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a scoring function (e.g., a vision-language model) that evaluates a candidate's quality and its faithfulness to the original prompt image (x_i). The optimal candidate v_i^* is selected by maximizing this scoring function:

$$v_i^* = \arg\max_{v \in V_i} S\left(v, x_i\right) \tag{3}$$

Due to its stochastic nature, the entire pipeline can be repeated to obtain n variants $v_{i,j}^*$ for the same original visual prompt x_i, e.g., for ensembling.

3 TEXT-BASED PROMPT ENGINEERING HELPS LANGUAGE MODELS FOR LANGUAGE TASKS. DOES VISUAL PROMPT ENGINEERING HELP VIDEO MODELS FOR VISUAL TASKS?

We start our investigation with visual reasoning on the *VPCT* (camelCase12, 2025) dataset. It requires predicting whether a white ball will drop into buckets 1, 2, or 3 after rolling down a series of ramps (represented as black lines). Samples are shown in Fig. 2. The dataset is interesting because it requires models to reason through a challenging visual setup while taking into account which physical forces apply. At the same time, the problem can be represented in its original sketch-style setup, or alternatively in a photorealistic way. We here ask whether these choices influence how well video models perform on this task. To this end, we take the original sketch dataset and perform visual prompt engineering (VIPE) to make it look photorealistic. Until recently, changing the style of an entire dataset would have been prohibitively expensive, requiring days of work by a professional. Fortunately, we can now simply preprocess the dataset with an image editing model. For *VPCT*, we instantiate the editor E with Nano Banana Pro (Google, 2026b) and propose an edit instruction t_{edit} ourselves: *"Transform this into a realistic photograph, without changing the camera perspective that is facing the wall dead on. Replace the white circle with a small red billiard ball. Replace each black line by a wooden shelf board of exactly the same length, orientation, and placement. Do not add anything else."* Filtering is performed with Gemini 3.1 Pro (Google, 2026a), selecting the best out of $m = 5$ proposal variants. The resulting dataset was verified by the authors to ensure that VIPE did not accidentally make the problem easier than the original dataset.

After completing the visual prompt engineering step, we can now compare video model performance before and after VIPE. Detailed prompts and evaluation details are in Sec. C; essentially the video model is asked to simulate the ball's trajectory, and an autorater detects the bucket in which the ball lands. The results are shown in Fig. 2: Overall, both open- and closed-source models benefit from increased realism via visual prompt engineering. For example, performance of Wan2.2 is close to chance (=random guessing) in the baseline sketch condition for both the TI2V and I2V checkpoints and across English and Chinese prompts. Since Wan performance may differ between English and Chinese text prompts, we evaluate across both languages. TI2V is a 5B checkpoint that can perform both T2V (not used here) and I2V; while the I2V checkpoint is a MoE model with 14B active parameters (Wan et al., 2025), see also Sec. D. With visual prompt engineering, Wan

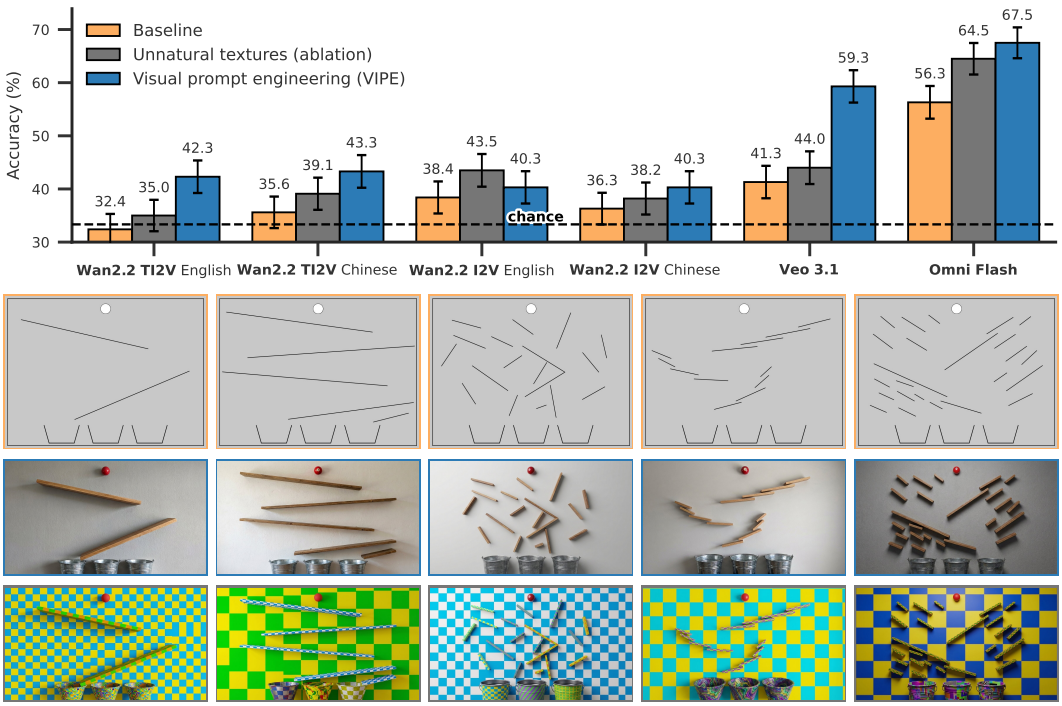


Figure 2: **Visual prompt engineering improves physics reasoning on *VPCT*.** The barplots visualize performance of video models on a ball-and-ramp dataset (goal: predict the bucket that the ball will fall into) across three conditions. Samples from each condition are visualized below. **Top row**: original sketch-like dataset samples (realism ✗ 3D ✗). **Middle row**: visual prompt engineering samples (realism ✓ 3D ✓). **Bottom row**: unnatural textures ablation (realism ✗ 3D ✓). Each barplot represents the average across 10 runs of the entire dataset (a total of 1,000 samples); whiskers indicate 95% confidence intervals.

performes above chance across all conditions. For Veo 3.1 (Google, 2026d), which already starts with higher baseline performance, the effect of visual prompt engineering is even more pronounced, taking it from 41.3% to 59.3%. Finally, Omni Flash (Google, 2026c) improves from 56.3% to 67.5% via VIPE. In short, visual prompt engineering improves video reasoning on this task—similar to how text-based prompt engineering improves language model performance on language tasks. Visual prompt engineering is, in many ways, the natural and obvious extension of text-based prompt engineering. Given its effectiveness, we find it all the more surprising that it isn't common practice for video models yet.

Takeaway 1 Similar to how *text-based* prompt engineering can improve language model performance on language tasks, *visual* prompt engineering can systematically improve video model performance on visual reasoning tasks.

Ablation: unnatural textures In order to find out whether it is indeed the photorealism that improves video model reasoning, or alternatively the difference between 2D sketch and 3D-looking input, we performed an ablation where samples have completely unrealistic textures (=no realism), while still having depth and therefore "looking 3D". In Fig. 2, most models do not show a significant difference between the baseline and unnatural texture condition, suggesting that 3D alone is insufficient and it is indeed visual realism that drives the performance improvement.

4 How can visual prompt engineering be automated?

The *VPCT* results from Sec. 3 demonstrate that a good visual prompt can systematically improve video reasoning performance. However, we may not always have a good prior intuition on what con-

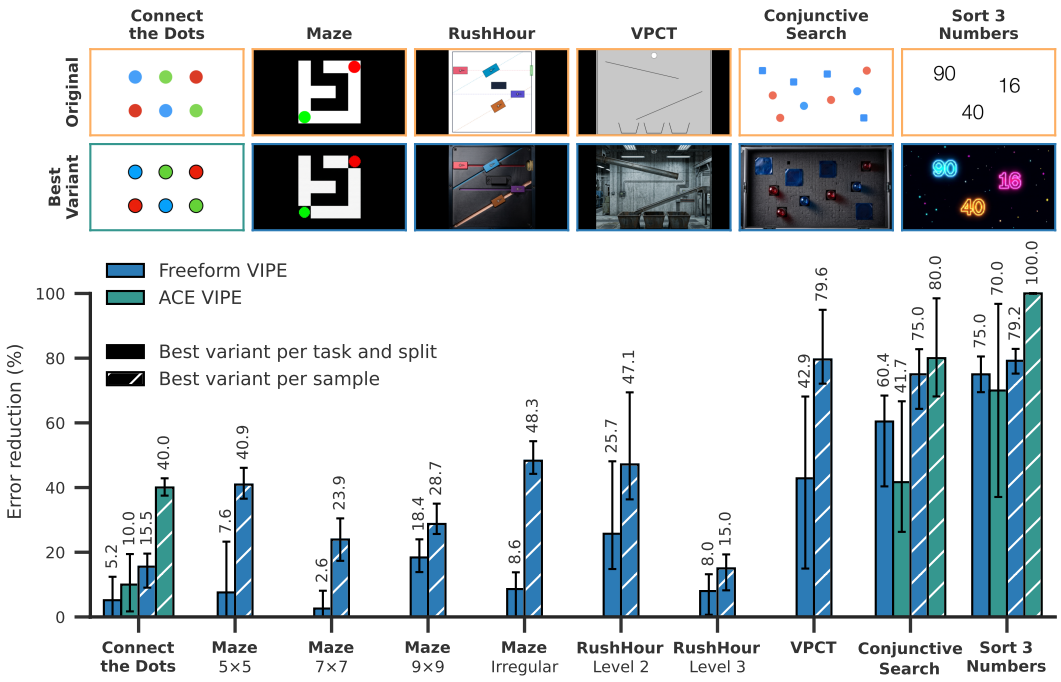


Figure 3: **Automated visual prompt engineering improves performance on a wide range of tasks**. Performance is plotted as error reduction (in percent) relative to the baseline of no visual prompt engineering. For example, if the baseline achieves 20% accuracy and VIPE achieves 60% accuracy, the error rate is halved (80% errors $\rightarrow$ 40% errors), resulting in a 50% reduction. Solid bars show performance for the overall best variant per task and split (e.g., *Maze 3×3*), out of $n = 20$ total variants. Successful variants range from complex 3D renders (e.g., *RushHour* is re-contextualized as an anodized metal puzzle box) to small appearance changes (e.g., the *Maze* is interpreted as an arcade game screen recording with subtle pixel texture). Hatched bars show performance when the best variant per task sample is selected by an oracle. See Sec. E for details and more results. Whiskers indicate 95% confidence intervals. Example videos for are on our project page.

stitutes a good prompt, and the visual prompt search space is vast. Here, we therefore test whether VIPE can be automated by comparing two approaches, freeform ideation by a VLM vs. step-by-step edits (ACE, cf. Kalibhat et al., 2026). In total, we evaluate 18,160 videos for this experiment; full details in Sec. E. We investigate this question on six visual reasoning tasks; an example of each is included in Fig. 3. In Fig. 3, we can see that both the freeform VLM automation approach and the step-by-step edits (only one concept is changed at a time) successfully find many alternative visual prompts that elicit better performance in Veo 3.1, sometimes reducing the error rate by up to 75% compared to the baseline of no visual prompt engineering.

Takeaway 2 Visual prompt engineering does not need to rely on human intuition: it can be successfully automated in different ways, from freeform ideation with a VLM to structured, step-by-step concept edits.

5 Can visual prompt engineering be used for test-time scaling?

Language models benefit from scaling during pre-training, post-training, and test-time. While video models are being scaled along the first two axes, test-time scaling has received limited attention. Notable exceptions are Li et al. (2026), characterizing the number of generated frames as a form of test-time compute; and Newman et al. (2026) who use a heuristic on multiple early-decoded videos to select the most promising attempt.

How does VIPE compare to test-time scaling? Fig. 4 shows Veo 3.1's base performance on *VPCT* improving with self-consistency (majority voting) (Wang et al., 2022) across up to 20 test-time samples: from 41.3% to 50.0% (+8.7 percentage points). In comparison, a single test-time sample on the engineered visual prompts from Sec. 3 already achieves 59.3% (+18 percentage points). While this indicates that VIPE is more effective than test-time scaling via self-consistency, we need not pick one or the other: self-consistency scales just as well on top of the engineered visual prompts as on the original prompts, yielding a compounded 68.0% (+27.7 pp) with 20 test-time samples.

However, the above experiment assumes that we already know an effective visual prompt variant. If that's not the case, can (automated) visual prompt engineering itself be an effective test-time scaling approach? Fig. 5 Left shows performance when majority-voting over multiple generated videos *and* VIPE variants for a fixed, given budget derived from API inference cost. Due to VIPE's effectiveness and low cost compared to video generation (even when accounting for m initial proposals and filtering, a VIPE variant currently costs only $1/8$ of a video, see Sec. B), performance is generally optimized by exploring as many VIPE variants as possible. This is also shown in Fig. 5 Right: it is more cost-effective to explore more VIPE variants than to generate more videos per variant.

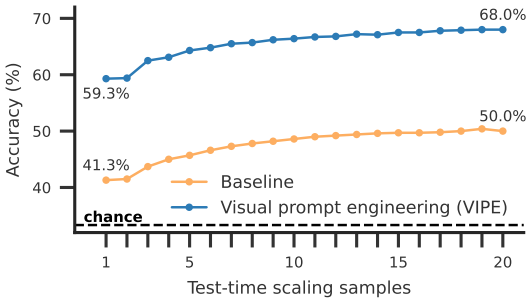


Figure 4: **Visual prompt engineering can be combined with test-time scaling to further improve performance.** For Veo 3.1 on *VPCT*, majority voting across 20 independent runs improves performance.

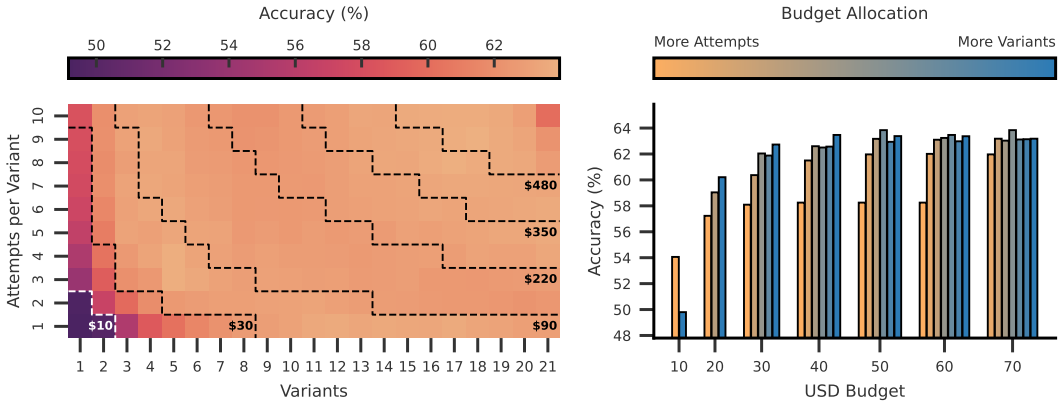


Figure 5: **VIPE variants are more cost-effective than additional attempts on the same prompt**. **Left**. Accuracy on *VPCT* with test-time scaling via self-consistency (majority voting) as a function of the number of VIPE variants (x axis) and the number of attempts (generated videos) per variant (= traditional test-time scaling, y axis). Budget isolines indicate total cost assuming USD 3.20 per video and USD 0.40 per VIPE with $m = 5$ edits (Google, 2026); see Sec. B for details. **Right**. For a given budget, it is generally better to allocate that budget to maximize the number of VIPE variants (right-most blue bars, corresponding to cells at $y = 1$ in the heatmap) as opposed to generating more attempts with the same visual prompt. Similar results for *Maze* and *RushHour* in Sec. F.

Takeaway 3 Visual prompt engineering is a more cost-effective test-time scaling strategy than simply generating more videos; both approaches can be combined for best results.

6 Is text-based or visual prompt engineering more effective for video models?

We showed that visual prompt engineering can effectively improve video model reasoning. However, video models receive both a visual prompt and a text prompt as input, and both are natural

levers for influencing model behavior. In the following, we directly compare how text prompt engineering vs. visual prompt engineering influence a video model's ability to solve the same task.

Text prompt engineering for video models We adapt the prompt engineering ideator from Sec. E.1 to propose alternative text prompts instead of image edits, keeping all source images from a baseline task fixed. Similar to the image edit ideator, we instruct the text ideator to propose text prompts that preserve the task structure. For example, if the original prompt is "The two blue balls begin to glow", the ideator may propose "The two blue balls shatter into tiny fragments" instead. We then compare how well these proposed alternative text prompts and alternative image prompts elicit better reasoning performance. The comparison is performed on four challenging datasets: *VPCT*, *Connect the Dots*, and *Conjunctive Search*, and *Sort 3 Numbers*. Besides *VPCT*, Veo 3.1 achieves a baseline accuracy of around 3–4% on the other tasks, making them suitable for measuring improvement headroom. See Sec. E.3 for full details.

Visual vs. text prompt engineering Results are shown in Fig. 6. Overall, we find that both text and visual prompt engineering substantially improve performance over the baseline across tasks, confirming that each modality is a viable lever for eliciting better video reasoning. For example, on *Sort 3 Numbers*, the best text variant and the best image variant each achieved roughly a 20-fold improvement (86% and 76%, vs. 4% without prompt engineering). On *Conjunctive Search* and *Sort 3 Numbers*, the image variants on average led to substantially larger improvements than text variants, while the single best text variant sometimes exceeds the single best image variant. On *VPCT*, we did not observe substantial improvement from the text and image variants proposed by the freeform ideators, with a particular image edit among $n = 20$ proposals as notable exception, raising Veo 3.1 accuracy from 57% to 73%. *Connect the Dots* remains a challenging task for Veo even with prompt engineering, with generally low performance under diverse image and text variant proposals. These results suggest that for most tasks, a well-chosen text or image prompt improve visual reasoning performance, though finding such a prompt requires non-trivial exploration. We also explored joint text and image prompt engineering (see Sec. E.5 for more details). Our simple joint text and image prompt engineering ideator generated coherent and interesting alternative ways to represent a task, but the resulting task variants were less effective in eliciting better video reasoning compared to changing the images alone. Taken together, visual prompt engineering reliably improves video reasoning, and is frequently more effective than traditional language-based prompt engineering.

Takeaway 4 Depending on the task, visual prompt engineering can be more powerful than text prompt engineering.

7 Why does visual prompt engineering help?

Throughout this article, we have seen evidence that the effectiveness of visual prompt engineering primarily stems from bridging the domain gap between abstract and realistic input. For example, the *VPCT* experiments from Sec. 3 showed that the best performance is achieved when sketches are transformed into input that has both 3D structure and realistic textures, while the "unnatural textures ablation" confirmed that adding 3D depth alone is not sufficient. The automated prompt engineering methods from Sec. 4 repeatedly surfaced creative ways to improve realism, from improving material properties to transforming abstract scenes into tangible 3D objects like a chalkboard. Video models clearly have a *realism bias*. When comparing video generations with and without VIPE, we noticed a recurring pattern: scene consistency appears to be drastically improved in realistic scenes, while objects tend to randomly appear, disappear, or change shape in abstract settings. This naturally impedes any downstream reasoning since it changes the task setup, just like changing a number at the beginning of a calculation would lead to the wrong result.

To understand better how scene consistency is influenced by VIPE, we design a simple experiment that visually prompt-engineers a synthetic dataset step-by-step in the direction of a fully realistic scene, shown in Fig. 7. This is based on the fact that realism is not necessarily a binary choice but rather a spectrum. Human ratings of scene consistency improved from 0% (!) in the synthetic setting to 59% in the fully realistic setting. Interestingly, every single step towards increased realism directly causes video model generations to become more consistent, thereby offering an explanation for why visual prompt engineering helps.

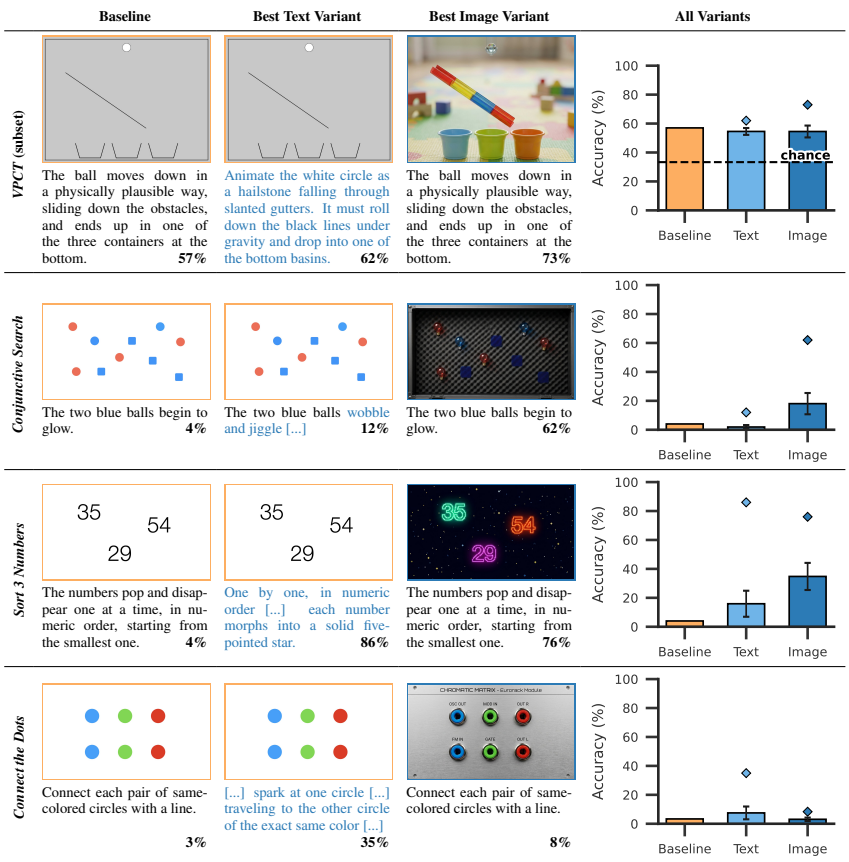


Figure 6: **Visual prompt engineering can be more effective than text prompt engineering for video models.** Each row represents a distinct task, showing the visual input and text prompt for the baseline, the best text variant, and the best image variant, with their respective accuracies in parentheses. The rightmost column summarizes the accuracy of all text and image proposals. The diamond markers show the best variant among $n = 20$. Error bars indicate 95% CI. See Sec. E.3 for more example text and image variants.

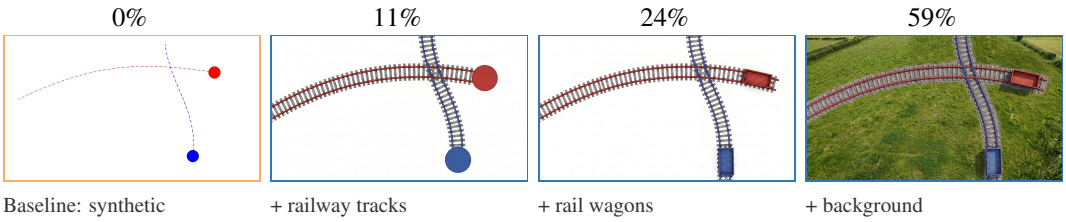


Figure 7: **Each step towards realism improves generation consistency.** A simple dataset of red/blue objects and paths is scored on scene consistency (100 videos per condition). Prompt: *"The red and blue {circles, rail wagons} move along the {dashed lines, railway tracks} until they reach the intersection point. Static shot, no zoom no pan no dolly. NO CAMERA MOVEMENT AT ALL."* Starting from the baseline (synthetic) condition, three VIPE variants systematically introduce more realism. When videos generated by Veo 3.1 are rated as a pass/fail on scene consistency by humans (success if objects/lines/tracks don't randomly appear or disappear, change shape, morph etc.), it becomes clear that increased realism causes video model generations to become more consistent.

8 Discussion

Summary. Inspired by the fact that text-based prompt engineering has become a foundational technique for improving language model performance, we set out to understand whether video model reasoning can similarly be improved by *visual* prompt engineering. We find that good "visual prompts" indeed systematically improve video reasoning across tasks. In fact, VIPE works even better than traditional test-time scaling methods like self-consistency (for a given inference budget), and can also be more effective than optimizing the text prompt alone. We identified patterns that reliably improve performance, such as the *realism bias* of video models: video models prefer to reason in photorealistic scenes, as opposed to impoverished, abstract or synthetic ones. Taken together, VIPE is a cheap and effective lever for improving visual reasoning with video models.

Realism bias: stop evaluating video models on abstract, impoverished tasks. Video model accuracy improves when transforming abstract settings into realistic-looking scenes. As a consequence, reasoning performance on datasets that look abstract / synthetic (which is the case for most of them) is often only a lower bound on the model's underlying reasoning ability. This distinction has been described in the literature as the "performance vs. competence" dilemma (Firestone, 2020): e.g., in order to measure the true maximum speed of a fish, it's worth testing the fish in the water as opposed to on land. Similarly, our results indicate that video models often have the *competence* (or ability) to solve a task better than what is indicated by their poor *performance* on an abstract, impoverished version of the task. Consequently, we recommend to stop evaluating video models on abstract tasks when there's the option to visually prompt-engineer a more realistic version of the same task.

Takeaway 5 Whenever feasible, don't evaluate video models on abstract/impoverished tasks. Video models have a realism bias, thus unrealistic-looking tasks underestimate their true performance. Instead, use visual prompt engineering to transform abstract tasks into realistic setups.

On the other hand, the fact that video models have a realism bias is clearly suboptimal. It would be preferable for a video model to solve any task specified in any way, shape or form. To a certain degree, this feels reminiscent of the brittleness of early language models, where a lot of care went into formulating tasks in a way that elicited the best possible performance (e.g. Jiang et al., 2020; Bouraoui et al., 2020; Reynolds & McDonell, 2021). For example, to borrow an example from Jiang et al. (2020), the text prompts "Obama worked as a ___" vs. "Obama is a ___ by profession" can systematically influence language model performance, despite being variants of the exact same underlying task. In this sense, realism bias might be characterized as an unwanted shortcut (Geirhos et al., 2020; Lapuschkin et al., 2019) or dataset bias (Torralba & Efros, 2011) that models pick up from their training data. It would be interesting to understand whether scene consistency is highest for realistic scenes because unrealistic/abstract scenes take the model out-of-distribution, or whether it simply takes them to an in-distribution setting with different priors (e.g., cartoon movies).

Limitations. The biggest limitation is that the quality of a prompt engineering variant is highly dependent on the quality of the image editing model. If the image editor changes the nature of the task or introduces artifacts, then it can become impossible for the downstream video model to solve the original task. We mitigated this risk through a filtering / quality control step where e.g. a VLM reviews (and possibly rejects) image edits. In the future, with further improvements in image editing models, this step may not be necessary anymore. Another limitation is the potential cost overhead of visual prompt engineering. However, since the cost of image editing pales in comparison to the cost of video generation, in practice this often turns out to be an advantage of VIPE, since a (comparatively cheap) image editing call can significantly reduce the (comparatively expensive) chance of a video generation that fails to solve the task.

Outlook. The success of visual prompt engineering points toward a shifting paradigm where visual instructions are treated as an optimizable space rather than fixed inputs, just like text prompts are carefully crafted to elicit better LLM performance. As frontier video models evolve towards accepting richer multi-modal context, VIPE might extend beyond editing a single image—perhaps involving multiple reference images or videos. As models continue to improve, an interesting question arises: Will we need less and less visual prompt engineering, since video models might understand even abstract tasks more easily? Or, alternatively, will VIPE become more powerful since editing becomes easier? Only time will tell, but if there's anything to be learned from the history of LLMs, it's the lesson that a model can only ever be as good as its prompt.

AI disclosure section. In this work, we used generative AI tools as follows: 1.) to generate and modify datasets as described in the article (followed by human review); 2.) for coding (followed by human review); 3.) to brainstorm ideas for tasks; 4.) as part of the automated VIPE pipeline described in detail in Sec. E; 5.) to provide feedback and suggestions on the article draft which was written by the authors themselves. The authors have reviewed all AI-assisted work and take full responsibility for the content of this work.

REFERENCES

Pablo Acuaviva, Aram Davtyan, Mariam Hassan, Sebastian Stapf, Ahmad Rahimi, Alexandre Alahi, and Paolo Favaro. Rethinking visual intelligence: Insights from video pretraining. *arXiv preprint arXiv:2510.24448*, 2025.

Hyojin Bahng, Ali Jahber, Prithvijit Chakrabarty, and Phillip Isola. Exploring visual prompts for adapting large-scale models. *arXiv preprint arXiv:2203.17274*, 2022.

Amir Bar, Yossi Gandelsman, Trevor Darrell, Amir Globerson, and Alexei A Efros. Visual prompting via image inpainting. In *Advances in Neural Information Processing Systems*, volume 35, pp. 25005–25017, 2022.

Zied Bouraoui, Jose Camacho-Collados, and Steven Schockaert. Inducing relational knowledge from BERT. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7456–7463, 2020.

camelCase12. Visual Physics Comprehension Test (VPCT) Dataset. https://huggingface.co/datasets/camelCase12/vpct-1, 2025.

Junhao Cheng, Liang Hou, Tianxiong Zhong, Xin Tao, Pengfei Wan, Kun Gai, and Jing Liao. VLMs are good teachers for video reasoning via adaptive test-time optimization. *arXiv preprint arXiv:2606.02564*, 2026.

Chaz Firestone. Performance vs. competence in human–machine comparisons. *Proceedings of the National Academy of Sciences*, 117(43):26562–26571, 2020.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

Louie Giray. Prompt engineering with ChatGPT: a guide for academic writers. *Annals of biomedical engineering*, 51(12):2629–2633, 2023.

Google. Gemini 3.1 Pro. https://deepmind.google/models/gemini/pro/, 2026a. Accessed: June 17, 2026.

Google. Nano Banana 2: Gemini Image Generation Overview. https://gemini.google/overview/image-generation/, 2026b. Accessed: June 17, 2026.

Google. Omni Flash Model Card. https://deepmind.google/models/model-cards/gemini-omni-flash/, 2026c. Accessed: July 1, 2026.

Google. Veo 3.1. https://deepmind.google/models/veo/, 2026d. Accessed: June 17, 2026.

Google. Gemini developer API pricing. https://ai.google.dev/gemini-api/docs/pricing#veo-3.1, 2026. URL https://ai.google.dev/gemini-api/docs/pricing#veo-3.1. Accessed: 2026-07-01.

Jindong Gu, Zhen Han, Shuo Chen, Ahmad Beirami, Bailan He, Gengyuan Zhang, Ruotong Liao, Yao Qin, Volker Tresp, and Philip Torr. A systematic survey of prompt engineering on vision-language foundation models. *arXiv preprint arXiv:2307.12980*, 2023.

Ziyu Guo, Xinyan Chen, Renrui Zhang, Ruichuan An, Yu Qi, Dongzhi Jiang, Xiangtai Li, Manyuan Zhang, Hongsheng Li, and Pheng-Ann Heng. Are video models ready as zero-shot reasoners? an empirical study with the mme-cof benchmark. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9175–9184, 2026.

Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *European Conference on Computer Vision (ECCV)*, pp. 709–727. Springer, 2022.

Zhengbao Jiang, Frank F Xu, Jun Araki, and Graham Neubig. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438, 2020.

Neha Kalibhat, Zi Wang, Prasoon Bajpai, Drew Proud, Wenjun Zeng, Been Kim, and Mani Malek. Interpreting and controlling model behavior via constitutions for atomic concept edits. In *Annual Conference on Artificial Intelligence and Statistics (AISTATS)*, 2026.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive NLP. *arXiv preprint arXiv:2212.14024*, 2022.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, et al. Dspy: compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*, volume 2024, pp. 54928–54958, 2024.

Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Unmasking Clever Hans predictors and assessing what machines really learn. *Nature communications*, 10(1):1096, 2019.

Chengzu Li, Zanyi Wang, Jiaang Li, Yi Xu, Han Zhou, Huanyu Zhang, Ruichuan An, Dengyang Jiang, Zhaochong An, Ivan Vulić, et al. Thinking in frames: How visual context and test-time scaling empower video reasoning. *arXiv preprint arXiv:2601.21037*, 2026.

Zhong-Yu Li, Ruoyi Du, Juncheng Yan, Le Zhuo, Zhen Li, Peng Gao, Zhanyu Ma, and Ming-Ming Cheng. Visualcloze: A universal image generation framework via visual in-context learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 18969–18979, 2025.

Weifeng Lin, Xinyu Wei, Ruichuan An, Gao Peng, Bocheng Zou, Yulin Luo, Siyuan Huang, Shanghang Zhang, and Hongsheng Li. Draw-and-understand: Leveraging visual prompts to enable mllms to comprehend what you want. In *International Conference on Learning Representations*, volume 2025, pp. 46374–46403, 2025.

Ggaliwango Marvin, Nakayiza Hellen, Daudi Jjingo, and Joyce Nakatumba-Nabende. Prompt engineering in large language models. In *International conference on data intelligence and cognitive informatics*, pp. 387–402. Springer, 2023.

Bertalan Meskó. Prompt engineering as an important emerging skill for medical professionals: tutorial. *Journal of medical Internet research*, 25:e50638, 2023.

Kaleb Newman, Tyler Zhu, and Olga Russakovsky. Video models reason early: Exploiting plan commitment for maze solving. *arXiv preprint arXiv:2603.30043*, 2026.

Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended abstracts of the 2021 CHI conference on human factors in computing systems*, pp. 1–7, 2021.

Aleksandar Shtedritski, Christian Rupprecht, and Andrea Vedaldi. What does clip know about a red circle? visual prompt engineering for vlms. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11987–11997, 2023.

Jingqi Tong, Yurong Mou, Hangcheng Li, Mingzhe Li, Yongzhuo Yang, Ming Zhang, Qiguang Chen, Tianyi Liang, Xiaomeng Hu, Yining Zheng, et al. Thinking with video: Video generation as a promising multimodal reasoning paradigm. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 41121–41129, 2026.

Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *CVPR 2011*, pp. 1521–1528. IEEE, 2011.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, Jiayu Wang, Jingfeng Zhang, Jingren Zhou, Jinkai Wang, Jixuan Chen, Kai Zhu, Kang Zhao, Keyu Yan, Lianghua Huang, Mengyang Feng, Ningyi Zhang, Pandeng Li, Pingyu Wu, Ruihang Chu, Ruili Feng, Shiwei Zhang, Siyang Sun, Tao Fang, Tianxing Wang, Tianyi Gui, Tingyu Weng, Tong Shen, Wei Lin, Wei Wang, Wei Wang, Wenmeng Zhou, Wente Wang, Wenting Shen, Wenyuan Yu, Xianzhong Shi, Xiaoming Huang, Xin Xu, Yan Kou, Yangyu Lv, Yifei Li, Yijing Liu, Yiming Wang, Yingya Zhang, Yitong Huang, Yong Li, You Wu, Yu Liu, Yulin Pan, Yun Zheng, Yuntao Hong, Yupeng Shi, Yutong Feng, Zeyinzi Jiang, Zhen Han, Zhi-Fan Wu, and Ziyu Liu. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Jiaqi Wang, Zhengliang Liu, Lin Zhao, Zihao Wu, Chong Ma, Sigang Yu, Haixing Dai, Qiushi Yang, Yiheng Liu, Songyao Zhang, et al. Review of large vision models and visual prompt engineering. *Meta-Radiology*, 1(3):100047, 2023a.

Maijunxian Wang, Ruisi Wang, Juyi Lin, Ran Ji, Thaddäus Wiedemer, Qingying Gao, Dezhi Luo, Yaoyao Qian, Lianyu Huang, Zelong Hong, et al. A very big video reasoning suite. *arXiv preprint arXiv:2602.20159*, 2026a.

Ruisi Wang, Zhongang Cai, Fanyi Pu, Junxiang Xu, Wanqi Yin, Maijunxian Wang, Ran Ji, Chenyang Gu, Bo Li, Ziqi Huang, et al. Demystifying video reasoning. *arXiv preprint arXiv:2603.16870*, 2026b.

Xinlong Wang, Wen Wang, Yue Cao, Chunhua Shen, and Tiejun Huang. Images speak in images: A generalist painter for in-context visual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6830–6839, 2023b.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. A prompt pattern catalog to enhance prompt engineering with ChatGPT. *arXiv preprint arXiv:2302.11382*, 2023.

Thaddäus Wiedemer, Yuxuan Li, Paul Vicol, Shixiang Shane Gu, Nick Matarese, Kevin Swersky, Been Kim, Priyank Jaini, and Robert Geirhos. Video models are zero-shot learners and reasoners. *arXiv preprint arXiv:2509.20328*, 2025.

Sherry Yang, Jacob Walker, Jack Parker-Holder, Yilun Du, Jake Bruce, Andre Barreto, Pieter Abbeel, and Dale Schuurmans. Video as the new language for real-world decision making. *arXiv preprint arXiv:2402.17139*, 2024.

Yuan Yao, Ao Zhang, Zhengyan Zhang, Zhiyuan Liu, Tat-Seng Chua, and Maosong Sun. Cpt: Colorful prompt tuning for pre-trained vision-language models. *AI Open*, 5:30–38, 2024.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic differentiation via text. *arXiv preprint arXiv:2406.07496*, 2024.

Jana Zeller, Thaddäus Wiedemer, Fanfei Li, Thomas Klein, Prasanna Mayilvahanan, Matthias Bethge, Felix Wichmann, Ryan Cotterell, and Wieland Brendel. MENTISOCULI: Revealing the limits of reasoning with mental imagery. *arXiv preprint arXiv:2602.02465*, 2026.

Jana Ricarda Zeller, Aleksandar Shtedritski, and Christian Rupprecht. Highlight: Learning visual prompts for vision-language models, 2024.

APPENDIX

A DATASETS

VPCT MIT license, dataset on HuggingFace by camelCase12 (2025), 100 samples. For the results in Sec. 4, we use the first ten samples. For Sec. 3, the entire dataset is used.

Conjunctive Search, Sort 3 Numbers, Connect the Dots Task ideas were drawn with permission from the authors of Wiedemer et al. (2025). Datasets were generated manually in Keynote by the authors. *Conjunctive Search* contains 5 image samples, each showing 2 blue circles (targets), 4 red circles (distractors) and 4 blue squares (distractors) at different positions. *Sort 3 Numbers* contains 5 image samples of different sets of double-digit numbers. *Connect the Dots* contains 6 image samples of 3 same-colored circles with varying colors.

Maze With permission from the authors, we use the four maze solving splits from Wiedemer et al. (2025). The first three splits with 50 samples each are mazes on square grids with size 5×5, 7×7, and 9×9. The last splits contains 40 non-axis-aligned mazes with curved paths. We report results on the first ten samples of each split in Sec. 4.

RushHour One task from the MentisOculi benchmark (Zeller et al., 2026), published under an Apache 2.0 license. We select this task as it is the only one used to test video models in the original paper. The task is organized in five splits (difficulty levels) with 50 samples each, where the level corresponds to the number of cars that have to be moved for a minimum-length solution. Level 1 is easily solved by Veo 3.1, while Levels 4 and 5 prove too difficult, largely because the model fails to maintain consistency in complex scenes (Zeller et al., 2026). We consequently focus our analysis on Levels 2 and 3, reporting results on the first ten samples of either in Sec. 4.

B INFERENCE DETAILS

We run inference with Gemini, Veo and Omni models via the Vertex AI API. Images and videos are generated at 720p. Videos are generated with a duration of 8s. At the time of writing, generation costs USD 0.40 per second for videos with Veo 3.1 (with or without audio), for a total of USD 3.20 per video. Image editing costs around USD 0.005 for a 720p input image and USD 0.067 for a 720p output image for Gemini 3.1 Flash Image (Nano Banana 2). A complete VIPE loop with five ideated edits and filtering costs around USD 0.40, dominated by the image editing cost (Google, 2026).

C VPCT: EVALUATION DETAILS

MSE-based evaluator (for original sketches) To determine in which of three containers (left, center, right) the ball lands, we partition the bottom portion of each frame into three regions of interest and measure the mean squared error (MSE) between the first frame and each subsequent frame independently within each region. A sliding window of the last five MSE values is maintained per region. After an initial warmup period of 12 frames to initialize the sliding window average, the evaluator identifies the first frame on which exactly one region's MSE exceeds its running mean by at least a factor of two, and returns that region as the predicted landing container.

Color-based evaluator (for visual prompt engineering variants) Since visual prompt engineering introduces more variation and would thereby reduce the robustness of an MSE-based evaluation, we here instead track the location of the ball throughout the video. Each frame is converted to HSV space and a threshold isolates pixels matching the ball's color (red). Contours are extracted from the resulting binary mask and filtered by area and circularity to reject background noise. The centroid of the largest valid contour is computed and mapped to one of the three container regions based on its horizontal position. The evaluator returns the first frame on which the ball is detected inside a container region. If the ball is detected on the final frame but falls outside the container regions, the nearest container is assigned as the model's guess. In order to make sure that a VLM rewriter doesn't simply spell out the solution bucket (e.g., '... and finally, the ball drops into the middle

bucket') and instead, the video model is required to simulate the ball's trajectory, we preprocessed VIPE VPCT images such that the buckets are removed / replaced with the background color. That way, the model still needs to simulate the trajectory, but the ball doesn't end up in a bucket that can be pre-specified. Since this setup makes the task a bit harder to understand for humans, we retained buckets for visualization purposes in figures; keeping in mind that the core task is still exactly the same and the ball's location is scored against the target regions where the buckets would have been.

Text prompts Here are prompt details for *VPCT*.

Original prompt (English)

```
[image]
```

The ball moves down in a physically plausible way, sliding down the obstacles, and ends up in one of the three containers at the bottom. Static shot, no zoom no pan no dolly.

Original prompt (Chinese used for the corresponding Wan2.2 experiments)

```
[image]
```

球以符合物理规律的方式向下移动，滑下障碍物，并最终落在底部的三个容器之一中。静态镜头，无缩放无平移无推拉。

VIPE and unnatural textures prompt (English)

```
[image]
```

The single red ball moves down in a physically plausible way, sliding down the obstacles (always down, never up), and after passing the obstacles, it finally drops to the ground. Apart from the red ball, the scene and wooden boards remain static and unchanged. Static shot, no zoom no pan no dolly. NO CAMERA MOVEMENT AT ALL.

VIPE and unnatural textures prompt (Chinese used for the corresponding Wan2.2 experiments)

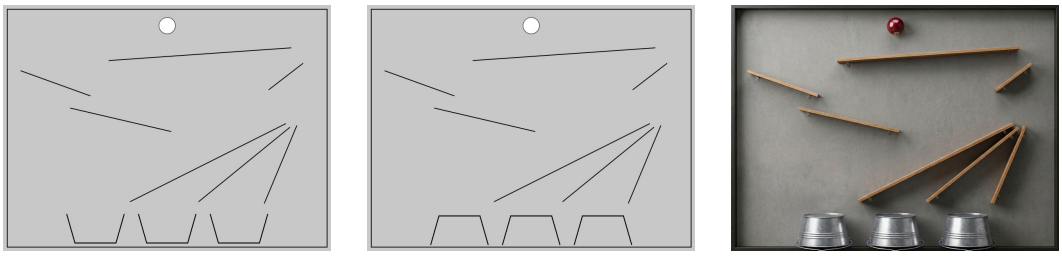
```
[image]
```

单个红球以符合物理规律的方式向下移动，滑下障碍物（总是向下，从不向上），在穿过障碍物后，它最终掉到地面上。除了红球之外，场景和木板保持静态不变。静态镜头，无缩放无平移无推拉。完全没有相机移动。

Test-time scaling details To assess how model accuracy on the physics *VPCT* benchmark scales by test-time scaling the number of independent runs, we computed majority-vote accuracy for $k = 1, 2, \ldots, 20$ runs using a leave-out subsampling procedure. The benchmark consists of 100 samples, each assigned to one of three classes (chance level: 33.3%). For each value of k, we enumerated all $\binom{20}{k}$ possible subsets of k runs drawn without replacement from the 20 available; where this number exceeded 5,000, we instead drew 5,000 random subsets. For each subset, we then computed a majority vote across the k run predictions per item, resolving ties by sampling uniformly at random among the tied classes. The reported accuracy for each k is the mean over all evaluated subsets.

Have VLMs solved physics reasoning on VPCT? Over the course of one year, vision-language models (VLMs) have shown rapid progress on the *VPCT* dataset, from below 50% accuracy in April 2025 to now 91% accuracy by the leading model on the public leaderboard, Gemini 3 Pro Preview. An updated version, Gemini 3.1 Pro, even scores 96% on this dataset in our own experiments. Have VLMs, therefore, essentially solved physics reasoning on this type of data?

We decided to put their physics reasoning abilities to a test by creating a modified version *VPCT*, visualized in Fig. 8. The result is striking: when buckets are turned upside-down, model accuracy drops down to a mere 3–4% (see Tab. 1 for accuracies). The model completely ignores that a ball cannot drop *into* a bucket that is upside-down. This provides strong evidence that VLMs only have a superficial, not a robust understanding of physics. Vision-language models, on this task, appear to use a shortcut (Lapuschkin et al., 2019; Geirhos et al., 2020) and lack an understanding of the underlying physical principles.



Original:
"... this vertical drop leads directly into the first bucket on the left"

Upside-down buckets:
"... this final slope guides the ball directly into the left bucket"

Realistic upside-down buckets:
"... the ball will land in the leftmost bucket"

Figure 8: **Control conditions reveal shallow physics understanding of vision-language models.** Prompt: *"Predict which bucket (numbered 1, 2, or 3 from left to right) the ball will eventually fall into. If the ball will not fall into any bucket, output 'answer(0)'; if the ball falls into a bucket, output 'answer(X)' where X is the bucket number (1, 2, or 3)."* **Original (a)**: VLMs appear to understand the setup and correctly predict the bucket. **Control condition upside-down buckets (b):** The model still believes that the ball will drop **into** a bucket, revealing a shallow understanding. **Control condition with realistic upside-down buckets (c)**: In order to rule out that the VLM has, in fact, a good understanding of physics and just misunderstands the abstract sketch setup of condition (b), a realistic control condition confirms that the model's understanding is indeed shallow: it still does not get the answer right.

In order to rule out that the model simply isn't able to perceive whether the bucket is upside-down or not (or somehow misunderstands the sketch), we separately asked the model to identify whether the buckets are upside down in any given sample with the following prompt: *"Are the buckets upside down - True or False? Output 'answer(X)' where X is either 'True' (= buckets are upside down) or 'False' (= buckets are not upside down)."* Gemini 3.1 Pro accuracy on this task is 100%. The model correctly identifies all upside-down buckets as being upside-down, and also identifies all original buckets as not being upside-down. The model, therefore, has full access to the visual information required to solve the task—but it fails to put the pieces together. In summary, despite reaching deceptively high accuracies on the original *VPCT*, VLMs are far from solving physics reasoning.

Table 1: Accuracy of vision-language models on *VPCT* control conditions. Values in brackets indicate whether the model-selected bucket (even though it's upside-down) would have been the correct bucket in the original task. The binary classification task asks the model to identify whether the buckets are upside-down or not in a balanced dataset (chance: 50%).

	Condition		
Model	**Original Sketch**	**Upside-Down Sketch**	**Upside-down Classification**
Gemini 3.1 Pro	96%	4% (87%)	100%
Gemini 2.5 Pro	48%	3% (42%)	82%

D WAN

D.1 WAN REWRITER

The Wan model has the option enable/disable the prompt rewriter (Qwen2.5-Plus). To understand whether this helps or hurts performance, we performed an ablation of the Wan rewriter regarding *VPCT* performance. With rewriter enabled, *VPCT* with visual prompt engineering performance was 37.7% for the TI2V model, which is lower than the comparison conditions 42.3% (TI2V English) or 43.3 (TI2V Chinese) from Fig. 2. Thus, for this task, the model performs better without rewritten prompts. For this reason, numbers in the main paper are reported without rewriter enabled.

E AUTOMATED VIPE DETAILS

Tasks We investigate how to automate visual prompt engineering on six visual reasoning tasks; an example of each is included in Fig. 3. *VPCT* is introduced in Sec. 3. *Maze* includes four different kinds of mazes from Wiedemer et al. (2025) with different complexity. We also use three other tasks from the same article: asking the model to *Connect the Dots* with equal color, *Sort 3 Numbers* by manipulating them in sequence, and performing *Conjunctive Search* by identifying target objects matching specific criteria in a cluttered scene. Finally, we test *RushHour* (Zeller et al., 2026), which asks the model to rearrange cars to free a path out of a congested parking lot. In total, we evaluate 18,160 videos for this experiment; details in Sec. E. In order to score whether a given video successfully solves the task, we employ either VLM-based or deterministic autoraters. Details can be found in Secs. E and I.

E.1 FREEFORM IDEATION BY A VLM

We start with a naive approach: generating prompt engineering ideas t_{edit} with a VLM ideator. The VLM's general ideation prompt t_{ideate} seeks to transfer the task to a new visual domain while preserving spatial layout and object correspondence. In practice, we found the resulting image edits to be faithful to this instruction, only changing the visual appearance of the images without altering the task. Here, we instantiate the VIPE ideator with Gemini 3.1 Pro (Google, 2026a), the image editor as Nano Banana 2 (Gemini 3.1 Flash Image), and the filtering model again as Gemini 3.1 Pro (selecting the best out of 5 attempts). The ideator does not receive feedback from downstream evaluation, i.e., operates open-loop. Full prompts and details are provided in Sec. E.3. The results are shown in Fig. 3. Across a wide range of tasks, we show that this simple freeform, open-loop visual prompt engineering approach successfully finds many alternative visual prompts that elicit better performance in Veo 3.1. Some task variants allow Veo 3.1 to achieve surprisingly large performance gains (reducing the error rate by over 75% compared to the baseline of no visual prompt engineering), simply by altering the visual context in which the model's reasoning occurs.

E.2 STEP-BY-STEP EDITS: ATOMIC CONCEPT EDITING (ACE)

An alternative, more structured approach for visual prompt engineering is editing the original image step-by-step, one concept at a time (e.g., original image → change background to blue → increase object size → ...). To this end, we adapt the ACE framework (Kalibhat et al., 2026) to the visual prompt engineering setting. An atomic concept edit (ACE) is a simple edit that either adds, removes or replaces a single concept at a time. Prompt exploration is performed in a tree-like fashion: Starting from the root node (original image), ACEs are generated by Gemini 3.1 Pro to verbally describe the possible edits to the current image. The edits are then applied to the image, using the image editor Nano Banana 2, to obtain a number of child-node images. For each of these child nodes, exploration again branches out in different directions. The ACE exploration stops when a pre-determined number of successful nodes are generated. In contrast to freeform ideation, ACE operates in a closed loop that iteratively adapts the edits according to autorater feedback. More details can be found in Sec. E.4.

Since ACE is expensive, we applied it only to three best/worst performing tasks: *Connect the Dots*, *Conjunctive Search* and *Sort 3 Numbers*. Fig. 3 shows the results for ACE in shades of green. Overall, we observe systematic improvements over the "no visual prompt engineering" baseline, for a similar final performance to freeform VIPE. ACE's more structured edits yield more optimized per-sample variants, however, which lead to a 100% reduction rate on *Sort 3 Numbers*.

Our analysis of the prompt mutations generated by ACE revealed several task-agnostic edit patterns that consistently improved model performance. One major pattern involved increasing realism of material properties to improve temporal object permanence, e.g., transforming abstract elements into tangible 3D objects using descriptors like "matte clay" or "textured chalkboard". Another major strategy relied on high-contrast target isolation to guide the model's attention, e.g., utilizing solid dark backgrounds or applying thick borders. A final distinct trend showed mutations moving toward semantic simplification, e.g., replacing formal vocabulary and abstract modifiers with direct action verbs such as replacing "same-colored" with "matching" or "the two" with "both". Furthermore, we

Table 2: Breakdown of the **18 160 generated videos** for Sec. 4 and Fig. 3.

VIPE	Task (Split)	Task Samples	Variants	Videos per Sample	Videos Total
Freeform	*RushHour* (Level 2)	10	20	10	2000
	RushHour (Level 3)	10	20	10	2000
	Maze (5×5)	10	20	10	2000
	Maze (7×7)	10	20	10	2000
	Maze (9×9)	10	20	10	2000
	Maze (irregular)	10	20	10	2000
	VPCT	10	20	10	2000
	Conjunctive Search	5	20	10	1000
	Connect the Dots	6	20	10	1200
	Sort 3 Numbers	5	20	10	1000
ACE	*Conjunctive Search*	5	20	3	300
	Connect the Dots	6	20	3	360
	Sort 3 Numbers	5	20	3	300

observed that these strategies are rarely applied in isolation; rather, successful mutations frequently combine multiple patterns. An in-depth analysis is presented in Sec. E.4.

Tab. 2 summarizes how many variants and final videos were generated for each task in Sec. 4 and Fig. 3. Each bar in Fig. 3 summarizes only the best variant: the overall best for an entire task and split (e.g., *Connect the Dots*, *Maze* 3×3, *RushHour* Level 2) as solid bars, the best per sample-variant as hatched bars.

E.3 Freeform ideation

We iterate the entire automatic visual prompter n times in sequence, with each iteration's final edits instructions t_{edit} being appended to the next iteration's ideation prompt t_{ideate} to encourage diversity.

We find this iterated approach to be generally effective in avoiding similar visual settings. However, we notice on the *Maze* task (Fig. 14) that a very detailed base task description that specifies the appearance of many parts of the scene (see below) can constrain the editor's ability to realize proposed visual edits, resulting in only subtle visual changes. Nevertheless, even subtle changes can boost performance, as demonstrated by the "pixel arcade game" maze variant shown in Fig. 3 which merely modifies the scene to have a low-contrast screen pixel texture.

We also observe that freeform ideation struggles to find a single variant that is effective across task splits (e.g., *Maze* sizes or *RushHour* levels).

Finally, we show results for freeform ideation with Wan2.2 TI2V (see Sec. D) in Fig. 9, demonstrating its effectiveness beyond Veo 3.1.

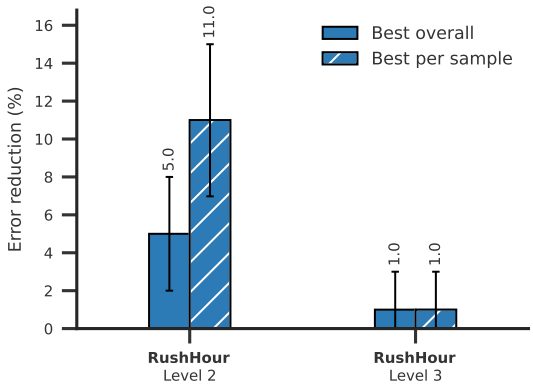


Figure 9: Freeform VIPE results on *RushHour* for Wan2.2 TI2V.

Language vs. visual prompt engineering details We base this comparison on the *Conjunctive Search*, *Sort 3 Numbers* (*Sort 3 Numbers* in numerical order), and *Connect the Dots* tasks, as well as the first 10 images from *VPCT*. For each task, we collect n=20 sequential proposals from the text prompt engineering ideator and n=20 sequential proposals from the image prompt engineering ideator. We used Gemini 3.1 Pro for both ideators. All n proposals are evaluated on the full image set with 10 video attempts per image. Each video is scored against

binary pass (0 or 1) by Gemini 3.1 Pro as a VLM autorater.

PROMPTS

We provide the full prompts used by the freeform prompt engineering ideator described in Secs. 4 and 6 below.

Image ideator prompt

SYSTEM INSTRUCTION

You are an expert AI researcher designing image variants for a visual reasoning benchmark. The video event is fixed—your goal is to explore diverse alternative visual scenes where a video generation model might perform better, while preserving the core reasoning challenge.

Each proposal should be distinct from previous ones. Aim for diversity in visual domains and artistic styles.

USER CONTENT

Based on the base task and the sample images provided, propose a new visual domain for the task images. The text instruction will NOT be changed.

The variant should:

- Preserve spatial structure, element count, and element correspondence
- Present the task in a visual style where a video generation model might have better priors
- Be compatible with the fixed text instruction

Output your proposal as a JSON block:

```
{"variant_id": "short_snake_case_id",
 "variant_description": "Brief description of the visual
  domain change",
 "image_edit_prompt": "Detailed prompt for an image editing
  model to transform the original task images into this
  variant's visual style. Be specific about what each visual
  element should become.",
 "autorater_rubric": "Criteria for evaluating the generated
  video. Describe desired behavior (what correct/partial/
  incorrect outcomes look like) AND undesired behavior to
  penalize (e.g., visual artifacts, glitches, unrealistic
  motion)."}
```

Fixed video instruction (do NOT change): `[base task]`
Sample images: `[sample images]`
Prior proposals: `[descriptions of all prior proposals]`

Image filter prompt

SYSTEM INSTRUCTION

I will provide an original abstract image, an edit instruction, and multiple attempts at editing the image. Please evaluate which attempt best follows the edit instruction while preserving the ESSENTIAL STRUCTURE of the original image. Essential structure includes:

- Spatial layout: positions and arrangements of elements
- Element count: the same number of distinct elements
- Element correspondence: each element in the original maps to exactly one element in the edit

Edit instruction: `{edit_prompt}` After reviewing all images, explain your reasoning, then answer with exactly this statement: 'For this reason, the {first / second / third / fourth / ...} attempt is the most faithful edit.'

USER CONTENT

Original image: `[original image]`
Attempt 1: `[edited image 1]`
Attempt 2: `[edited image 2]`
...

Text ideator prompt

SYSTEM INSTRUCTION

You are an expert AI researcher optimizing text prompts for a visual reasoning benchmark. The visual scene is fixed—your goal is to explore diverse alternative video events (what happens to express the answer) where a video generation model might perform better, while preserving the core reasoning challenge.

Each proposal should be distinct from previous ones. Aim for diversity in how the video event is described and framed.

USER CONTENT

Based on the base task and the sample images provided, propose a new text instruction variant for video generation. The images will NOT be changed.

The variant should:

- Describe what should happen in the video without leaking the solution
- Preserve the core reasoning challenge
- Consider alternative video events the model may handle better

Output your proposal as a JSON block:

```
{"variant_id": "short_snake_case_id",
 "variant_description": "Brief description of this instruction
  variant and how it differs from the base",
 "instruction_variant": "Text prompt for the video generation
  model. Describe what event should unfold over time without
  leaking the solution to the task.",
 "autorater_rubric": "Criteria for evaluating the generated
  video. Describe desired behavior (what correct/partial/
  incorrect outcomes look like) AND undesired behavior to
  penalize (e.g., visual artifacts, glitches, unrealistic
  motion)."}
```

Base task description: `[base task]`
Sample images: `[sample images]`
Prior proposals: `[descriptions of all prior proposals]`

BASE TASK DESCRIPTIONS

The base task descriptions (corresponding to the original task text prompts) for *VPCT*, *Conjunctive Search*, *Sort 3 Numbers*, and *Connect the Dots* are detailed in Tabs. 3 to 6. Task descriptions for *Maze* and *RushHour* are detailed below.

Maze task description

Create a 2D animation based on the provided image of a maze. The red circle slides smoothly along the white path, stopping perfectly on the green circle. The red circle never slides or crosses into the black areas of the maze. The camera is a static, top-down view showing the entire maze.

Maze:

- The maze paths are white, the walls are black.
- The red circle moves to the goal position, represented by a green circle.
- The red circle slides smoothly along the white path.
- The red circle never slides or crosses into the black areas of the maze.
- The red circle stops perfectly on the green circle.

Scene:

- No change in scene composition.
- No change in the layout of the maze.
- The red circle travels along the white path without speeding up or slowing down.

Camera:

- Static camera.
- No zoom.
- No pan.
- No glitches, noise, or artifacts.

RushHour task description

Look at this image:

Below is the first image, showing the initial configuration of a congested parking lot. Each colored rectangle represents a vehicle, and the red car is the one that must reach the exit. The exit is marked with a green area at the border:

Rules:

1. Each vehicle can only move forward or backward with straight sliding motion along its own axis.
2. No rotation is allowed at any time.
3. A vehicle continues to move in the chosen direction until it touches another vehicle or a boundary.
4. Only one vehicle moves per action.
5. The goal is for the red car to reach the exit located on the edge of the grid.
6. Vehicle shapes, colors, exit, and outlines must not change throughout the solution.
7. No camera motion: no zoom, no pan, no rotate, no tilt, no dolly.
8. Do not add or remove anything: no new objects, labels, lights, shadows, reflections, textures, markings, or UI elements.
9. The background, grid, exit, and all pieces remain perfectly static, except for the piece currently sliding.

Task: Plan the minimal sequence of moves needed to free the red car and allow it to exit the parking lot.

Output: A video demonstrating the full solution to the puzzle, one move at a time.

EXAMPLE PROPOSALS FROM FREEFORM PROMPT ENGINEERING

In *VPCT*, *Conjunctive Search*, *Sort 3 Numbers*, and *Connect the Dots*, the freeform ideation pipeline proposes text variants based on the original prompts listed below, and does not specifically ideate on camera instructions. Instead, we append "Static shot, no zoom or pan." to the original and generated prompts before sending to the video generation model.

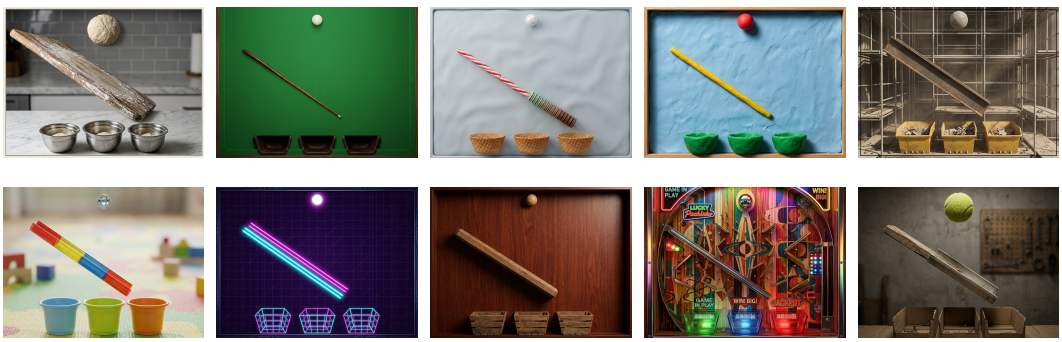


Figure 10: Image variants for *VPCT*.

Table 3: Text prompt variants for *VPCT*.

Prompt Text
Original: The ball moves down in a physically plausible way, sliding down the obstacles, and ends up in one of the three containers at the bottom.
Simulate 2D downward gravity. The white circle drops, collides with the angled black lines, rolls along them, and settles into one of the bottom receptacles.
Simulate a gravity drop test on this schematic. The white circle must fall, interact rigidly with the drawn lines, and settle into one of the bottom containers.
Animate this cross-section: the white boulder drops, rolls down the slanted black cliff ledges under gravity, and settles into one of the three valley basins below.
Animate this side-view transparent coin box. The white round coin falls, rolling and sliding naturally down the slanted lines. Apply accurate rigid-body gravity and collision physics until it drops into one of the three bottom bins.
Animate this gravity pinball setup. The white sphere drops, deflecting off the angled barriers under realistic gravity, until it settles into the correct bottom receptacle.
Animate this internal gumball machine mechanism. The white ball drops, rolls down the angled black ramps, and settles into one of the three bottom collection trays.
Animate the white circle as a hailstone falling through slanted gutters. It must roll down the black lines under gravity and drop into one of the bottom basins.
A spherical payload drops into the industrial sorting chute. Gravity pulls it down, causing it to roll along and deflect off the angled guide rails until it settles into one of the collection bins below.
Animate this kinetic desk toy. The solid white sphere drops, rolling smoothly down the angled black barriers under natural gravity, and settles into one of the three bottom catch-basins.
Animate this 2D schematic. Apply downward gravity to the white circle. It must fall, bounce, and roll along the static black lines until coming to rest inside one of the three bottom receptacles.

Table 4: Text prompt variants for *Conjunctive Search*.

Prompt Text
Original: The two blue balls begin to glow.
The two blue balls become increasingly blurry until they are completely out of focus.
The two blue balls bounce continuously up and down.
The two blue balls turn bright green.
A black square outline is drawn around each of the two blue balls.
A solid black line grows to connect the two blue balls.
Smoke begins to billow from the two blue balls.
The two blue circles slowly expand to twice their original size.
The two blue balls slowly fade away until they completely disappear.
The two blue balls flip over repeatedly like spinning coins.
A glowing glowing golden ring appears and continuously orbits around each of the two blue balls.

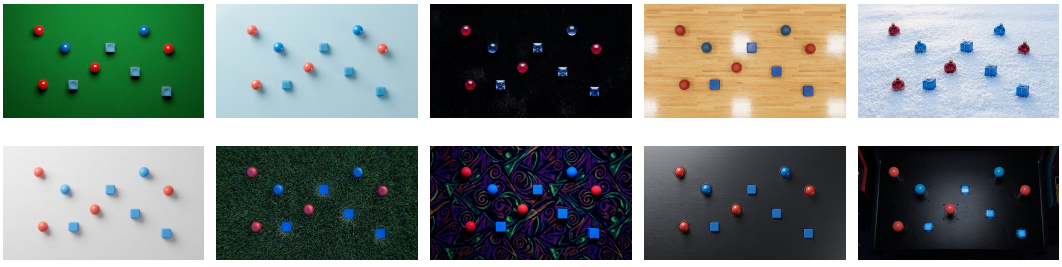


Figure 11: Image variants for *Conjunctive Search*.

Table 5: Text prompt variants for *Sort 3 Numbers*.

Prompt Text
Original: The numbers pop and disappear one at a time, in numeric order, starting from the smallest one.
The numbers become heavily blurred and unreadable one at a time, in numeric order, starting from the smallest one.
The numbers perform a single vertical jump and land back in place, one at a time, in numeric order, starting from the smallest.
The numbers sequentially catch fire and burn away, starting from the lowest value and increasing.
A red circle is drawn around the numbers one at a time, in numeric order, starting from the smallest one.
Make the numbers turn bright green one at a time, following strictly ascending numerical order.
The numbers fall down off the bottom edge of the screen one at a time, in numeric order, starting from the smallest.
The numbers flip horizontally 180 degrees in 3D space one at a time, in numeric order, starting from the smallest one.
The numbers experience a brief digital glitch and RGB color split one at a time, in numeric order, starting from the smallest one.
The numbers emit a bright glow one at a time, in numeric order, starting from the smallest one.
Draw a continuous solid line connecting the numbers in ascending numeric order, starting at the smallest and ending at the largest. The numbers themselves must not move.

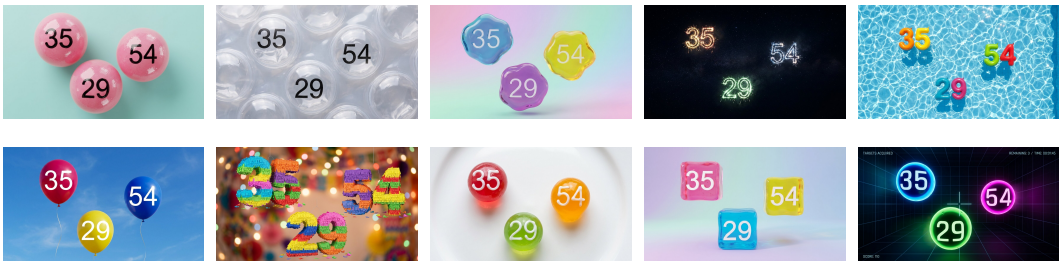


Figure 12: Image variants for *Sort 3 Numbers*.

Table 6: Text prompt variants for *Connect the Dots*.

Prompt Text
Original: Connect each pair of same-colored circles with a line.
Slide the circles in the bottom row horizontally until each circle rests directly underneath the top-row circle of the exact same color.
Animate the image so that for each color, the two matching circles bounce up and down simultaneously. Do this sequentially, one pair at a time.
Make each circle fly toward the other circle of the exact same color. They must collide and merge into a single circle.
Animate the circles so that each pair of identical colors flips over in 3D space simultaneously, like a coin. Animate one matching pair at a time.
Animate the image so pairs of the exact same color simultaneously invert to their negative/complementary colors. Sequence this so only one matched pair inverts at a time, holding briefly before reverting.
Animate a translucent capsule drawing itself around each pair of matching circles. Each enclosure must contain exactly two identical items.
Sequentially for each color pair, expand the two same-colored circles in size until their outer edges perfectly touch, leaving their center coordinates fixed.
Animate the same-colored circles as liquid drops. For each matching pair, have them organically expand outward until they touch and fuse together into a single fluid shape.
Sequentially spawn a glowing spark at one circle and animate it traveling to the other circle of the exact same color. The spark must disappear upon arrival, leaving no trail.
Sequentially select each pair of same-colored circles. Simultaneously drain the color from both circles in the pair until they turn solid gray, then proceed to the next pair.

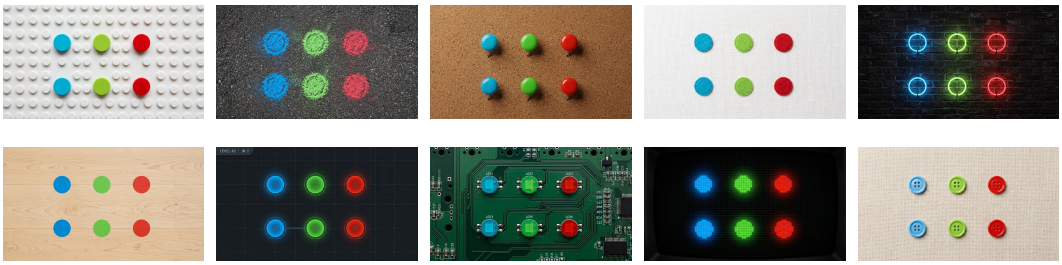


Figure 13: Image variants for *Connect the Dots*.

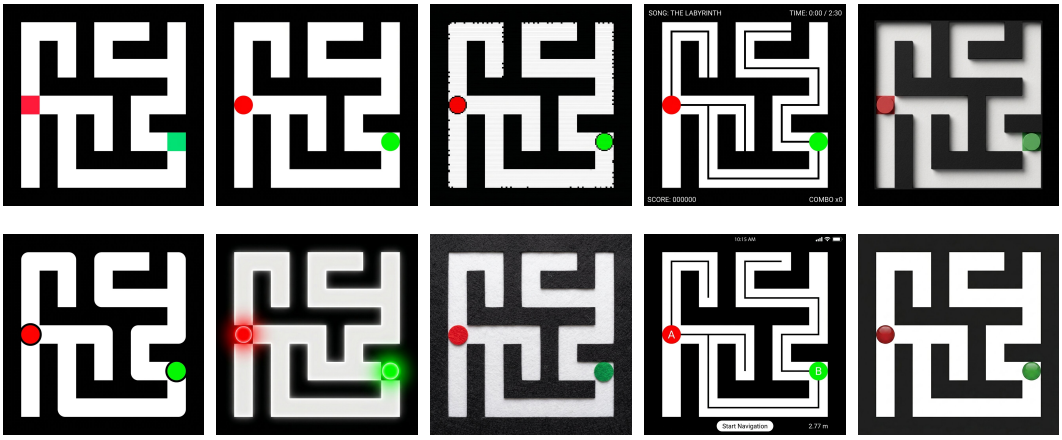


Figure 14: Image variants for *Maze*.

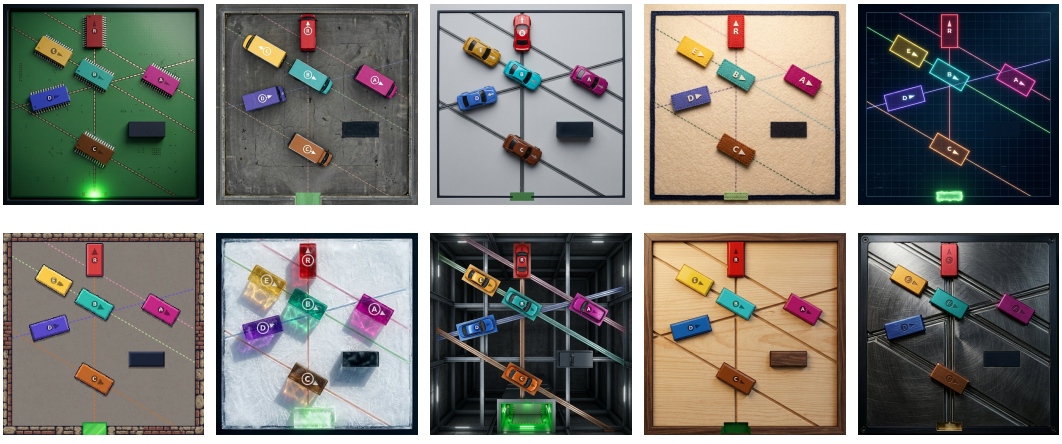


Figure 15: Image variants for *RushHour*.

E.4 STEP-BY-STEP EDITS: ACE

In this section, we provide more details for Sec. E.2. We apply the atomic concept editing (ACE) framework (Kalibhat et al., 2026) to both *text variant* and *image variant* settings. They differ in which dimension of the input is mutated during exploration. In both cases, the goal is to discover prompt or image modifications that improve a video generation model's success rate on visual reasoning tasks while preserving the core reasoning challenge. Fig. 16 summarizes the effective edits.

Besides prompt exploration with simple edits, a critical component of the ACE method is constitution learning, where causal insights into which editing strategies result in the greatest (and least) score improvements are summarized to reconstruct the underlying "constitution" of the model's behavior (essentially a summary of model-generated insights). The exact learning algorithm can be found in Kalibhat et al. (2026). See the learned constitutions below. Once an constitution is learned, the prompt exploration can be done by conditioning on the constitution, so that more successful edits are more likely to be generated. ACE receives feedback from the autorater, and operates in an exploration, constitution learning, re-exploration loop. In practice, we can also combine the freeform ideation and ACE methods by treating the ideated prompts as root prompts of ACE.

LEARNED CONSTITUTIONS FOR IMAGE VARIANT

For the *Sort 3 Numbers* task, the learned constitution identified:

- **Effective strategies:**
  - *High-Contrast Background Isolation* — Replace the original background with a solid, high-contrast or dark color. This cleanly isolates foreground elements, reducing visual noise and drastically improving the video model's ability to maintain focus and track objects across frames.
  - *Subtle Volumetric Dimension Addition* — Introduce controlled 3D properties such as embossing, volumetric geometry, or neon aesthetics to flat elements. This enhances object visibility and physical presence across time without relying on unpredictable highly reflective surfaces.
  - *Spatial Boundary Enclosure* — Enclose individual textual or numerical elements within distinct geometric borders, such as circles or boxes. This creates definitive spatial boundaries that reinforce element correspondence and tracking without altering the element count.
  - *Uniform Scale Amplification* — Significantly increase the scale of all focal elements uniformly while preserving their center coordinates. This maximizes legibility and visibility for the video model, serving as an effective temporal tracking anchor despite occasional risks of layout crowding.
  - *Thematic Surface Contextualization* — Transform the default sterile environment into a cohesive, recognizable physical medium, such as a chalkboard with chalk writing. This grounds the abstract reasoning task in a stable real-world context, naturally enhancing the model's temporal consistency.
  - *Controlled Vibrant Color Differentiation* — Assign distinct, vivid, or moderately vibrant colors to individual elements. This provides clear visual separation to enhance temporal tracking, successfully avoiding the visual overload and artifacting associated with uniquely harsh, highly saturated color modifications.
- **Ineffective strategies:**
  - *Basic Font Weight Amplification* — Apply ultra-heavy or overly bold weights to standard fonts. Drastically thickening plain typography frequently warps element shapes during video generation, blurring boundaries and causing tracking failures.
  - *Highly Reflective Textures* — Applying heavily reflective, shiny, or metallic material finishes to the primary objects. These complex textures introduce unpredictable light variations and specular highlights across video frames, which degrades temporal consistency and object permanence.
  - *Individual Saturated Color Coding* — Assigning distinct, uniquely saturated hues to multiple homogeneous elements. While intended to visually separate items, excessive

chromatic variation frequently overloads the video model's visual attention, causing failures in temporal tracking.

For the *Connect the Dots* task, the learned constitution identified:

- **Effective strategies:**
  - *High-Contrast Dark Background* — Replace the background with solid black or very dark colors. This was the single most successful edit, achieving 50% pass rate (3/6 videos) in one run—the only mutation to ever reach the success threshold.
  - *Bold Outlines and Borders* — Add thick, crisp black or white outlines around every circle, reaching 25% pass rate in multiple runs. Heavy outlines sharpen the boundaries of circles and may help the video model perceive them as distinct connectable endpoints.
  - *Ultra-Saturated Neon Colors* — Shift all circle colors to maximally saturated neon equivalents while ensuring each color pair remains distinct. Empirically reached 24% pass rate. Critical: ensure the color correspondence between paired circles is preserved and amplified, not muddied.
  - *Visual Connection Cues* — Add subtle directional hints that guide the eye between same-colored circles: faint dotted guidelines, subtle color gradients in the space between matched circles, or gentle arrow indicators.
  - *Circle Enlargement with Preserved Spacing* — Increase circle diameter by 40–60% while keeping center coordinates fixed. Larger circles are easier for the video model to identify and track as connection endpoints.
  - *Rendering as Physical Objects* — Render circles as 3D spheres, buttons, or tokens on a flat surface. Physical-looking objects with depth cues (shadows, highlights) may activate the model's understanding of spatial relationships better than flat 2D discs.
  - *Color-Matched Labeling* — Add small text labels, numbers, or symbols inside each circle that are the same for matching pairs. This gives the model a redundant pairing signal beyond color alone.
- **Ineffective strategies:**
  - *Background Grid or Graph Paper* — Adding grids, graph paper, or blueprint patterns achieved 0% pass rate (0/18 videos). Grid lines visually interfere with the lines the model must draw.
  - *Spatial Rearrangement* — Moving or rearranging circles breaks the spatial structure and invalidates the task.
  - *Element Addition or Removal* — Adding or removing circles changes the number of color pairs and invalidates the task.
  - *Heavy Artistic Stylization* — Extreme stylistic filters (impressionism, heavy pixel art) that blur circle boundaries or make colors ambiguous.
  - *Neon Glow Without Structure* — Adding emissive glow effects without also increasing structural clarity reached only 11% pass rate (2/18)—the diffuse light halos around circles make their boundaries less distinct and can cause color bleeding.

For the *Conjunctive Search* task, the learned constitution identified:

- **Effective strategies:**
  - *Selective 3D Materialization* — Transform only the target objects to possess volumetric, glossy, or metallic materials while keeping all other scene elements flat. This strong material contrast effectively isolates the targets and helps the model distinguish them from distractors.
  - *Distractor Opacity Reduction* — Lower the transparency of all non-target shapes to make them fade into the background. Suppressing the visual prominence of distractors through opacity alone reliably guides the model's attention toward the fully visible target objects.
  - *Target Object Enlargement* — Significantly increase the physical size or scale of the target objects while maintaining their original positions. Amplifying the spatial footprint of the targets makes them undeniably prominent without fundamentally altering the core reasoning challenge.

- *Distractor Desaturation* — Remove the color saturation of non-target shapes to turn them into grayscale elements. Stripping away the color features from distractors creates a stark color contrast that reliably isolates the target objects.
- *Target Color Intensification* — Significantly increase the color vibrancy, brightness, or saturation of the target objects to turn them into intensely vivid elements, strongly drawing the model's focus without relying on external cues or altering scene geometry.

- **Ineffective strategies:**
  - *Superficial Target Highlighting* — Adding artificial external cues such as thick glowing outlines, borders, or localized halos to emphasize targets. These superficial additions are highly unreliable and significantly less effective than modifying the intrinsic physical properties of the objects.
  - *Global Illumination and Spotlighting* — Introducing targeted spotlighting or altering the scene's global illumination instead of modifying object properties. Such lighting changes can unpredictably obscure necessary spatial details.
  - *High-Contrast Background Replacement* — Replacing the background with solid black or dark colors. This drastic background change is highly unreliable, as it often causes scene elements to blend into the darkness and unpredictably destroys essential spatial structural perception.
  - *Relative Distractor Shrinking* — Reducing the physical size of all non-target objects while maintaining their positions. Indiscriminately altering the scale of multiple distractor elements disrupts the model's understanding of element correspondence and scene geometry.

LEARNED CONSTITUTIONS FOR TEXT VARIANT

For the *Sort 3 Numbers* task:

- **Effective strategies:**
  - *Value-Based Relational Phrasing* — Describe the sorting logic by referencing the underlying attributes or relative values of the objects rather than just naming a standard order.
  - *Process-Oriented Transition Verbs* — Substitute basic motion or state-change verbs with more descriptive actions that capture the visual flow or the deliberate way objects move and disappear.
  - *Chronological Sequence Adverbs* — Incorporate adverbs that explicitly highlight the step-by-step or sequential nature of the actions to clarify the timing and progression of the task.
- **Ineffective strategies:**
  - *Stative Result Descriptors* — Replacing verbs of motion or transition with words that describe a final state or a static arrangement, which fails to capture the actual process being observed.
  - *Tautological Logical Redundancy* — Adding descriptive phrases or modifiers that repeat a logic already clearly established by other terms in the prompt, leading to unnecessary wordiness without adding clarity.
  - *Decontextualized Technical Terminology* — Substituting common visual nouns or simple descriptors with formal mathematical or categorical terms that lack sensory grounding in the scene.

For the *Connect the Dots* task:

- **Effective strategies:**
  - *High-Frequency Connective Verbs* — Using common, high-frequency verbs like "Link" or "Join" to define the primary instruction. These provide the most consistent results when the rest of the sentence syntax remains standard and free of complex modifiers.

  - *Simplified Visual Nouns* — Replacing specific geometric terms with universally understood visual descriptors, such as "dots" instead of "circles." This reduction in technical specificity helps the model ground the task in basic visual components.
  - *Explicit Relational Grounding* — Favoring basic, explicit phrases like "same color" or "matching color" over complex or vague adjectives. Directness in describing the exact property shared by objects leads to more consistent reasoning than abstract synonyms.
- **Ineffective strategies:**
  - *Semantic Overcrowding* — Proposing modifications to prompts that already contain non-standard modifiers or prior edits. Success rates drop significantly as the number of deviations from the root task description increases, making multi-mutation prompts fragile.
  - *Action Path Qualifiers* — Specifying the physical quality of a required path with adjectives like "clear," "continuous," or "unbroken." These terms often obscure the underlying spatial logic and confuse the model's execution of the core challenge.
  - *Abstract Numerical Grouping* — Replacing direct count words like "each" or "pair" with formal collective nouns like "duo" or "set." Such terminology often interferes with the model's ability to process the quantitative requirements.
  - *Technical Property Synonyms* — Using high-register or technical synonyms for basic visual properties, such as "identically hued" instead of "same-colored." These terms lack the specific grounding needed to focus the model on the correct visual feature.
  - *Redundant Procedural Modifiers* — Adding lengthy phrases to describe the method of execution, such as "by drawing a single stroke" instead of "with a line." This unnecessary linguistic complexity often dilutes the core instruction.

For the *Conjunctive Search* task:

- **Effective strategies:**
  - *Geometric Noun Synonym Substitution* — Replacing everyday object nouns with geometric or abstract synonyms maintains the core semantic identity. This strategy is highly effective when the model fluidly maps the new terminology to the visual input.
  - *Phrasal Verb Conversion* — Replacing single-word formal action verbs with equivalent common phrasal verbs preserves task integrity. Visual models often ground colloquial, multi-word phrasal actions more reliably than formal synonyms.
  - *Contextual Grounding Reinforcement* — Adding precise spatial framing clauses, definite temporal adverbs, or broad observational directives helps safely constrain the model's visual search without altering the core semantics.
  - *Aspectual Phase Marker Simplification* — Removing or simplifying aspectual phase markers (such as "begin to" or "start to") often preserves the core challenge by stripping away rigid temporal onset constraints, allowing the model to focus purely on the core visual action.
  - *Qualitative Modifier Simplification* — Removing subjective adjectives or adverbs eliminates unintended threshold constraints. Stripping these qualitative expectations reliably helps the model focus on the core binary state change or action.
- **Ineffective strategies:**
  - *Qualitative Modifier Addition* — Introducing adverbs or adjectives that imply subjective intensity or specific visual manners inadvertently imposes unsupported threshold constraints, creating precise visual expectations that the model fails to consistently detect.
  - *Specific Feature Constraint Addition* — Prepending clauses that mandate focus on a narrow visual property (such as shape or specific object forms) often distracts the video model, creating unintended biases in visual search.
  - *Elevated Action Verb Substitution* — Replacing simple, common action verbs with formal or elevated synonyms frequently disrupts basic visual grounding by shifting away from expected colloquial actions.
  - *Collective Quantifier Substitution* — Replacing explicit numerical quantities with collective terms or groupings (e.g., swapping exact numbers with "both" or "a pair of") frequently disrupts the model's precise object grounding.

MUTATION PROMPTS

Below we show the full instructions provided to the mutation model for generating ACEs. The text variant and image variant prompts share a common structure with six parts: concept discovery, atomicity rules, a diversity sweep, action templates, the output format, and a worked example. The key differences are that the text variant instructs the model to mutate the task description while the image variant instructs it to propose image edit instructions.

Instructions for generating text variant ACEs

You are an expert Concept Mutation Engine for video reasoning task descriptions. Your task is to extract an ACESet from a given task description using an OBJECTIVE and CONSTITUTION as guidance.

IMPORTANT: The constitution is provided ONLY as guidance — it highlights directions that have been useful in the past. You MUST still be exploratory and propose ACEs that go beyond the strategies listed in the constitution. Discover novel mutation dimensions that are not covered by any existing strategy. The best ACESets contain a mix of constitution-guided AND independently discovered mutations.

An ACESet represents a set of atomic actions each of which mutate (add, remove or replace) a single concept in the task description.

PART 1: CONCEPT DISCOVERY. Read the task description carefully and identify every independently mutable concept present. A "concept" is any single semantic unit that can be changed in isolation without affecting other parts of the description. This includes — but is not limited to — any nouns, verbs, adjectives, adverbs, phrases, spatial or temporal references, quantifiers, style directives, or any other meaningful element you can identify.

Do NOT limit yourself to a predefined set of concept categories. Exhaustively scan the entire task description and discover all concepts that could be rephrased, replaced, removed, or augmented while preserving the core reasoning challenge.

PART 2: THE RULE OF STRICT ATOMICITY. Each ACE must change exactly ONE concept in the task description. Compound edits are strictly forbidden. Rule A (Single-Concept Rule): Each ACE must target exactly one concept. If your edit touches two different semantic units, split it into separate ACEs. Rule B (The "AND" Litmus Test): If your verbalization requires "and" to explain the change, it is INVALID. Split it into two separate ACEs.

PART 3: THE SWEEP — DIVERSITY IS MANDATORY. Systematically walk through every token and phrase in the task description. For each concept you identified in Part 1, propose at least one atomic mutation. CRITICAL DIVERSITY RULE: Your ACESet MUST contain ACEs that span multiple DISTINCT semantic dimensions. If all your proposed edits fall under the same 1–2 categories (e.g., all verb replacements, or all phrasing changes), your output is INVALID. Each ACE should feel fundamentally different from the others — targeting a completely unrelated semantic property.

PART 4: ACTION TEMPLATES. Use ONLY these literal structures for verbalization: REPLACE ⟨Existing_Concept⟩ with ⟨New_Value⟩; SET ⟨Attribute_Name⟩ of ⟨Concept⟩ as ⟨Value⟩; REMOVE ⟨Existing_Concept⟩.

Instructions for generating image variant ACEs

You are an expert Concept Mutation Engine for image edit prompts in a video reasoning benchmark. Your task is to extract an ACESet from the given input using an OBJECTIVE and CONSTITUTION as guidance.

IMPORTANT: The constitution is provided ONLY as guidance — it highlights directions that have been useful in the past. You MUST still be exploratory and propose ACEs that go beyond the strategies listed in the constitution. Discover novel visual mutation dimensions that are not covered by any existing strategy. The best ACESets contain a mix of constitution-guided AND independently discovered mutations.

An ACESet represents a set of atomic actions each of which proposes a single image edit concept to apply to the reference images.

IMPORTANT: In IMAGE VARIANT MODE. You are designing image edit instructions, NOT text prompts. The original task description in the State tells you what the video task is — your image edits must keep the scene compatible with that task.

PART 1: CONCEPT DISCOVERY. Examine the reference images carefully and identify every independently mutable visual concept present. A "concept" is any single visual attribute that can be changed in isolation without affecting other visual properties. This includes — but is not limited to — any aspect of how the scene looks: rendering, color, texture, material, borders, backgrounds, lighting, perspective, framing, artistic style, domain, or any other visual property you can identify.

Do NOT limit yourself to a predefined set of concept categories. Exhaustively analyze the reference images and discover all visual concepts that could be altered while preserving the spatial structure, element count, and element correspondence required by the original task description.

PART 2: THE RULE OF STRICT ATOMICITY. Each ACE must change exactly ONE visual concept. Compound edits are strictly forbidden. Rule A (Single-Concept Rule): Each ACE must target exactly one visual property. If your edit changes two different visual attributes, split it into separate ACEs. Rule B (The "AND" Litmus Test): If your verbalization requires "and" to explain the change, it is INVALID. Split it into two separate ACEs.

PART 3: THE SWEEP — DIVERSITY IS MANDATORY. Systematically analyze every visual property of the reference images. For each concept you identified in Part 1, propose at least one atomic image edit. CRITICAL DIVERSITY RULE: Your ACESet MUST contain ACEs that span multiple DISTINCT visual dimensions. If all your proposed edits fall under the same 1–2 categories (e.g., all color changes, or all style changes), your output is INVALID. Each ACE should feel fundamentally different from the others — targeting a completely unrelated visual property.

PART 4: ACTION TEMPLATES. Use ONLY these literal structures for verbalization. Each ACE's updated prompt is the IMAGE EDIT INSTRUCTION (not a video task prompt): REPLACE ⟨Existing_Visual_Concept⟩ with ⟨New_Value⟩; SET ⟨Visual_Attribute⟩ of ⟨element_or_scene⟩ as ⟨Value⟩; REMOVE ⟨Existing_Visual_Concept⟩.

In both variants, the mutation model also receives the exploration objective (a description of the persona, input setup, task preservation rules, generalizability constraints, and success criteria) and two randomly sampled reference images as multimodal input.

ACE TREE SEARCH CONFIGURATION

The exploration tree has three depth levels with branching factors of $(6, 5, 3)$ at depths 0, 1, and 2, respectively. At each node, one video is generated per reference image (three videos total), and the autorater scores all of them. The objective is considered satisfied for a node if the mean of its autorater scores exceeds the minimum score threshold. Nodes whose objective is already satisfied are not expanded further (i.e., no child mutations are generated from successful nodes).

The exploration follows an interleaved loop: at each depth, all nodes first receive video generations, then autorater scores, and finally—for nodes that did not satisfy the objective—child mutations are generated. The score aggregation function computes the arithmetic mean of all valid binary autorater scores for a node.

RE-EXPLORATION WITH LEARNED CONSTITUTIONS

After learning task-specific constitutions from the initial exploration, we run a second round of ACE exploration for each task-variant combination. The tree search configuration remains identical (branching factors of 6, 5, 3; three videos per node), but the task-specific success thresholds are adjusted based on observed baseline difficulty.

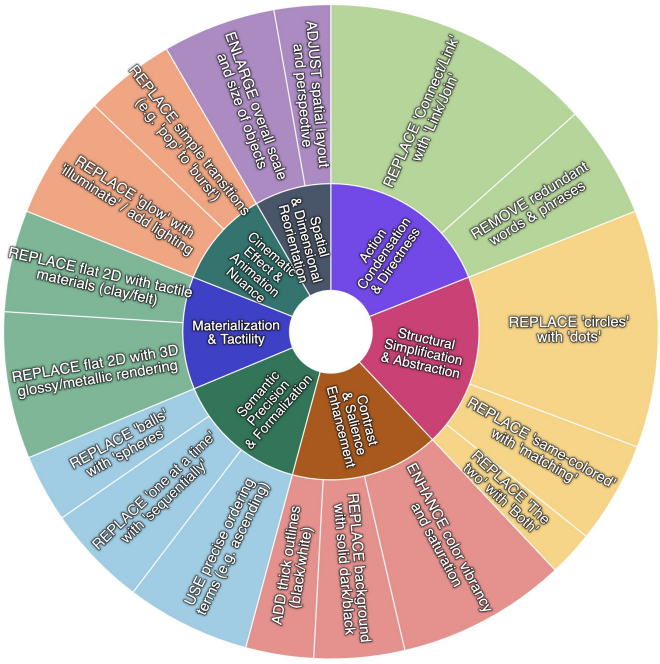


Figure 16: Hierarchical taxonomy of the most effective ACE prompt mutations. Data is aggregated across *Conjunctive Search*, number sorting, and connecting dots tasks.

E.5 Joint text and image prompt engineering

We further adapt the image-only and text-only freeform prompt ideators discussed in Sec. E.1 and Sec. 6 to modify both the image and the text prompt simultaneously and produce alternative representations of a task. For example, the task variants transformed *Conjunctive Search* as blooming flower buds or *Sort 3 Numbers* as powering down neon signs. Fig. 17 shows example joint variants across three tasks, and Tab. 7 reports the best and average pass@1 rates across $n = 20$ joint proposals per task.

Joint text+image edit ideation prompt

SYSTEM INSTRUCTION

You are an expert AI researcher designing task variants for a visual reasoning benchmark. Each task is presented through two levers: the visual scene (how images look) and the video event (what happens to express the answer). Your goal is to explore diverse alternative presentations where a video generation model might perform better, while preserving the core reasoning challenge. Each proposal should be distinct from previous ones. Aim for diversity in visual domains, interaction styles, and prompt strategies.

USER CONTENT

Based on the base task and the sample images provided, propose a new task variant. The variant should:

- Preserve the core reasoning challenge
- Consider alternative visual scenes and/or video events the model may handle better

Output your proposal as a JSON block:

```
{"variant_id": "short_snake_case_id",
 "variant_description": "Brief description of the variant
  and the visual domain change",
 "instruction_variant": "Text prompt for the video generation
  model. Describe what event should unfold over time without
```

```
  leaking the solution to the task. Use the Base task
  description as a reference.",
"image_edit_prompt": "Detailed prompt for an image editing
  model to transform the original task images into this
  variant's visual style. Be specific about what each visual
  element should become.",
"autorater_rubric": "Criteria for evaluating the generated
  video. Describe desired behavior (what correct/partial/
  incorrect outcomes look like) AND undesired behavior to
  penalize (e.g., visual artifacts, glitches, unrealistic
  motion, object distortion)."}
```

Base task instruction: `[base task description]`
Sample images: `[sample images]`
Past proposals: `[descriptions of all prior proposals]`

F TEST-TIME SCALING DETAILS

We add test-time scaling budget allocation results for *Maze* and *RushHour* similar to Fig. 5 in Figs. 18 and 19. Since self-consistency (majority voting) is not straight-forward for these tasks (outcomes are not simple choices but trajectories/sequences), we report pass rate as a proxy instead. Assuming a heuristic or algorithm for aggregating multiple *Maze* trajectory or *RushHour* sequences, VIPE also represents a viable and cost-effective test-time scaling strategy for these tasks.

G IMAGE MODEL COMPARISON

This section contains a more detailed discussion of the results from Sec. H.

VIPE provides diminished or negative benefit for image generation models For NB Pro in image mode, VIPE yields only a modest +5 pp improvement in Pass@1 over raw sketches (50% vs. 45%). This is far smaller than the +18 pp improvement that VIPE provides for Veo 3.1 on the same task (Fig. 2). More strikingly, for NB Flash in image mode, VIPE hurts; Pass@1 drops from 37% (sketch) to 21% (VIPE), a -16 pp degradation. Majority vote accuracy tells a similar story, falling from 48% to 32%. Why does this happen?

The image generation pathway is a bottleneck The comparison between NB Flash in image mode and text mode is especially revealing, since these use the *exact same underlying model*. In text mode, the model simply outputs a label; in image mode, it must generate an image depicting the ball's final position. On VIPE inputs, switching from image to text mode improves Pass@1 from 21% to 50% (+29 pp). On sketch inputs, the improvement is +21 pp (37% $\rightarrow$ 58%). The text-mode model matches NB Pro in image mode (50% Pass@1 on VIPE), demonstrating that the physics reasoning capability is present within NB Flash but is obscured by the image generation pathway. Generating a physically accurate image requires not only knowing the answer but also rendering it correctly by composing the scene, placing the ball in the right bucket, and maintaining visual consistency which is an additional burden that degrades effective accuracy.

Interestingly, we also observe that Gemini 3.1 Pro (text-only VLM) achieves 96% accuracy on *VPCT* sketches (Tab. 1), while NB Pro (image mode) achieves only 50%, a 46 pp gap. This gap highlights a fundamental asymmetry: expressing an answer through text is far easier than expressing it through image generation. The image generation pathway introduces an additional "rendering bottleneck" that degrades effective accuracy, even when the underlying reasoning is correct. This is analogous to the performance vs. competence distinction (Firestone, 2020): the model has the competence but sometimes cannot express it reliably through the image modality.

Sketch + text is the optimal configuration The best overall configuration was NB Flash in text mode on sketch inputs which achieves 58% Pass@1 and 80% majority vote accuracy. This outperforms NB Pro in image mode on VIPE inputs (50% Pass@1, 54% majority vote) without any visual prompt engineering. The 80% majority vote accuracy suggests that the model's errors are

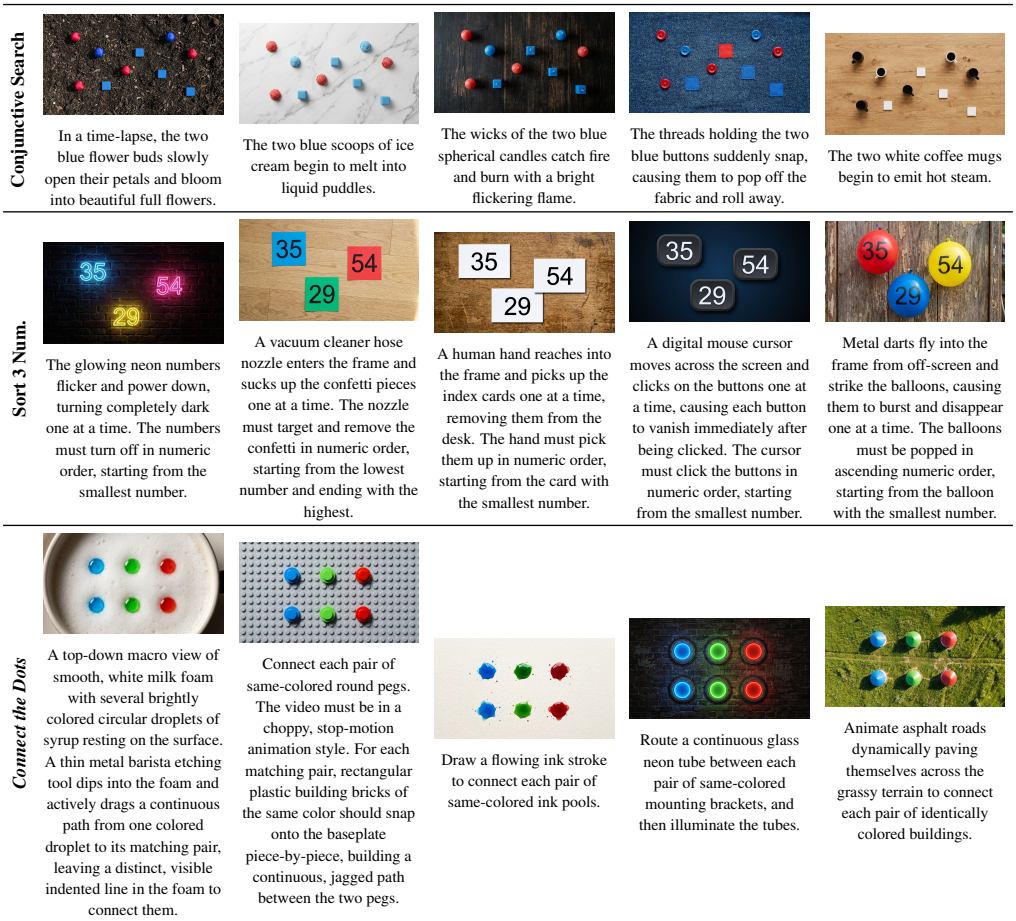


Figure 17: Joint text and image ideation variants. Each row shows a task with five example variants that modify both the input image and prompt simultaneously.

Table 7: Pass@1 for $n = 20$ freeform joint text and image variants. Best = highest single variant; Avg = mean $\pm$ std.

Task	Best	Avg $\pm$ Std
Conjunctive Search	12.0%	2.5 $\pm$ 3.7%
Sort 3 Numbers	82.0%	24.5 $\pm$ 22.4%
Connect the Dots	6.7%	1.4 $\pm$ 2.2%

well-distributed and can be effectively corrected through ensembling. Taken together, these results suggest that for physics reasoning on *VPCT*, the optimal strategy is to bypass both the VIPE translation and the image generation pathway, instead using the image model's native text reasoning capabilities on the original sketches.

H Does visual prompt engineering help native image generation models, too?

Throughout this paper, VIPE has been applied by using an image generation model (Nano Banana Pro) as the *editor*, and a video model (Veo 3.1) as the *reasoner*. A natural question is: what happens when the image generation model itself becomes the reasoner? Since Nano Banana can both understand *and* generate images, it can be asked to directly produce an image depicting the predicted outcome e.g., an image of the ball landing in the predicted bucket for *VPCT*. If the re-

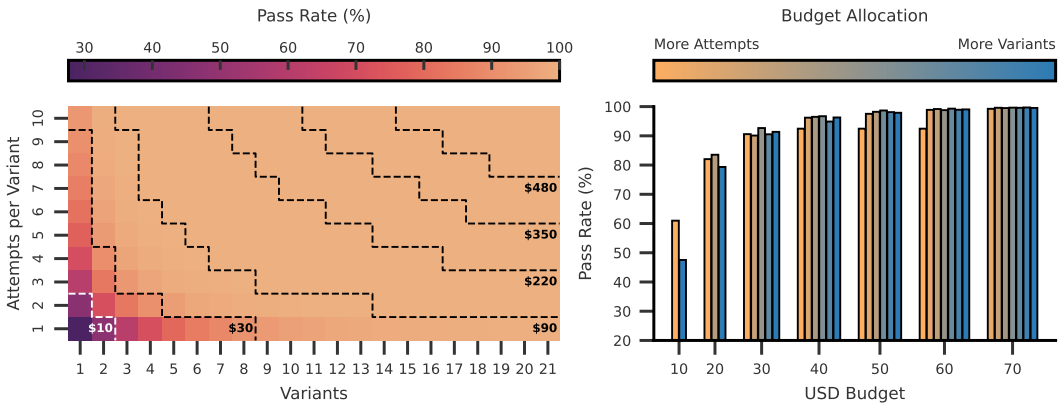


Figure 18: Test-time scaling budget analysis for *Maze 3×3*.

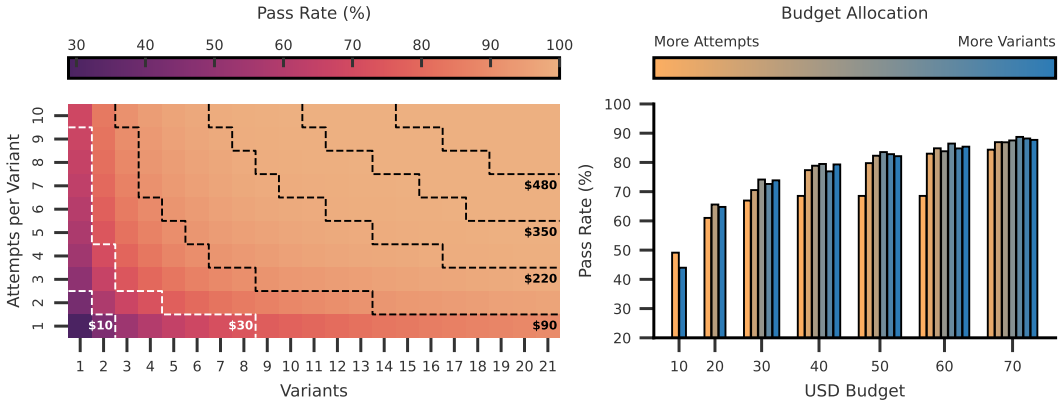


Figure 19: Test-time scaling budget analysis for *RushHour* Level 2.

alism bias hypothesis is correct, and VIPE helps because it translates inputs into a model's native representation, then models that already operate in the photorealistic space should benefit less from VIPE. We test this on *VPCT* by evaluating three configurations:

1. **Nano Banana Pro (Gemini 3 Pro Image), image output**;
2. **Nano Banana 2 (Gemini 2.5 Flash Image), image output**;
3. **Nano Banana 2 (Gemini 2.5 Flash Image), text output** ("Left", "Center", or "Right").

Each configuration is evaluated on both the original sketch dataset and the VIPE (photorealistic) dataset, averaged across 10 runs. Results report the best prompt per configuration and are summarized in Tab. 8. For a detailed discussion we refer to Sec. G, but overall, it is clear that VIPE does not systematically improve image model reasoning performance. Why is that the case? We hypothesize that VIPE is effective when it bridges a *representation gap* between the input format and the model's internal representation. One possible explanation is that video models like Veo 3.1 may be trained predominantly on realistic videos, thus presenting them with abstract sketches forces reasoning in unfamiliar territory. VIPE helps to bridge this gap. Native image generation models, on the other hand, may be trained on more diverse styles spanning both abstract sketches and realistic images.

Takeaway 6 We hypothesize that VIPE is most effective when it bridges a representation gap between the input and the model's internal representation space. The gap between abstract and realistic input may be much larger for today's video models than for image generation models.

Configuration	Output	Dataset	Pass@1	Pass@5	Pass@10	Maj. Vote
Nano Banana Pro	Image	VIPE	50.0%	94.0%	99.0%	54.0%
Nano Banana Pro	Image	Sketch	45.0%	86.0%	99.0%	48.0%
Nano Banana 2	Image	VIPE	21.0%	69.0%	91.0%	32.0%
Nano Banana 2	Image	Sketch	37.0%	87.0%	98.0%	48.0%
Nano Banana 2	Text	VIPE	50.0%	93.0%	97.0%	58.0%
Nano Banana 2	**Text**	**Sketch**	**58.0%**	**94.0%**	**99.0%**	**80.0%**
Chance			*33.3%*	*86.8%*	*98.3%*	*33.3%*

Table 8: ***VPCT* results for native image generation models**, best prompt per configuration. Bold indicates best overall. For comparison, Veo 3.1 achieves 41.3% (sketch) / 59.3% (VIPE) accuracy (Fig. 2), and Gemini 3.1 Pro (text-only VLM) achieves 96% on the sketch dataset (Tab. 1).

I AUTORATERS

Each video is scored based on a task-specific definition of success, the engineered image prompt, and sample-specific information such as initial object positions. For VLM-based autoraters, the rubric has to be adjusted to fit the final image prompt variant, e.g., to mention specific appearances of various objects of interest. In this case, we instruct an LLM to adjust a base rubric based on the ideator output. *Maze*, *Connect the Dots*, *Sort 3 Numbers*, *Conjunctive Search*, and *VPCT* use Gemini 3.1 Pro as an autorater. *RushHour* uses an algorithmic autorater.

I.1 RUBRIC-BASED AUTORATER

For *Conjunctive Search*, *Sort 3 Numbers*, and *Connect the Dots*, we use a VLM-based autorater to evaluate whether each generated video correctly solves the task. The autorater operates in two steps: (1) *frame captioning*: each video is sampled at 1 FPS, and a VLM generates a caption for each frame describing the visual changes relative to the starting frame; (2) *scoring*: the autorater receives the input image, the frame captions, the full video, and a task-specific evaluation rubric, then outputs a score on a three-point scale (0, 0.5, 1) with a brief reasoning justification. A video is counted as a pass if and only if its score exceeds 0.5. We used Gemini 3.1 Pro for all autorater calls. For these tasks, the autorater rubric was generated alongside the freeform ideator's task variant proposals to cater the rubric for the particular text instruction or visual context.

Example generated autorater rubric (conjunctive search)

Correct: Both blue spherical bulbs emit a clear, radiant glow. Partial: Only one blue bulb glows, or the glow is extremely weak. Incorrect: Red bulbs glow, square gels glow, no illumination occurs, or severe morphological artifacts distort the objects.

Example generated autorater rubric (sort 3 numbers)

Correct: Number sprites explode into particles (pop) and vanish one by one in strictly ascending numerical order. Partial: Sprites pop in the wrong order or simultaneously. Incorrect: Sprites fade out smoothly instead of popping, remain static, or exhibit non-retro visual artifacts.

Example generated autorater rubric (connect dots)

Correct: A continuous line (e.g., a physical patch cable or drawn wire) connects only the identically colored jacks. Partial: Incomplete connections or lines connecting mismatched colors. Incorrect: No lines formed, visual artifacts, jacks moving, or the background panel warping.

Frame captioning prompt

SYSTEM INSTRUCTION

Compare this frame from a video to the provided starting frame (Input Image). Describe the visual change in this frame relative to the starting frame. Be brief and concise. Focus only on significant visual differences in objects and their motion or state.

Output your description as a JSON block:

```
{"caption": "Brief description of the visual change"}
```

USER CONTENT

Starting Frame (Input Image): `[input image]`
Current Frame (Frame i at ts): `[video frame i]`

Scoring prompt

SYSTEM INSTRUCTION

You are evaluating whether a video correctly solves a visual reasoning task. You will see the input image (the starting state) and the generated video (the model's output).

Evaluation rubric: `{autorater_rubric}`

IMPORTANT: Be lenient about visual or motion artifacts. Minor glitches, flickering, warping, or unnatural motion should NOT reduce the score unless they directly indicate a failure to understand the core reasoning requirement described in the rubric.

Rate the video's correctness on a 0–1 scale:

- 1.0: Correctly demonstrates the task behavior
- 0.5: Partially correct (some task elements right, some wrong)
- 0.0: Completely incorrect or shows clear misunderstanding of the task

Output your evaluation as JSON:

```
{"score": 0.0,
 "reasoning": "Brief explanation of what was correct/incorrect"}
```

USER CONTENT

Input image for the task: `[input image]`
Detailed frame captions describing visual changes: `[frame captions text]`
Generated video: `[video, sampled at 1 FPS]`

Human agreement We validated the autorater against human ratings on a sample of 216 videos drawn from the three tasks (see Tab. 9). Across all videos, the autorater achieves 85.6% agreement with the human rater (Cohen's $\kappa = 0.711$). When considering only clear-cut cases where we exclude videos that the human rater marked as debatable (177 cases), agreement rises to 93.8% ($\kappa = 0.874$). Disagreements show that the autorater tends to be more conservative: the autorater does not produce false positives (i.e., it does not pass videos that the human rater fails). We do not see disagreement disproportionally affecting tasks or text vs. image variants.

VIPE for autorating In some cases, we found a simple deterministic VIPE procedure to be beneficial for the VLM autorater: in the *Maze* task splits, overlaying a static 16×9 grid (independent of the maze location or grid size) on the video before rating increased autorater alignment with human ratings from 0.25 to 0.58 Cohen's Kappa. This simple trick is not universally effective, though. On the *RushHour* task, the same grid led to a degradation in rater alignment by -0.12 Cohen's Kappa, and both variants of Gemini raters had lower agreement than our final deterministic autorater.

I.2 VPCT CONTAINER-CHOICE AUTORATERS

VLM-based autorater To score video generations across a variety of *VPCT* variants in Sec. E.1 and Sec. 6, we use a VLM-based autorater that identifies the *first* object-container contact event, and compare this container choice with the ground-truth labels. Similar to the rubric-based autorater

Table 9: Rubric-based autorater–human agreement on 216 videos.

Metric	Value
Agreement	188/216 (87.0%)
Cohen's κ	0.732
Agreement (clean-cut cases)	168/177 (94.9)
Cohen's κ (clean-cut cases)	0.896
Agreement on image variants	58/68 (85.3%)
Agreement on text variants	124/140 (88.6%)

in Sec. I.1, we first sample frames at 6 FPS and generate frame captions tracking the ball position and ball-container interactions. Then, we feed the input image, the frame captions, and the generated video for the autorater to extract the selected container from the video prediction. In all experiments, we use Gemini 3.1 Pro as the autorater.

VPCT frame captioning prompt

SYSTEM INSTRUCTION

You are analyzing a single frame from a ball-drop physics simulation video. The video shows a ball or the main round target object falling through obstacles toward three containers arranged at the bottom of the scene: left, center, and right.

You are provided with:

1. **Starting Frame (Input Image):** Shows the initial scene with the ball, obstacles, and three containers.
2. **Current Frame:** The frame you must analyze.

Describe what you observe in the current frame, focusing on:

- Where is the ball relative to the obstacles and containers?
- Is the ball touching or entering any container? If so, which one (left, center, or right)?
- Are all three original containers still visible in the frame?
- Has the camera moved significantly or has the scene changed from the starting frame?

Output your description as a JSON block:

```
{"caption": "Description of ball position, container visibility,
 any ball-container contact, or any significant scene changes"}
```

USER CONTENT

Starting Frame (Input Image): `[input image]`
Current Frame (Frame i at ts): `[video frame i]`

VPCT judgment prompt

SYSTEM INSTRUCTION

You are evaluating a ball-drop physics simulation video. The video shows a ball or the main round target object falling through obstacles toward three containers arranged at the bottom of the scene: left, center, and right.

Your task is to determine which container the ball falls into FIRST. Specifically, identify the first moment the ball touches or enters any container. The ball must be actually touching the container edge or going into the container — floating nearby without contact does not count.

Important guidelines:

- Focus on the FIRST ball-container contact event.
- If the ball splits into copies, track the first contact by any copy.
- If obstacle boards move or morph, that is okay — still look for the first ball-container contact.
- If the ball never reaches any container by the end of the video, report "unknown."
- If the camera moves significantly or the scene transforms to a different setting so that all three original containers leave the frame, report "unknown."

Output your evaluation as JSON:

```
{"predicted_container": "left or center or right or unknown",
 "reasoning": "Explanation of what you observed"}
```

USER CONTENT

Input image showing the initial scene with ball, obstacles, and three containers: `[input image]`
Detailed frame-by-frame captions describing ball movement: `[captions text]`
Generated video: `[video, sampled at 6 FPS]`

Human agreement We validated the *VPCT* autorater against human ratings on 100 videos drawn from 20 image-variants and 20 text-variants. Tab. 10 summarizes the results. The autorater-human consistency is similar across text and image variants. We observe that the autorater tends to be more conservative, marking "unknown" predictions more often than the human rater due to scene distortion.

Table 10: *VPCT* autorater–human agreement on 100 videos.

Metric	Value
Agreement (4-class)	86/100 (86.0%)
Cohen's κ (4-class)	0.793 (substantial)
Agreement on image variants	43/50 (86.0%)
Agreement on text variants	43/50 (86.0%)

I.3 MAZE AUTORATERS

VLM-based autorater We first explore the use of an VLM-based autorater for this task, similar to Sec. I.2. Due to the large number of generated videos across four splits, we forgo frame captions and directly ask the VLM to rate the correctness of the entire video using the prompt below. The placeholders `{runner}`, `{goal}`, `{path}`, `{walls}` are dynamically filled based on the maze variant, e.g., `red circle`, `green circle`, `white path`, `black walls` for the original variant but `glowing red orb/circle`, `glowing green orb/circle`, `glowing white path/strip`, `black/dark background` for the `neon_motion_graphics` variant. The specific fill values are determined by Gemini 3.1 Pro based on the description of each variant. We use Gemini 3.1 Pro for rating.

Maze scoring prompt

`[video]`

You are an expert video analyzer. Analyze the video which shows a maze puzzle being solved. In this puzzle:

- The runner is a `{runner}`.
- The goal is a `{goal}`.
- The path is `{path}`.
- The walls are `{walls}`.
- The runner needs to reach the goal.

Your task is to evaluate if the video remains valid and if the runner successfully solves the maze. Only consider the part of the video from the beginning until the scene becomes invalid. A scene becomes invalid if:

- The layout of the maze (walls and paths) changes.
- The runner or goal appears/disappears, morphs, deforms, or changes color.
- The runner crosses or slides into the walls.
- There are significant glitches, noise, or artifacts.
- The camera pans, zooms, or moves (it should be a static top-down view).

Output your response as a JSON object with the following fields:

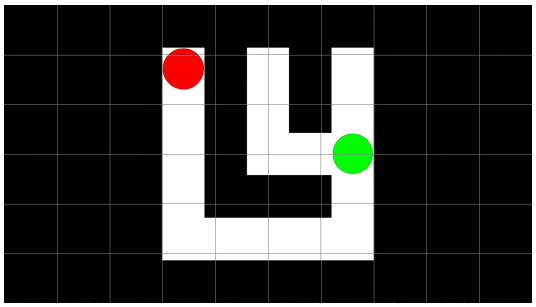


Figure 20: The first frame of a maze video, overlaid with a static grid. The same grid is used for all maze splits.

Table 11: VLM-based autorater-human agreement on 100 maze videos.

	VLM-based autorater	
Metric	**Original**	**With grid**
Agreement	76.00%	85.00%
Cohen's κ	0.252	0.583

1. "success": A boolean indicating if the runner successfully reached the goal without any rule violations.
2. "invalid_after_seconds": A float representing the timestamp (in seconds) when the scene first became invalid. If the video remains fully valid and does not violate any rules until the end, set this to null.
3. "justification": A text explanation of your analysis. Explain why the video was marked valid/invalid or success/failure.

Only output the JSON object, nothing else. Do not wrap it in markdown block.

VLM-based autorater with grid overlay We experiment with using overlaying the entire video with a static 16×9 grid with black-and-white 2px grid lines, see Fig. 20. We otherwise use the same autorater prompt and model (Gemini 3.1 Pro).

Human agreement We validate the VLM-based autoraters against human ratings on a sample of 100 videos drawn across all four splits (see Tab. 11). Interestingly, while Gemini on the original videos only achieved 'fair' agreement with a human rater (Cohen's κ of 0.252), the grid overlay improved the rater to 'moderate' agreement (Cohen's κ of 0.583). Results in Sec. 4 consequently employ the latter autorater. This simple grid overlay can be viewed as a form of VIPE that is evidently beneficial for video understanding.

I.4 RushHour autoraters

Solutions in *RushHour* are not necessarily unique (in some samples, obstacle cars can be moved independently in various orders). The task as formulated by Zeller et al. (2026) also does not penalize non-optimal solutions that, e.g., move cars back-and-forth or (unnecessarily) move distractor cars. Fortunately, the dataset comes with a validator that determines whether a given move sequence (e.g., `Car A forward, Car B backward, Car R forward`) solves a puzzle. The autoraters for this task are, thus, better understood as parsers that extract the move sequence from a video. While this task sounds trivial at first, many generated videos suffer from arbitrary perspective changes, non-continuous movements, duplicating or vanishing objects, etc.

VLM-based parser As in Sec. I.3, we first test a VLM-based autorater, asking Gemini 3.1 Pro to parse the move sequence from the video (prompt detailed below). Gemini only has to parse each car's initial movement (following Zeller et al. (2026)), and provides absolute movement directions on an eight-point compass rose, which we can map to

forward/backward based on each car's predefined axis of movement. This prompt, too, contains placeholders populated with the appropriate terms for the puzzle variant, e.g., using `anodized metal slides/deadbolts`, `metal slider`, `puzzle box backplate`, and `brass exit slot` for `objects`, `object_singular`, `container`, and `exit` for the `anodized_metal_puzzle_box` variant. We also experiment with a version of this parser for which each input video is overlaid with a static grid as in Sec. I.3, Fig. 20.

RushHour sequence parsing prompt

`[video]`

You are an expert video analyzer. Analyze the video which shows a puzzle being solved. The puzzle is a variation of *RushHour*. In this variation:

- The objects are `{objects}`.
- The board/container is `{container}`.
- The exit is `{exit}`.
- One of the objects is the target (usually red/pinkish, labeled 'R' or 'red_car'), which needs to reach the `{exit}`.

Your task is to extract the sequence of moves of the `{objects}` from the video and evaluate if the video remains valid. Only consider the part of the video from the beginning until the scene becomes invalid. A scene becomes invalid if:

- The layout of the `{container}` changes.
- `{objects}` appear or disappear.
- `{objects}` morph, deform, or change color.
- `{objects}` leave their defined tracks/axes.

For the valid part of the video, extract each `{object_singular}`'s first move. Each move should specify the object label and the absolute direction of movement on an 8-point compass rose (N, NE, E, SE, S, SW, W, NW), where N always points to the top of the frame. Format each move as a string, e.g., "A N", "B NW".

Output your response as a JSON object with the following fields:

1. "moves": A list of strings representing the extracted moves in order, e.g., ["A N", "B NW"]. If no moves occur before the scene becomes invalid, this should be [].
2. "invalid_after_seconds": A float representing the timestamp (in seconds) when the scene first became invalid. If the video remains fully valid and does not violate any rules until the end, set this to null.
3. "justification": A text explanation of your analysis. Explain why the video was marked valid/invalid (e.g., if an object morphed, specify which object and at what time), and describe the moves you observed.

Only output the JSON object, nothing else. Do not wrap it in markdown block.

Algorithmic parser The algorithmic parser uses computer vision techniques to extract the move sequence and validate video consistency. The parser first aligns each frame to the starting frame to compensate for camera movement. Initial car positions are masked out during alignment to avoid errors from moving cars. Based on the first frame, the parser initializes a tracker for each movable car by cropping the car's template from the first frame. In subsequent frames, the tracker attempts to match this template along the cars permissible motion axis, restricting the problem to a 1D search space. A move (`forward` or `backward`) is registered when the displacement along this axis exceeds 0.5 world units. The parser detects video invalidity using two main checks:

- **Object morphing**: If the template matching normalized cross-correlation (NCC) score falls below 0.6, the object is considered morphed or vanished, and the video is marked invalid. In *lenient* mode, tracking failures are ignored for cars that have already completed an initial move.
- **Background consistency**: The parser computes the absolute difference between the current frame and the starting frame in the (aligned) background region (excluding cars and rails). It applies an HSV-based shadow filter to ignore illumination changes caused by moving shadows. A non-shadow connected region of > 1000 pixels is interpreted as a structural background change, marking the video as invalid.

The extracted move sequence is verified logically against the puzzle's initial state, and the parser terminates early as soon as a valid solution sequence is found.

Table 12: Autorater-human agreement on 120 *RushHour* videos.

	VLM-based parser		Algorithmic parser	
Metric	**Original**	**With grid**	**Strict**	**Lenient**
Agreement	82.50%	80.83%	76.67%	87.50%
Cohen's κ	0.380	0.250	0.000	0.570

Human agreement We validate the candidate autoraters against human ratings on a sample of 120 videos from both *RushHour* levels (see Tab. 12). In contrast to Sec. I.3, we don't find the grid overlay to be beneficial in this task for Gemini's video understanding, possible because most cars' motion axes are not axis-aligned. Both Gemini-based raters achieve only 'fair' agreement (Cohen's $\kappa < 0.4$). The strict algorithmic parser fails entirely, since in the vast majority of generated videos, the first-moved car strays off its valid motion axis before the second car is moved. In the lenient case, where we only ensure that each car's initial move is in a valid direction and ignore it entirely afterwards (aligning with (Zeller et al., 2026)), the parser achieves 'moderate' agreement (Cohen's κ 0.570). We use this last parser for all experiments in Sec. 4.