**Algorithm 1:** Adapting World Models from VLM Plans for Robotic Tool Use

**Input:** initial observation $\mathbf{o}_1$, task description $\mathcal{L}$, goal metric $\mathcal{G}$, VLM $\mathcal{V}$, reward model $R$, iterations $X$, batch size $B$, environment executions per iteration $K$, test batch size $N$

**Output:** plan $P = (\mathbf{a}_1, \ldots, \mathbf{a}_n)$

Initialize ensemble of world models $E \leftarrow (f_{\theta_1}, \ldots, f_{\theta_e})$ using pretrained weights

$\mathcal{D} \leftarrow$ `collect_vlm_demos()` ▷ Collect small initial dataset by proposing and executing plans

Seed $E$ with different random seeds and subsets of $\mathcal{D}$

**for** $i \leftarrow 1$ **to** $X$ **do**

  $\mathbf{P} \leftarrow$ `propose_plans`$(\mathcal{V}, B)$

  $\mathbf{P}_{\text{best}} \leftarrow$ `rollout_and_score_plans`$(\mathbf{P}, K, E, R)$

  $\mathcal{D}_{\text{self}} \leftarrow$ `execute`$(\mathbf{P}_{\text{best}}, \mathbf{o}_1, \mathcal{L}, \mathcal{G})$

  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{\text{self}}$

  Fine-tune the world models $E$ with $\mathcal{D}$

**end**

Propose $N$ VLM plans

$P_{\text{best}} \leftarrow$ `rollout_and_score_plans`$(P, 1, E, r)$

**return** $P_{\text{best}}$

# APPENDIX

## VIII. Method Algorithm

Alg. 1 summarizes our framework for adapting an action-conditioned world model to an unseen tool-use task using VLM-proposed plans and their executed outcomes.

## IX. VLM Planner

Given an initial RGB image of the scene $\mathbf{o}_1$ and language instruction $\mathcal{L}$ describing the goal, the VLM $\mathcal{V}$ outputs a sequence of gripper actions $(\mathbf{a}_1, \ldots, \mathbf{a}_t)$. Each gripper action $\mathbf{a}_i$ specifies a 3D gripper keypoint position $\mathbf{p}_i = (x_i, y_i, z_i)$, and binary gripper openness $g_i \in \{0, 1\}$ such that $\mathbf{a}_i$ denotes moving the gripper to the next keypoint $\mathbf{p}_i$, followed by changing the gripper state to $g_i$.

We prompt the VLM to specify each gripper action $\mathbf{a}_i$ as a tuple $(\mathbf{u}_i, \Delta\mathbf{x}_i)$, where $\mathbf{u}_i = (u_i, v_i)$ are normalized image-plane coordinates and $\Delta\mathbf{x}_i = (\Delta x_i, \Delta y_i, \Delta z_i)$ is a 3D offset. Then we combine $\mathbf{u}_i$ and $\Delta\mathbf{x}_i$ to obtain the final gripper keypoint position $\mathbf{p}_i$ in 3D world coordinates. The full VLM prompt we use (Fig. 8) is adapted from prior work [52].

For each of the tasks we evaluate in this work, it is enough to assume a fixed gripper orientation throughout the task. We could, however, feasibly extend the prompt with gripper rotation actions to allow more types of motion.

To execute a VLM-generated plan, we control the robot end-effector toward each proposed gripper keypoint $\mathbf{p}_i$ in sequence. In simulation [53], we use joint interpolation, while in real world, we use motion planning [49].

## X. Action-Conditioned Generative World Model

Given a gripper action $\mathbf{a}_i$ specified as image coordinate $\mathbf{u}_i$ and offset $\Delta x$, we obtain the intended gripper keypoint $\mathbf{p}_i$ by backprojecting $\mathbf{u}_i$ into world coordinates using the depth image of the initial scene and then adding $\Delta\mathbf{x}_i$. Intuitively, the VLM uses $\mathbf{u}_i$ to "anchor" on a specific point on an object in the scene and then adds a 3D offset $\Delta\mathbf{x}_i$ (Fig. 3).

Inspired by prior works [44], [45], [52], we condition the world model on VLM-proposed actions by "drawing" the action as a line segment directly onto the input image of the current scene. To condition on action $\mathbf{a}_{i+1}$, meaning the gripper should move from current 3D keypoint $\mathbf{p}_i$ to proposed gripper keypoint $\mathbf{p}_{i+1}$, we project $(\mathbf{p}_i, \mathbf{p}_{i+1})$ into image coordinates $(\mathbf{u}_i, \mathbf{u}_{i+1})$ and then draw the 2D line segment connecting $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$ onto the current image observation $\mathbf{o}_i$ to get image $\tilde{\mathbf{o}}_i$. Given $\tilde{\mathbf{o}}_i$, the world model predicts the next observation, denoted $f_\theta(\tilde{\mathbf{o}}_i, \mathcal{L}) = \hat{\mathbf{o}}_{i+1}$.

A gradient in the red channel indicates temporal progression [52]. A colored circle indicates the target gripper state: green for closed and blue for open. The circle appears for every action, including actions that preserve the gripper state. See Fig. 9 for examples.

## XI. World Model Adaptation and Plan Selection

Using the VLM planner and prior world model, we iteratively learn and align the world model to the previously-unseen tool manipulation task (training time) and then use the world model to select the best plan (test time). In both phases, we use an uncertainty estimate from the world model to guide decision-making.

At training time, we collect demonstrations by executing some VLM-proposed plans and adding the observed outcomes to a data buffer. Since executions are expensive, we want to learn efficiently by selecting plans that result in the most informative trajectories. To do so, we consider two axes:

(1) We want to improve the world model's predictions for states and actions where it is currently uncertain, so it can rapidly learn what it does not know.

(2) We want to learn from demonstrations that achieve high expected reward, so we need a way to accurately predict task success from the observed trajectories.

### A. DECU-Inspired Ensemble Disagreement

We use disagreement among $e = 5$ world models as a proxy for epistemic uncertainty, following the shared-branch construction of Diffusion Ensembles for Capturing Uncertainty (DECU) [46]. For each action transition, we sample an initial diffusion latent and use the first ensemble member as a fixed reference to denoise it up to branch index $b$. We then evaluate every member at the same latent $x_b$, diffusion timestep, and conditioning input $c$, comprising the action-marked image and task instruction. The index $b$ is zero-based in the inference schedule: $b = 10$ means that ten denoising steps have completed and disagreement is evaluated before the eleventh update.

**3D Trajectory Prompt**

In the image, please execute the command described in '`{goal}`'.
Provide a sequence of keypoints denoting a trajectory of a robot gripper to achieve the goal. Keep in mind these are keypoints, so you do not need to provide too many points.
This image represents a 3D scene, so provide each keypoint as a 2D coordinate of some anchor point on an object's surface and a 3D offset relative to that anchor point, so that we can combine them to get the 3D position of each keypoint.
Format your answer as a list of tuples enclosed by `<ans>` and `</ans>` tags. For example:
`<ans>[((0.25,0.32),(0,0,0.05)), ((0.32,0.17),(0,0,0.05)), ((0.13,0.24),(0,0,0.05)), <action>Open Gripper</action>, ((0.74,0.21),(0,0,0.05)), <action>Close Gripper</action>, ...]</ans>`
The tuple denotes point $(x, y)$ in the image and $(dx, dy, dz)$ offset relative to that point. The action tags indicate gripper actions.
The anchor coordinates should be floats ranging between 0 and 1, indicating the relative locations of the points in the image. Make sure that each 2D anchor point is on the surface of some object, not in the air.
The offsets should be expressed in meters in world frame, where $x$ is pointing from the robot base toward the left side of the table, $y$ is pointing toward the front of the table, and $z$ is pointing upward.
Remember to provide points between `<ans>` and `</ans>` tags and think step by step.

Fig. 8: The full text prompt used for 3D trajectory prediction, adapted from prior work [52].

*a) Reverse-step distributions:* We use the Euler ancestral scheduler with epsilon prediction. Let $s$ and $s_{\text{next}}$ denote its current and next noise levels at the branch. The ancestral noise variance and deterministic target noise level are

$$s_{\text{up}}^2 = \frac{s_{\text{next}}^2(s^2 - s_{\text{next}}^2)}{s^2}, \qquad s_{\text{down}} = \sqrt{s_{\text{next}}^2 - s_{\text{up}}^2}.$$

For member $i$, let $\epsilon_i$ be its guided noise prediction at this shared input. Its reverse-step distribution is

$$p_i = \mathcal{N}(\mu_i, s_{\text{up}}^2 I), \qquad \mu_i = x_b + (s_{\text{down}} - s)\epsilon_i.$$

We compare these means before sampling fresh ancestral noise. The pairwise distance computed by the implementation is

$$d_{ij} = \frac{\|\mu_i - \mu_j\|_2^2}{2\max(s_{\text{up}}^2, 10^{-12})}. \tag{4}$$

The squared norm sums over all latent dimensions. When the variance floor is inactive, $d_{ij} = D_{\text{KL}}(p_i \| p_j)$ for these equal-covariance Gaussians; otherwise it is a variance-regularized distance.

*b) Pairwise and trajectory aggregation:* Our implementation averages $d_{ij}$ over the distinct unordered pairs of ensemble members. For action transition $h$ and a trajectory with $H$ transitions, the stepwise and trajectory-level scores are

$$q_h = \frac{2}{e(e-1)} \sum_{i<j} d_{ij}^{(h)}, \qquad U(\tau) = \frac{1}{H} \sum_{h=1}^{H} q_h,$$

respectively. The reference member completes each image prediction, which supplies the shared conditioning image for the next action transition in this uncertainty rollout. We use branch index $b = 10$, 20 denoising steps, text guidance scale 7.0, and image guidance scale 1.5.

This raw mean pairwise KL score differs from DECU's Pairwise-Distance Estimator (PaiDE), which applies a log-sum-exp aggregation.

*c) Normalization:* To combine disagreement with task reward (Sec. XI-B), we use

$$\bar{u}(\tau) = \frac{U(\tau)}{U(\tau) + C} - \frac{1}{2}.$$

We set $C = 25$, the median raw disagreement in an initial calibration batch. This monotone transformation maps nonnegative disagreement to $[-0.5, 0.5)$, with $\bar{u}(\tau) = 0$ when $U(\tau) = C$. We use the resulting score for both data collection and test-time selection; it is an ensemble-disagreement proxy, not a calibrated probability of execution failure.

### B. Reward Model

To accurately predict task success from the observed trajectories, we assume access to a goal-conditioned reward model $R$ that maps an image rollout $\tau = (\mathbf{o}_1, \ldots, \mathbf{o}_t)$ and language instruction $\mathcal{L}$ to a scalar reward $r = R(\tau, \mathcal{L}) \in [0, 1]$. Reward model learning is not our focus, so we fine-tune Robometer [14] on 100 demonstrations of each task. However, in the future, we expect zero-shot reward models to become increasingly better, thus minimizing the need for any fine-tuning.

We follow the provided instructions for LoRA fine-tuning Robometer on custom datasets and use the same hyperparameters. In the LoRA fine-tuning setup, there are $4{,}513{,}065{,}228$ parameters in total, of which $75{,}920{,}140$ $(1.68\%)$ are trained.

## XII. Method Hyperparameters

Table II summarizes the default hyperparameters used by our method. Unless otherwise stated, these settings are shared across tasks.

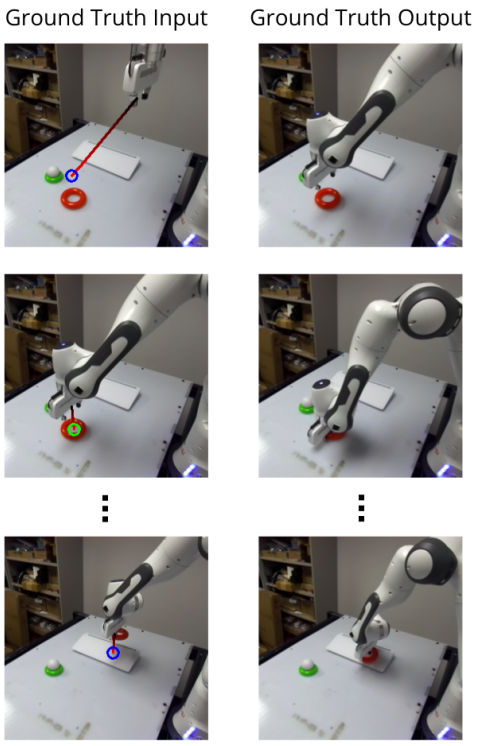



Fig. 9: **Example ground-truth world model inputs and outputs**. We condition the world model on VLM-proposed actions by drawing the action as a line segment onto the current image observation. Given this trace-drawn image and a language description of the task, the world model learns to predict the next observation.

*a) World model ensemble:* Our method maintains an ensemble of five action-conditioned image diffusion world models. Each ensemble member is initialized from the same SuSIE [42] U-Net checkpoint.

We have a hyperparameter for how many steps to train the world model ensemble for; we schedule this to be 5 for the first iteration and $400 \times i$ for subsequent iteration $i$. We also have a hyperparameter for how many transitions to sample; we schedule this to be 32 initially, and then we increase it to 100. We initialize training of each ensemble member with different random seeds.

All world model fine-tuning uses images of resolution $256 \times 256$, batch size 4, gradient accumulation of 4, learning rate $5 \times 10^{-5}$, no learning-rate warmup, gradient clipping at 1.0, and conditioning dropout probability 0.05.

*b) Uncertainty-Guided Learning:* We run four learning iterations by default. At each iteration, the VLM proposes $B = 50$ candidate 3D plans using up to 10 parallel query workers. Each candidate trajectory is rolled out with every ensemble member. We score the imagined rollouts $\tau$ using our fine-tuned Robometer [14] model as the reward model, defining $R(\tau, \mathcal{L})$ as the mean Robometer progress score across ensemble members. We compute DECU-inspired ensemble disagreement (see App. XI-A) over the current world model ensemble before it is updated with the new executions from that iteration. Candidate plans are ranked by

$$s(\tau, \mathcal{L}) = R(\tau, \mathcal{L}) + \beta \bar{u}(\tau),$$

where $\bar{u}(\tau)$ is the uncertainty estimate, the magnitude of $\beta$ controls the relative weighting between the two terms, and the sign of $\beta$ determines whether uncertainty is encouraged or discouraged. During learning, we set $\beta = 0.5$, so selection favors plans that are both predicted to make task progress and uncertain under the current ensemble. We execute the top $K = 10$ selected plans in each iteration.

*c) World model update schedule:* After each data collection iteration, all five ensemble members are warm-started from the previous iteration and fine-tuned on the full accumulated dataset. Learning iteration $n$ trains each ensemble member to $400n$ total gradient steps. For the Flatten Dough experiments, we use a shorter update schedule with 100 world model update steps per iteration.

*d) Online evaluation:* After each learning iteration, we run 10 online evaluation rounds. In each round, the VLM proposes $N = 15$ candidate plans. We use the same ensemble-mean Robometer reward and ensemble-disagreement computation as in the training phase but set $\beta = -0.5$ in the scoring function. This discourages high-uncertainty plans at deployment, favoring plans with high predicted reward and low ensemble disagreement.

TABLE II: **Default hyperparameters for our method.**

| **Component** | **Setting** |
|---|---|
| World model ensemble size | 5 |
| Image resolution | $256 \times 256$ |
| Fine-tuning batch size | 4 |
| Gradient accumulation | 4 |
| Learning rate | $5 \times 10^{-5}$ |
| Learning-rate warmup | None |
| Gradient clipping | 1.0 |
| Conditioning dropout | 0.05 |
| Learning iterations | 4 |
| VLM candidates per iteration | $B = 50$ |
| Executed plans per iteration | $K = 10$ |
| DECU [46] PaiDE branch step | 10 |
| World model denoising steps | 20 |
| Text guidance scale | 7.0 |
| Image guidance scale | 1.5 |
| Uncertainty weight | $\beta_1 = 0.5$ (train) , $\beta_2 = -0.5$ (test) |
| Uncertainty normalization constant | $C = 25$ |
| Test rounds per iteration | 10 |
| VLM candidates per test round | $N = 15$ |

## XIII. Experimental Setup

### A. Task Details

Fig. 10 depicts the setups for each of our tasks. The following are the task descriptions and goal metric success criteria for each task:

*a) Flatten Dough:*

- **Task Description:** "flatten the dough to a height smaller than 0.03"
- **Metric:** `1 - max(0, (dough.height-0.03)/(height0-0.03))`, where `height0` is the initial height of the dough
- **Success threshold:** $\mathcal{G}(\mathbf{s}) > 0.8$

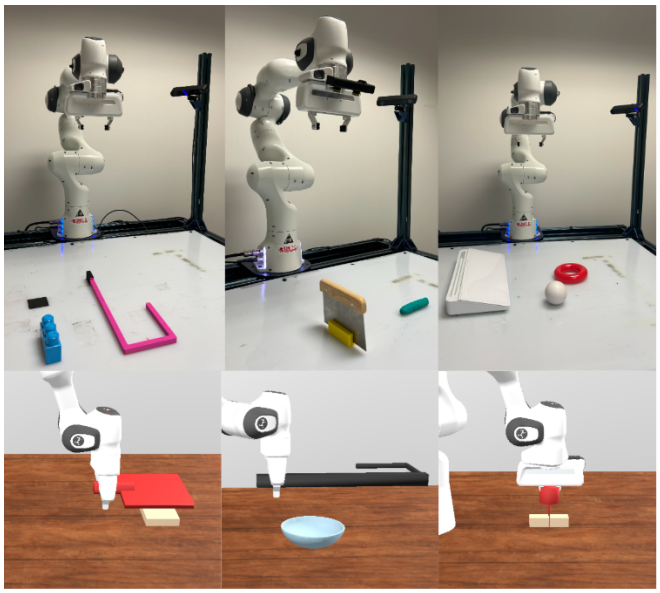


Fig. 10: **Task setups**. From left to right and top to bottom: Reach, Cut Dough (real world), Ball in Ring, Flatten Dough, Lift Bowl, Cut Dough.

*b) Lift Bowl:*

- **Task Description:** "lift up the blue bowl without touching its inner surface (the gripper is not wide enough to span the bowl)"
- **Metric:** `1 - max(0, (0.07 - bowl.pos[2])/0.1)` AND the bowl is within 30° from upright, i.e., if the bowl is tilted more than 30° from upright, then the metric is 0
- **Success threshold:** $\mathcal{G}(\mathbf{s}) > 0.8$

*c) Cut Dough (simulated):*

- **Task Description:** "cut the dough into two even parts and separate them by 0.2 meters"
- **Metric:** First, we compute `dough1, dough2 = KMeans(n_clusters=2).fit(dough).` Then the metric is `1 - max(0, (0.2-(dough1.pos-dough2.pos).norm()))/0.2`
- **Success threshold:** $\mathcal{G}(\mathbf{s}) > 0.8$

*d) Ball in Ring:*

- **Task Description:** "place the ball stably on the slanted surface"
- **Metric / success threshold:** success is judged by a human evaluator

*e) Cut Dough (real-world):*

- **Task Description:** "cut the dough into two even parts and separate them by 0.2 meters"
- **Metric / success threshold:** success is judged by a human evaluator

*f) Reach:*

- **Task Description:** "bring the lego brick that is outside of the robot arm's reachability to the black target location"
- **Metric / success threshold:** success is judged by a human evaluator

For real world tasks, a human evaluator determines execution success by examining the final state after robot execution. Prior to starting experiments, the evaluators agreed on what optimal, suboptimal, and failed examples look like.

## XIV. Baseline Implementation Details

We provide implementation details for the baseline methods used in our experiments. For all self-improvement baselines, we use the ground-truth simulator task metric to filter training data collected during learning. A rollout is added to the training buffer if its terminal task metric is greater than 0.8.

### A. Behavior Cloning Improvement Loop (BCIL)

We adapt BCIL [13] to our setting by bootstrapping it with successful VLM executions. Specifically, we execute some VLM-generated plans, retain the resulting trajectories whose task metric exceeds 0.8, and train an initial Diffusion Policy (DP) [51] on this seed dataset. The subsequent BCIL iterations are policy-driven: in each iteration, we roll out the current DP policy, retain successful rollouts, and fine-tune DP on the expanded success buffer. The number of initial VLM-generated trajectories we use for each simulated task is 80, 200, and 100 for Flatten Dough, Lift Bowl, and Cut Dough, respectively.

We use image observations of size $256 \times 256$ collected at 10Hz and a 10-dimensional absolute action representation (3D gripper position + 6D rotation + 1D gripper openness). The policy uses the default DP horizon of 16, with two observation steps and eight action steps, a ResNet-18 visual encoder, DDPM noise scheduling with 100 diffusion steps, AdamW with learning rate $10^{-4}$, and exponential moving average (EMA) weights. The initial policy is trained from scratch for 1000 epochs on the successful seed trajectories. We then run 10 BCIL iterations. Each iteration collects 30 DP rollouts, with a maximum of six high-level control steps per rollout. Successful rollouts are appended to the cumulative buffer, and the policy is warm-start fine-tuned for 50 epochs on the updated buffer.

### B. BCIL trained on the same data as our method (BCIL-Data)

BCIL-Data controls for the data collected by our method. It uses the same DP architecture, action representation, optimizer, and training schedule as BCIL. It is also initialized in the same way, by training DP on successful trajectories from the initial set of executed VLM-generated plans. Unlike BCIL, however, its subsequent training data are not collected from DP rollouts. Instead, at each learning iteration, BCIL-Data uses the exact VLM-generated plans executed by our method, filters the resulting executions using the same terminal metric threshold of 0.8, and fine-tunes DP on the expanded set of successful trajectories.

As in BCIL, the initial policy is trained for 1000 epochs, and each subsequent warm-start fine-tuning stage uses 50 epochs. Evaluation uses 30 policy rollouts, with a maximum of six high-level control steps per rollout.

TABLE III: After filtering for successful demonstrations (task metric $> 0.8$), these were the numbers of trajectories available for training the baseline on each task.

| | Baseline | | |
|---|---|---|---|
| Task | BCIL | VM+IDM | BCIL-Data |
| Flatten Dough | 20 | 20 | 16 |
| Lift Bowl | 36 | 25 | 16 |
| Cut Dough | 3 | 3 | 1 |

### C. Video Model + Inverse Dynamics Model (VM+IDM)

VM+IDM is a model-based imitation baseline that combines a video prediction model with an inverse dynamics model (IDM). The video model is trained on $256 \times 256$ image keyframe pairs, conditioning on the current frame and predicting one future latent frame. The IDM maps the current frame and predicted next frame to a single 4-dimensional action (3D gripper position + 1D gripper openness). The gripper orientation is fixed per task.

We initialize VM+IDM by executing 100 random VLM-generated plans and retaining the successful episodes according to the same metric threshold of $0.8$. The video model is pretrained for $70{,}000$ gradient steps with image size $256$, batch size $4$, and learning rate $10^{-6}$. The IDM is pretrained for $1000$ epochs with batch size $128$ and learning rate $10^{-4}$.

After pretraining, VM+IDM runs five self-improvement iterations. In each iteration, the current VM+IDM policy is rolled out in a closed-loop manner. At each replanning step, the video model predicts a next keyframe from the current observation, and the IDM converts the current and predicted frames into one action. We control the robot toward the predicted end-effector pose using inverse kinematics and joint interpolation, in the same way that we do for executing VLM-proposed plans in our main method.

Planning uses $50$ inference steps, guidance scale $2.5$, prior strength $0.5$, no internet video prior, and a maximum of eight replanning cycles. Each action is executed with $50$ low-level motion steps. Each self-improvement iteration collects $10$ rollouts. Rollouts whose terminal metric exceeds $0.8$ are appended to the training buffer. The video model is then fine-tuned for $10{,}000$ gradient steps, and the IDM is fine-tuned for $30$ epochs using the newly accepted demonstrations, with batch size $30$ and learning rate $10^{-4}$.

### D. Rollout budgets

Per learning iteration, BCIL is allowed $30$ rollouts, and VM+IDM is allowed $10$. BCIL-Data does not collect additional data from its own policy rollouts during learning; instead, it uses the same VLM-generated plans executed by our method, which isolates whether the gains come from the collected data alone or from using the adapted world model to evaluate and select plans.

### E. Number of successful trajectories available for training

The number of successful trajectories available for training the self-improvement baselines on each task are shown in Table III.

## XV. Results

We present full BCIL baseline results in Fig. 11.

### A. Qualitative Behavior and Failure Modes

The generated rollouts in Fig. 12 show that the learned world model captures task-relevant visual changes, such as moving the ball toward the ring, deforming or cutting dough, and lifting the bowl. These rollouts need not be photorealistic to be useful; they must be accurate enough for the reward model and uncertainty estimate to rank candidate plans.

Failure can occur if the world model predicts an overly optimistic outcome for a physically incorrect action. For example, Fig. 7 shows a plan whose grasp misses the scraper handle, causing the execution to diverge from the imagined rollout. This failure mode reinforces the main limitation of our approach: the world model can select among proposed plans, but it does not synthesize new actions outside the VLM-generated candidate set.

### B. Inference Times

We report approximate inference times for the main components of our method. Querying Gemini 3.1 Pro for VLM plans takes about 1–3 minutes per query, and we set an API timeout of $5$ minutes. We use up to $10$ parallel query workers whenever sampling candidate plans, including the $B = 50$ candidates used during each learning iteration.

Candidate scoring consists of world-model rollout, Robometer reward prediction, and DECU-inspired disagreement estimation. A single world-model generation takes about 0.58 seconds and can be batched, but multi-step plan evaluation is sequential over the actions in the plan. Across tasks, VLM plans contain approximately 4–11 actions. After generating a rollout, one Robometer inference call on the image sequence takes about 0.6 seconds. Disagreement is computed per action transition; one such computation takes about 0.75 seconds, including rollout to diffusion branch step $b = 10$ and mean pairwise-disagreement computation over the $e = 5$ ensemble members.

At test time, our method evaluates $N = 15$ candidate VLM plans by computing world-model rollouts, Robometer rewards, and ensemble disagreement for each plan before executing the highest-scoring candidate. In our implementation, this full scoring procedure takes about 10 minutes, excluding execution time. The runtime is dominated by sequential multi-step world-model rollout and per-transition uncertainty estimation.

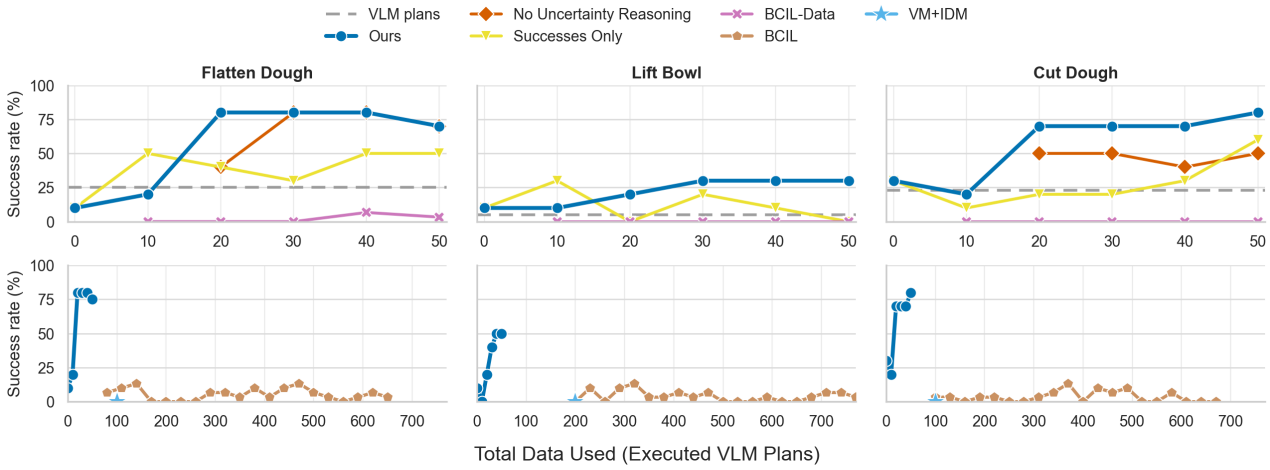



Fig. 11: **Execution success rates in simulation**. The bottom row shows full BCIL baseline results.

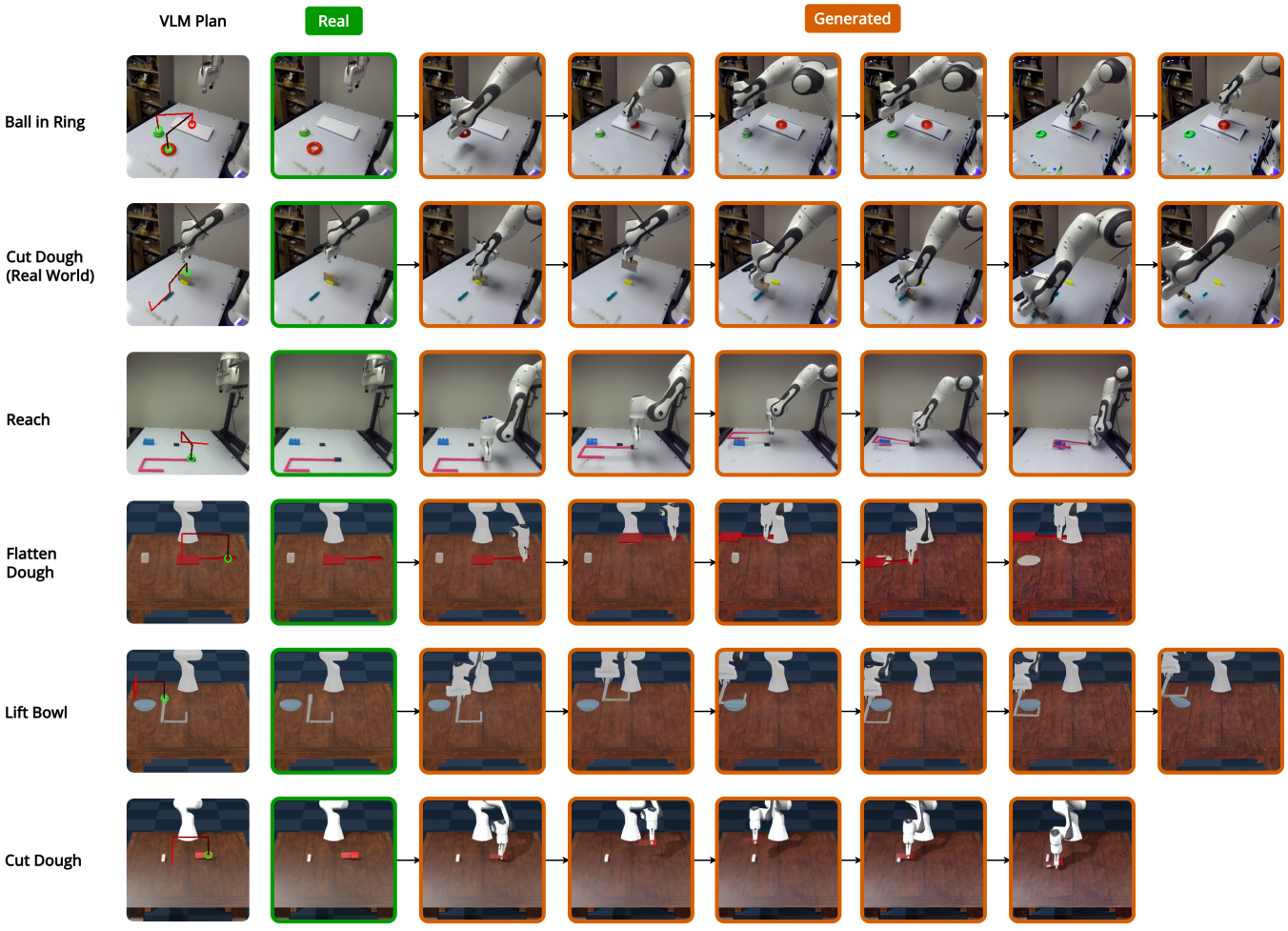



Fig. 12: **Example world model rollouts**. Only the images in the leftmost column are "real"; the rest are generated by a world model trained using our method.