

Appendix

We structure the supplement into the following subsections:

- A Details on best practices for video generation.
- B Overview of prompt and examples of video summaries with GPT responses used for video filtering.
- C Results and discussion on our method’s mesh-free object tracking version.
- D Details on reducing noise in 6D pose rollouts for stable and realistic motion.
- F Adaptation and implementation details of baseline methods.
- E Comprehensive example of Rekep Predictions and Execution.
- G Discussion of limitations of Tracking using point tracks.
- H Elaboration on our method’s robustness.
- I Thorough analysis of errors caused by depth estimation.
- J Discussion regarding the choice between the use of MegaPose and FoundationPose, focusing on trajectory stability.
- K Additional analysis of generated videos and human demos using VBench++ metrics.

We are attaching the source code in the supplementary materials for reproducibility.

A BEST PRACTICES FOR VIDEO GENERATION

We found that the following practices lead to reliable video generation: (1) having a clean background without visual distractions, (2) minimizing the number of distractor objects in the scene, (3) ensuring objects are reasonably large and viewed from a natural, human-like perspective, (4) ensuring there is one clearly identifiable task that can be performed, (5) using simple and concise text prompts, and (6) setting the relevance factor to 0.7 with the negative prompt “fast motion” led to the most reliable video generations.

B PROMPTING FOR VIDEO FILTERING AND FILTERING STATISTICS

The prompt for GPT o1-based filtering is shown in Figure 16. We provide GPT o1 with the prompt, a video summary—created by vertically concatenating evenly sampled frames from the video—and the language command (e.g., “pour water”). GPT o1 then responds with “Yes” or “No” to indicate whether the task is successfully performed.

C MESH-FREE OBJECT TRACKING

We experiment with a mesh-free object tracking version of our method. Specifically, we use BundleSDF (Wen et al., 2023a), which jointly performs 6-DoF object tracking and reconstruction from RGBD observations. For the *pouring* task, we evaluate our method using trajectories obtained via BundleSDF over 10 trials and observe a success rate of (90%), matching our default tracking setup. While the BundleSDF paper reports real-time capabilities, we found that its official implementation takes approximately 30 minutes to process each video in practice, which limits its applicability for real-time deployment. In contrast, our default tracker operates in real-time, enabling closed-loop execution and recovery from disturbances as discussed in Sec. 4.5. While the BundleSDF paper reports real-time capabilities, we observed significantly higher runtimes in practice with the official implementation. We expect that future advances in model-free tracking will address these efficiency bottlenecks, allowing for real-time mesh-free deployment.

D SMOOTHING OBJECT TRAJECTORIES

To reduce noise and jitter in the estimated object poses, we apply a moving average filter with a fixed sliding window (centered on each point) to the position and orientation components. Translations are smoothed independently along each axis, while orientation is processed similarly after converting from quaternions to rotation vectors. This approach mitigates abrupt changes, resulting in a more stable and realistic object trajectory with smoother transitions.

E REKEP PREDICTIONS AND EXECUTIONS

A detailed example of ReKep’s keypoint and VLM predictions for pouring task is shown in Fig. 17. The VLM first predicts grasping the watering can at keypoint 1. For the transport phase, it instructs moving keypoint 8 above keypoint 15, while keeping its height above keypoint 7. For the pouring action, keypoint 8 remains above 15 (to place the spout over the plant) and above keypoint 4 (to induce tilting). The resulting robot execution fails. We attribute most ReKep failures to inaccurate keypoint predictions, as shown in Fig. 11. In the lid image, no keypoint appears at the lid handle. In the placing task, keypoints cluster around pan corners. For the sweeping task, the keypoints are generally well-placed, and executions succeeded. Suboptimal initial keypoints lead to inaccurate downstream VLM predictions.

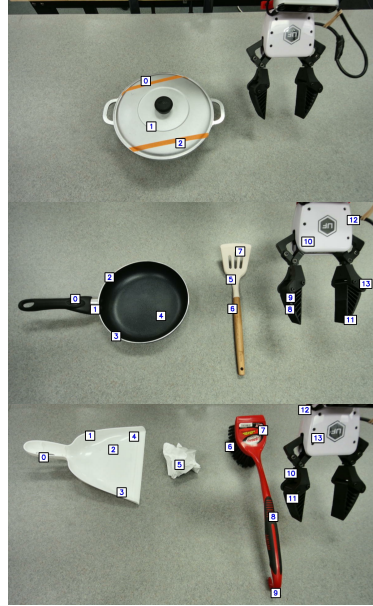


Figure 11: **Examples of ReKep’s Keypoint Locations.** The keypoint placements are often suboptimal, except for sweeping task, where the keypoints are reasonable.

F DESCRIPTION OF BASELINES

Track2Act (Bharadhwaj et al., 2024b): We adapt Track2Act’s procedure to our setup preserving its core idea of object-centric trajectory estimation from point tracks. Track2Act generates a future interaction plan by predicting 2D point trajectories (using a DiT-based diffusion model) between an initial image and a goal image, then recovers a sequence of 3D object transforms via Perspective-and-Point (PnP) (Zhang, 2000).

To integrate this into our pipeline, we use their published checkpoint but modify the input formulation—while the initial image remains identical to our real camera’s view, the goal image is taken from the last frame of a generated video rather than being physically captured. We then use PnP on the predicted point tracks along with the initial depth image to estimate the object’s rigid motion across frames, thereby defining the end-effector trajectory. We use interpolation between consecutive poses because Track2Act generates only a sparse set of frames, and denser sampling is needed for smooth trajectory estimation and execution. However, we exclude Track2Act’s closed-loop residual policy correction, focusing solely on open-loop 6D object-pose estimation and execution. This adaptation allows us to directly evaluate how well a vision-based, open-loop approach generalizes to our setting without additional corrections.

AVDC (Ko et al., 2023): The AVDC approach models action trajectories by synthesizing a task-driven video (using a trained text-conditioned video generation model) and using optical flow from GMFlow (Xu et al., 2022) to estimate dense pixel correspondences. It then reconstructs 3D object motion using an optimization step that refines pose estimates based on the tracked flow and depth information. To improve robustness, AVDC also includes a replanning mechanism that re-executes the pipeline when predicted motion stagnates.

Since the trained text-conditioned video generation model did not generalize well to our setup, we use the same generated video as in other experiments to ensure a fair comparison. While we do not employ AVDC’s replanning strategy, we predict object poses using a similar optimization framework based on flow and depth information.

4D-DPM (Kerr et al., 2024): 4D-DPM is designed to track 3D motion of articulated object parts from a single video. It constructs a 3D Gaussian splatting (Kerbl et al., 2023) representation of the scene to capture object features, then applies GARField (Kim et al., 2024) to cluster the Gaussians into discrete object components. In our adaptation, we modify this to operate on entire objects rather than individual parts. Specifically, we set the clustering parameters to treat the object as a

single entity, ensuring that motion estimation is performed at the object level rather than segmenting it into multiple parts. This allows us to track and execute trajectories for the whole object.

Gen2Act (Bharadhwaj et al., 2024a): Gen2Act introduces a video-conditioned policy learning framework that first generates a human video using a video generation model from a scene image and a task description. It then extracts object tracks using BootsTAP (Doersch et al., 2024), and trains a policy using behavior cloning with an auxiliary track prediction loss and offline robot demonstrations. At inference, Gen2Act uses the generated video and the learned policy to predict robot actions.

Our approach presents a simplified adaptation of this framework that removes the need for behavior cloning, and offline demonstrations. Instead of using the extracted tracks as an auxiliary loss, we directly process them for pose estimation. To recover 3D object positions, we leverage an initial depth image corresponding to the scene image, allowing us to obtain depth values for the extracted 2D tracks. We apply RANSAC filtering to remove outlier track points and then use the Perspective-n-Point (PnP) (Zhang, 2000) to estimate the object’s 6DoF pose. This adaptation preserves the core idea of leveraging video and track-based signals while eliminating the need for supervised policy learning.

G LIMITATION OF TRACKING WITH POINT TRACKS

All point tracks fail under extreme rotations, as initially visible points often become occluded. This is a fundamental limitation of any correspondence-based tracking method relying solely on visible surface features. We show this failure in Fig. 12. As the object rotates, most initial points are lost, resulting in insufficient 2D-3D correspondences to solve a stable PnP problem. This degrades pose estimation quality, leading to large drift or abrupt jumps in estimated object motion. Such instability cascades into execution errors, often causing the robot to fail the task altogether. As a result, both variants of Gen2Act—despite stronger tracking backbones like CoTracker—still fail under large out-of-plane rotations. In contrast, RIGVid’s model-based 6D tracking handles these situations more robustly, as it uses full-object geometry and SE(3) filtering to maintain stable trajectories.

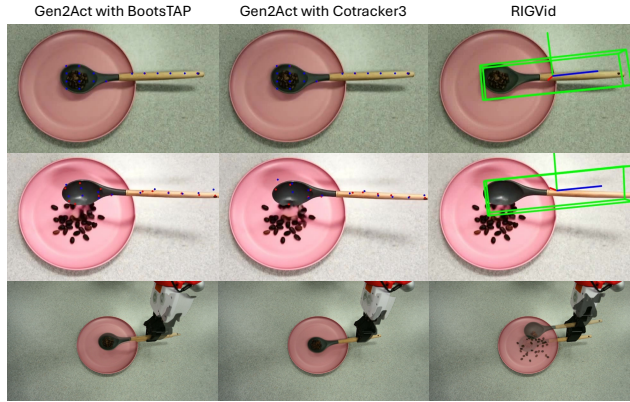


Figure 12: **Gen2Act with BootsTAP, CoTracker, and RIGVid.** Blue points denote the tracked points used for PnP; red points represent the reprojected 3D points. For a good PnP solution, these should align, as seen in the first frame. For Gen2Act, the blue points drift significantly from the red ones in later frames, indicating failure in pose estimation due to tracking loss, which leads to failed robot execution.

H ADDITIONAL ROBUSTNESS EXAMPLES

Examples of RIGVid’s robustness are shown in Fig. 13. In the first row, the robot grasps the object, but due to a misaligned grasp, the object rotates unexpectedly. The robot compensates by rotating it back to the correct orientation and then resumes the planned trajectory, completing the task successfully. In the bottom row, a human perturbs the object during execution while it is held by the robot. RIGVid detects the resulting change in the relative transformation and automatically re-aligns the object before continuing. When the human intervenes a second time, RIGVid again corrects the deviation, resulting in successful task completion.

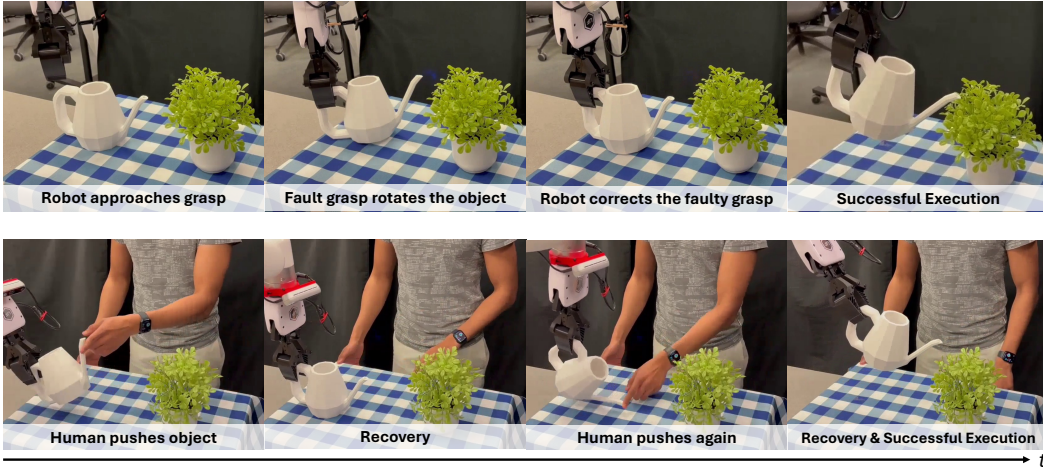


Figure 13: **Additional examples of RIGVid’s robustness.** In the top row, RIGVid recovers from a faulty initial grasp by reorienting the object before continuing execution. In the bottom row, it corrects for external disturbances on the object when a human pushes it mid-execution, realigning and successfully completing the task.

I ERRORS FROM DEPTH ESTIMATION

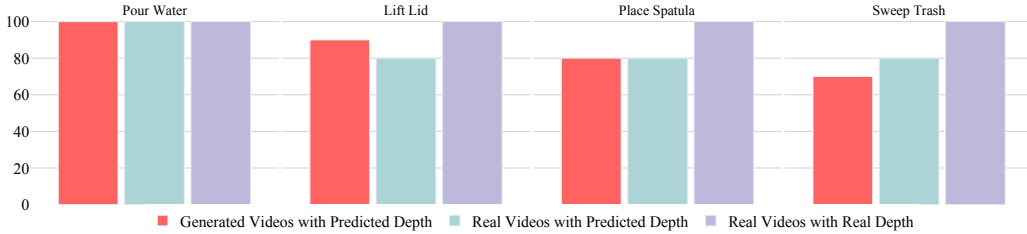
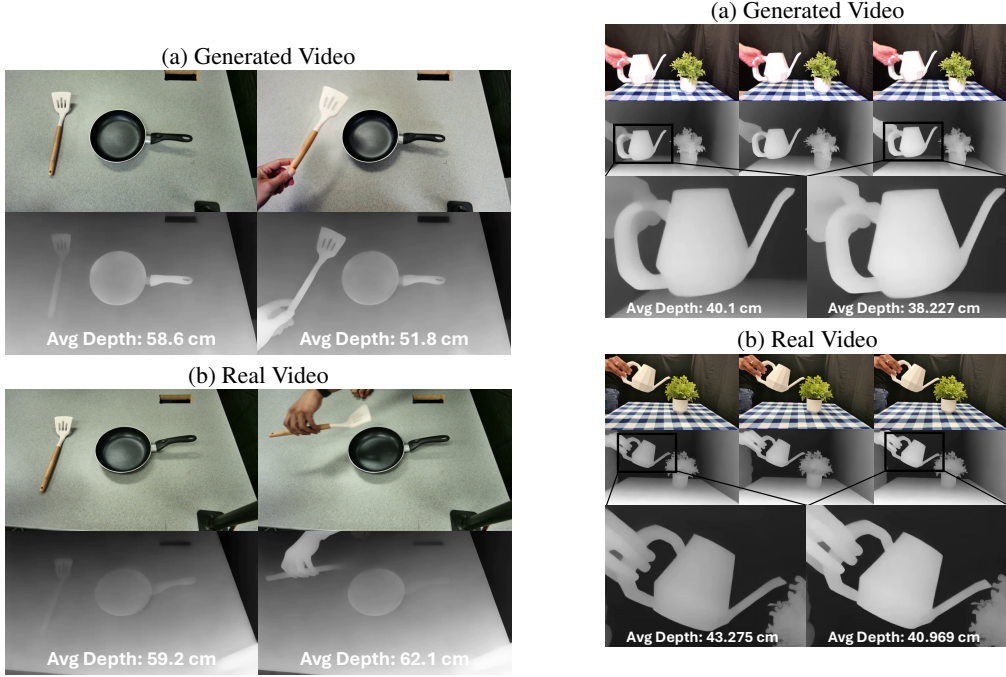


Figure 14: **Impact of Depth Estimation Errors on RIGVid performance.** Errors in monocular depth estimation result in worse performance of generated and real videos. RIGVid achieves perfect success across all tasks with real videos and real depth.

In Fig. 14, we isolate the impact of depth estimation errors. Robot executions on real videos with real depth (captured using an RGBD camera) achieve a 100% success rate, whereas executions from real videos with generated depth result in an 85% average success. Similarly, executions from Kling V1.6-generated videos with generated depth also achieve 85% success, suggesting that the primary source of error lies in monocular depth estimation. Upon inspection, we observe two common undesirable behaviors in the predicted depth: inaccurate depth values and temporal flickering. An example of inaccurate depth is shown in Fig. 15a. In the generated video, when the spatula is brought close to the camera, the depth changes by only 6.8 cm, which is visibly inconsistent with the video and likely much smaller than the real-world change. Inaccuracies also occur in real videos, as shown in the figure—the head of the spatula is estimated to be far from the camera, despite appearing close, revealing another failure mode in monocular depth estimation. Flickering is shown in Fig. 15b. Although the position of the watering can relative to the camera remains nearly unchanged across three consecutive frames, the estimated depth varies significantly. The zoomed-in region on the right shows the can appearing much whiter than on the left, indicating a substantial change in predicted depth. The average depth of the can changes from 40.1 cm to 38.2 cm—a 1.9 cm difference over just 0.066 seconds—which is physically implausible for the generated video. We find similar flickering behavior in real videos as well, where the depth changes from 43.2 cm to 40.9 cm in the given example—a 2.3 cm difference. Since errors in the generated depth are the main source of failure, we also tested removing it entirely by estimating object pose directly from the RGB frames of the



(a) **Errors in Monocular Depth Estimation.** In the generated video (top), the depth of the spatula changes only slightly despite a large visual change. In the real video (bottom), the spatula’s head is predicted to lie farther away, contradicting the visual appearance.

(b) **Flickering in Depth Prediction.** We show three consecutive frames of the video and its corresponding predicted depth. The depth of the watering can change noticeably across frames—appearing significantly whiter in the third frame despite minimal actual motion. We observe this behavior in both generated and real videos.

Figure 15: **Combined Figure:** Comparing depth estimation errors (left) and prediction flickering (right) in generated and real videos.

generated video using MegaPose. However, this approach leads to even more unstable and noisy trajectories, as detailed in App. J.

J CHOICE BETWEEN MEGAPOSE AND FOUNDATIONPOSE

We compare trajectory stability from MegaPose (Labbé et al., 2022) and FoundationPose (Wen et al., 2023b) by computing the translational and rotational RMS jitter. For each method, we apply a Gaussian smoothing filter ($\sigma = 2$ frames) to the raw SE(3) pose sequences, compute the residual between original and smoothed trajectories, and then calculate:

$$\text{jitter}_{\text{trans}} = \sqrt{\frac{1}{N} \sum_{t=1}^N \|\Delta \mathbf{t}_t\|^2}, \quad \text{jitter}_{\text{rot}} = \sqrt{\frac{1}{N} \sum_{t=1}^N \theta_t^2},$$

where $\Delta \mathbf{t}_t$ is the translational residual at frame t , and θ_t is the angular magnitude (in radians) of the relative rotation $R_{\text{smooth}}^{-1} R_{\text{raw}}$, converted to degrees. Metrics are averaged over ten pouring trajectories from generated videos.

MegaPose yields an average translational RMS jitter of 0.0045m and rotational RMS jitter of 37.47°, whereas FoundationPose achieves 0.0029m translational and 14.31° rotational jitter. This demonstrates that FoundationPose produces significantly smoother and more stable trajectories. Additionally, it allows for real-time tracking during the execution, making RIGVid robust to external disturbances.

K COMPARING VIDEO GENERATIVE MODELS

To further assess video quality, we report VBench++ (Huang et al., 2024b) metrics in Table 2 and explain them below. The numbers in the table are scaled $100\times$ for easier interpretation. We collect these metrics on 40 randomly selected and unfiltered videos per model, 10 for each of the four tasks. Kling v1.6 outperformed the other models on most metrics but performed similarly or worse in video-text consistency and dynamic degree. Human evaluations discussed in Sec. 4.2 suggest that the video-text consistency and I2V subject consistency are not reliable indicators of whether a generated video correctly follows a given command. Sora scored high on dynamic degree, likely due to its tendency to drastically alter the scene, resulting in exceptionally large motions. Generated videos from these models and their corresponding metrics are shown in Fig. 18 and further details on these metrics can be found the next section.

VBench++ Metric Definitions:

- **Subject Consistency.** Subject consistency describes whether subjects’ appearance remain consistent, which is computed by DINOv1 (Caron et al., 2021) similarities across video frames.
- **Background Consistency.** Background temporal consistency by CLIP (Radford et al., 2021) similarities across frames.
- **Motion Smoothness.** Evaluates smoothness of videos by utilizing video frame interpolation model AMT (Li et al., 2023).
- **Dynamic Degree.** Describes whether the video contains large motions as a binary metric.
- **Aesthetic Quality.** Human perceived artistic and beauty value such as photo-realism, layout and color harmony.
- **Imaging Quality.** Assesses the presence of distortion in a video, such as noisiness, blurriness, and over-exposure.
- **Video-Text Consistency.** Text-to-video alignment score calculated by ViCLIP (Wang et al., 2023).
- **I2V Subject Consistency.** Similarity between subjects in input image and each video frame, as well as similarity between consecutive frames. Features are extracted from DINOv1 (Caron et al., 2021).

Metrics	Video Generation Models			Human Demos
	Kling V1.6	Kling V1.5	Sora	
Subject Consistency	96.34	91.66	83.09	94.91
Background Consistency	96.64	93.97	89.34	95.00
Motion Smoothness	99.68	99.57	99.06	99.51
Dynamic Degree	52.5	57.5	70.0	80.0
Aesthetic Quality	51.75	49.77	46.22	49.30
Imaging Quality	72.80	71.48	68.68	72.52
Video-Text Consistency	22.01	22.61	21.42	21.57
I2V Subject Consistency	97.88	95.96	89.09	97.89

Table 2: **Video generation quality metrics for real human demonstration videos and different models.** Higher values indicate better quality. Kling v1.6 performs comparably to or surpasses other models on most metrics.

Prompt

```
## Instructions
You need to determine whether a specific action occurs in the
given image.
### Image Format
- The image has four rows.
- The action progresses from top to bottom.
### Task
- Examine the image and the provided query.
- Decide if the action described in the query is clearly
  depicted.
- A hand should be performing the action. Without it, the
  actions are likely implausible as the action needs to be
  executed by an actor.
- The task has to be done with the specified object in the first
  frame. The video cannot bring in a new object such that there
  are two of the same objects.
### Output Format
Your response must be in the following format
Explain if we see a hand.
Explain if a new object has been brought in that is different
from any objects in the first frame
Explain your reasoning for the given image.
action_done: Yes or No
## Query
Query Task: Pour water from the watering can on the plant
Query Image:
```



GPT o1 Response

We do see a hand holding and tilting the same watering can from the first frame. Water is visibly flowing onto the plant in the final frame.
action_done: Yes

Prompt

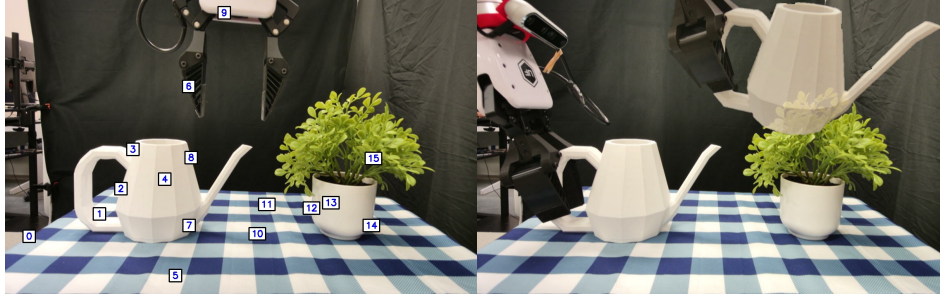
```
## Instructions
Same as previous prompt. Instructions omitted for simplicity.
## Query
Query Task: Place the spatula on the pan
Query Image:
```



GPT o1 Response

1) Hand present? Yes, a hand is visible holding the spatula.
2) New object introduced? No, there is no new object introduced; it is still the original spatula and the original pan.
3) Reasoning: Although we see a hand holding the spatula, there is no point at which the spatula is placed on or in the pan. The spatula instead ends up resting on the table next to the pan.
action_done: No

Figure 16: Examples of prompting GPT o1 to filter generated videos. We sample frames from the generated video and prompt GPT o1 to assess whether the specified task is performed successfully in the video. The top example passes the filtering, while the bottom does not.



```
def stage1_subgoal_constraint1(end_effector, keypoints):
    """Align the end-effector with the watering can handle (keypoint 1)."""
    handle_point = keypoints[1]
    cost = np.linalg.norm(end_effector - handle_point)
    return cost
```

```
def stage2_path_constraint1(end_effector, keypoints):
    """The robot must still be grasping the watering can handle (keypoint 1)."""
    return get_grasping_cost_by_keypoint_idx(1)

def stage2_path_constraint2(end_effector, keypoints):
    """The watering can must stay upright to avoid spilling (spout above base)."""
    spout = keypoints[8]
    base = keypoints[7]
    # Ensure spout is not lower than base
    cost = max(0, spout[2] - base[2])
    return cost
```

```
def stage2_subgoal_constraint1(end_effector, keypoints):
    """The watering can spout (keypoint 8) needs to be 10cm above the
    center of the plant (keypoint 15)."""
    spout = keypoints[8]
    plant = keypoints[15]
    offsetted_point = plant + np.array([0, 0, 0.10]) # 10cm above the plant
    cost = np.linalg.norm(spout - offsetted_point)
    return cost
```

```
def stage3_path_constraint1(end_effector, keypoints):
    """The robot must still be grasping the watering can handle (keypoint 1)."""
    return get_grasping_cost_by_keypoint_idx(1)

def stage3_path_constraint2(end_effector, keypoints):
    """The watering can spout (keypoint 8) is directly above the plant (keypoint 15)
    in x-y plane."""
    spout = keypoints[8]
    plant = keypoints[15]
    cost = np.linalg.norm(spout[:2] - plant[:2])
    return cost
```

```
def stage3_subgoal_constraint1(end_effector, keypoints):
    """The watering can spout (keypoint 8) needs to be 10cm above the
    center of the plant (keypoint 15)."""
    spout = keypoints[8]
    plant = keypoints[15]
    offsetted_point = plant + np.array([0, 0, 0.1]) # 10cm above the plant
    cost = np.linalg.norm(spout - offsetted_point)
    return cost

def stage3_subgoal_constraint2(end_effector, keypoints):
    """The watering can spout (keypoint 8) must be tilted below the base (keypoint 4) to pour."""
    spout = keypoints[8]
    base = keypoints[4]
    # Ensure spout is lower than base
    cost = max(0, spout[2] - base[2])
    return cost
```

Figure 17: **ReKep’s output for the pouring task and the resulting robot execution (top-right).** The VLM predicts to grasp at keypoint 1, move keypoint 8 above 15 and 7 during transport, and above 15 and 4 for pouring—leading to failed execution.

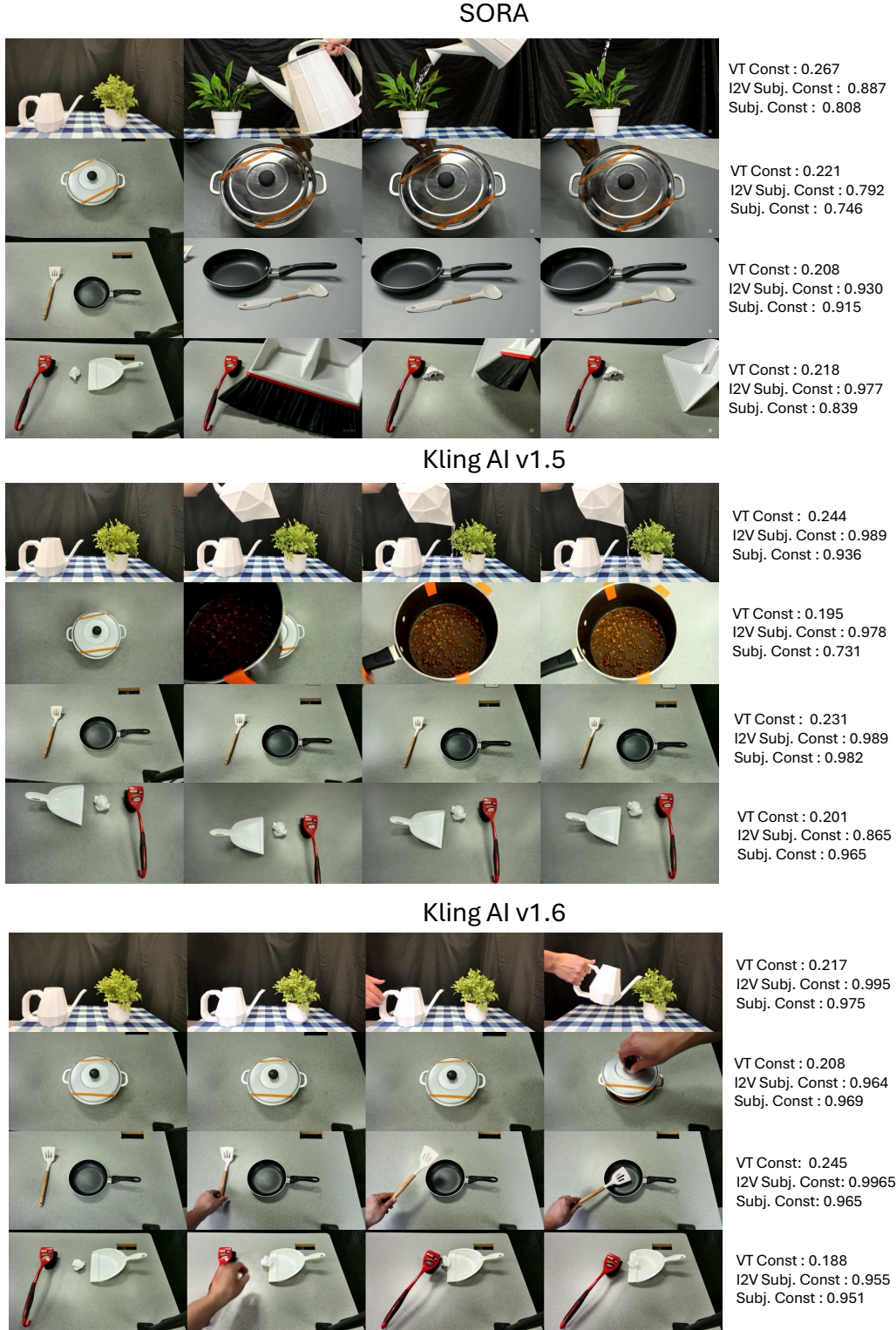


Figure 18: Qualitative Comparison of Different Video Generative Models. Videos from the three video generation models are shown using evenly sampled frames, along with VBench++ (Huang et al., 2024b) metrics: video-text consistency, image-to-video subject consistency, and subject consistency. Kling v1.6 scores highest on these metrics, followed by Kling v1.5 and then Sora.

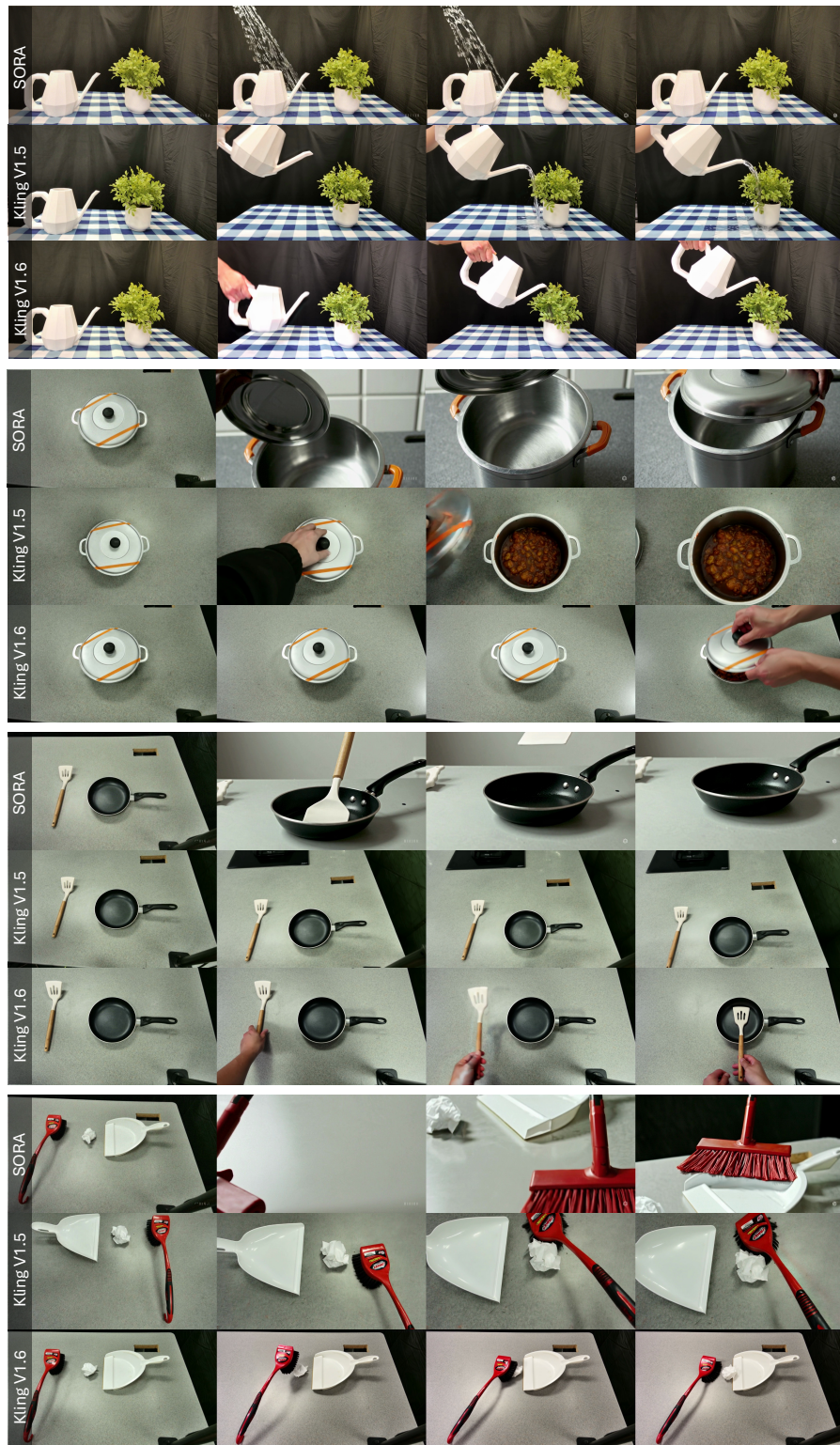


Figure 19: **Qualitative comparison of video generation.** Sora-generated videos often alter the scene layout and objects. Kling V1.5 produces more plausible results but includes physically implausible elements. Kling V1.6 better preserves scene fidelity and closely follows the human command.