

Evolve Vision-Language-Action Model into an Agent with On-the-fly Tool-use

Supplementary Material

Summary. In the supplementary materials, we provide additional details that complement the main paper. Sec. 7 specifies the implementation of the underlying VLA architecture, the concrete tool modules, and the LoRA design and gating mechanism. Sec. 8 describes the construction of the AT dataset, including source datasets, task and disturbance generation, tool-chain and trajectory synthesis, and detailed statistics and examples. Sec. 9 provides extended quantitative and qualitative results, including per-task and per-category performance, robustness to different disturbance levels, plug-and-play tool case studies, and additional rollouts with failure analyses.

7. Additional Implementation Details

7.1. Base VLA Architecture and Action Chunking

We build ART on top of the 3B π -FAST backbone [35], an autoregressive VLA model that factorizes observation encoding and action decoding into a unified next-token prediction architecture. Unless otherwise stated, we adopt the default configuration of π -FAST and only introduce the minimal modifications required by tool injection.

Observation encoding. At each control step t , the observation $\mathbf{o}_t = (\mathbf{l}, \mathbf{I}_t, \mathbf{q}_t)$ consists of a language instruction \mathbf{l} , one or more RGB images \mathbf{I}_t , and the robot proprioceptive state \mathbf{q}_t . Following [28, 35], we treat both \mathbf{l} and \mathbf{q}_t as text tokens. Concretely, the natural language instruction is tokenized by the underlying VLM tokenizer, while the proprioceptive state is first discretized into 256 uniform bins per dimension and then mapped to integer IDs, which are appended to the text sequence as additional tokens. Visual observations are encoded by the frozen vision encoder of the backbone VLM, producing a sequence of visual tokens that are concatenated with the text tokens. For LIBERO experiments, we use one third-person RGB camera; for As-tribot S1, we use the robot head camera and one wrist-mounted camera, and encode each view independently before concatenation.

FAST action tokenization. The π -FAST backbone converts continuous action chunks into discrete tokens via the FAST tokenizer [35]. Given a sequence of H low-level actions $\mathbf{a}_{t:t+H-1}$, each action dimension is first quantile-normalized to $[-1, 1]$, and a discrete cosine transform (DCT) is applied independently along the temporal dimension. The resulting frequency coefficients are quantized by a scale-and-round operation and flattened into a 1D integer

vector in a low-frequency-first order. A byte-pair encoding (BPE) tokenizer is then trained on these coefficient sequences to obtain the final action tokens. We reuse the publicly released FAST tokenizer with vocabulary size 1024, and overwrite the least used text tokens in the backbone vocabulary with FAST action tokens, following the standard practice in VLA models [6, 28].

Sequence format. During training, the model receives as input the concatenation of instruction tokens, proprioceptive tokens, visual tokens, and previously generated action or tool tokens. The target is the shifted sequence of action or tool tokens, and the model is trained with a standard next-token cross-entropy loss. ART extends this format by inserting reasoning tokens $\mathbf{a}_{r,t}$ and tool tokens $\mathbf{a}_{t,t}$ into the same sequence, while keeping the underlying transformer architecture unchanged.

7.2. Tool Implementations and Integration

In the main paper we conceptually categorize tools into visual, affordance, and embodiment enhancements. Here we describe the concrete instantiations used in our experiments and how they are integrated into the π -FAST observation pipeline. Importantly, all tools operate *outside* the VLA backbone: they only modify the observations $(\tilde{\mathbf{I}}_t, \tilde{\mathbf{l}}, \tilde{\mathbf{q}}_t)$ that are fed into the frozen encoders, so that no architectural change to the backbone is required.

Visual enhancement tools. We implement visual enhancement tools using a family of generic image-to-image translation models designed for low-level restoration. Each model follows an encoder-decoder architecture with multi-scale feature fusion and self-attention blocks, and is trained to map a degraded input image to its restored counterpart. To cover the visual degradations encountered in our generalized tasks, we instantiate ten variants, each specialized for a particular degradation type: (1) low-light conditions, (2) Gaussian noise, (3) color consistency bias, (4) JPEG compression artifacts, (5) quantization artifacts, (6) defocus blur, (7) motion blur, (8) over-exposure, (9) low contrast, and (10) low resolution. For each degradation type $k \in \{1, \dots, 10\}$, we construct paired training data by applying the corresponding corruption to clean images collected from our robot datasets and additional high-quality image collections, and train a dedicated restoration model f_k^{vis} to reconstruct the original clean image.

Given a raw RGB frame $\tilde{\mathbf{I}}_t$ and a visual tool token predicted by the policy, the selected model processes $\tilde{\mathbf{I}}_t$ and

outputs an enhanced image $\mathbf{I}_t^{(k)} = f_k^{\text{vis}}(\tilde{\mathbf{I}}_t)$. When multiple visual tools are activated within the same chunk, we apply the corresponding models sequentially in a fixed canonical order (sorted by tool ID), which makes the composed transformation deterministic. The final enhanced image \mathbf{I}_t is then fed into the standard backbone vision encoder without any architectural modification, so that visual enhancement is realized purely at the observation level.

Affordance tools. Affordance tools produce additional geometric or semantic information about the scene. We implement two such tools in our experiments: depth estimation using Metric3D [47] and category-level object detection using DINO-X [36]. The depth estimator outputs a dense depth map, which we normalize to $[0, 1]$ and render as a grayscale image. The detector outputs bounding boxes and class labels for the top- K objects. To integrate these signals into the VLA, we follow a purely observation-level strategy: (i) we overlay detected bounding boxes and category labels onto the RGB frame, and (ii) optionally stack the depth map as an extra channel and convert it to a pseudo-color image. Both augmented views are fed through the same vision encoder as additional camera inputs, so that the backbone treats affordance information as extra visual context rather than as a new modality. This design keeps the architecture unchanged while exposing rich affordance cues to the policy.

Embodiment tools. Embodiment tools directly manipulate the robot or camera configuration. In LIBERO, we provide tools for camera pan/tilt ($\pm 30^\circ$), zoom-in / zoom-out (implemented as cropping and resizing in the renderer), and resetting the robot arm to a nominal home pose. On Atribot S1, we expose tools for head pan/tilt, small base shifts of the upper body, and resetting both arms to an operational ready pose. When a corresponding tool token is activated, the low-level controller applies the requested change once; the subsequent images and proprioceptive states reflect the new viewpoint or configuration. Embodiment tools therefore do not alter the network inputs directly, but induce new observations \mathbf{I}_{t+1} and \mathbf{q}_{t+1} through the environment dynamics.

Tool state and execution semantics. We maintain a binary state for each tool function in the tool set \mathcal{F} . At the beginning of each action chunk, the LoRA-augmented policy predicts a multi-hot vector $\mathbf{a}_{t,t} \in \{0, 1\}^{|\mathcal{F}|}$ indicating which tools to activate for the next chunk. Visual tools remain active for all frames within the chunk, i.e. they are applied to every $\tilde{\mathbf{I}}_{t:t+H-1}$. Affordance tools are invoked at the first frame of the chunk and their outputs are reused for the entire chunk. Embodiment tools are executed once and

may change the underlying camera or robot state. Conflicting tool combinations (e.g. multiple reset operations) are filtered at data generation time so that the policy only sees valid tool subsets during training.

7.3. LoRA Design and Inference Gating Mechanism

To realize the factorization in Eq. (4) of the main paper while avoiding catastrophic forgetting, ART introduces lightweight LoRA adapters on top of the π -FAST backbone and uses a gating mechanism to decouple tool reasoning from embodied action generation.

Where LoRA is applied. Let $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ denote a linear projection in the transformer backbone. For each such matrix in the language-vision transformer blocks (self-attention, cross-attention, and MLP projections), we attach a LoRA adapter [23] parameterized as

$$W_{\text{eff}} = W + g \cdot \Delta W, \quad \Delta W = BA,$$

where $A \in \mathbb{R}^{r \times d_{\text{in}}}$, $B \in \mathbb{R}^{d_{\text{out}} \times r}$, and r is the LoRA rank. We use a single global rank $r = 16$ for all projections, which adds less than 2% additional parameters compared to the 3B backbone. The scalar gate $g \in \{0, 1\}$ controls whether the LoRA update is active. We do not modify the vision encoder or the FAST tokenizer; LoRA is only attached to the multimodal transformer blocks.

Two-stage training. We adopt a two-stage fine-tuning procedure:

Stage 1: tool reasoning fine-tuning. Starting from the pre-trained π -FAST checkpoint, we freeze all backbone parameters W and optimize only the LoRA parameters $\{A, B\}$ using the tool reasoning loss $\mathcal{L}_{\text{tool}}$ defined in Eq. (5). During this stage, we set the gate $g = 1$ when predicting reasoning tokens $\mathbf{a}_{r,t}$ and tool tokens $\mathbf{a}_{t,t}$, and $g = 0$ when predicting action tokens. This encourages the adapters to specialize exclusively on language-chain construction and tool selection, while preserving the original action decoding behavior of the backbone.

Stage 2: action refinement (optional). After tool reasoning has converged, we optionally perform a short refinement stage for embodied actions using the enhanced observations produced by the ground-truth tool trajectories. In this stage we freeze the LoRA parameters and unfreeze the backbone parameters W . The gate is fixed to $g = 0$ throughout, so that gradients from the action loss $\mathcal{L}_{\text{action}}$ only update the original VLA weights. In practice, we found that a small number of fine-tuning steps on AT is sufficient to recover or slightly improve the original π -FAST action performance under enhanced observations.

Source	Train	Val	Total
LIBERO	10,800	1,200	12,000
DROID	8,100	900	9,000
BridgeData-v2	8,100	900	9,000
Total	27,000	3,000	30,000

Table 4. Composition of the AT dataset across source benchmarks. We construct 30,000 augmented trajectories by extending demonstrations from LIBERO, DROID, and BridgeData-v2 with tool-use reasoning.

Inference-time gating. At inference time, we alternate between *tool mode* and *action mode* for each action chunk. Given the current raw observations $\tilde{o}_{t:t+H-1}$, we first run the transformer with gate $g = 1$ and decode the reasoning tokens $\mathbf{a}_{r,t}$ and tool tokens $\mathbf{a}_{t,t}$ until the end-of-tool symbol is predicted. The corresponding tools are executed, producing enhanced observations $\mathbf{o}_{t:t+H-1}$. We then switch to action mode by setting $g = 0$ and decode FAST action tokens for the next H steps from the same backbone. Since the effective weights in action mode reduce to $W_{\text{eff}} = W$, the backbone behaves identically to the original π -FAST policy during action generation, which empirically stabilizes training and mitigates catastrophic forgetting on the pre-trained action repertoire.

8. Details of the AT Dataset

8.1. Source Datasets and Splits

The AT dataset augments existing robot demonstrations with long-horizon tool-use reasoning and disturbance-aware observations. We build AT on top of three widely used VLA benchmarks: LIBERO [33], DROID [27], and BridgeData-v2 [44]. Across all sources, we construct in total 30,000 augmented trajectories paired with tool-use annotations.

We select base demonstrations from each dataset so as to balance simulated and real-world domains and to cover diverse object sets and manipulation skills. Table 4 summarizes the resulting composition. We use 27,000 trajectories for training and 3,000 for validation, keeping a consistent 90/10 split within each source.

8.2. Task and Disturbance Generation

Given a base demonstration, AT injects controlled disturbances and task transformations to create scenarios that explicitly require tool use. We operate on three axes: visual degradation, affordance incompleteness, and embodiment perturbation, corresponding to the three tool families in Sec. 4.1.

Visual disturbances. For each selected episode, we sample one or more visual degradation types from the following ten categories: (1) low-light, (2) Gaussian noise, (3) color consistency bias, (4) JPEG compression artifacts, (5) quantization artifacts, (6) defocus blur, (7) motion blur, (8) over-exposure, (9) low contrast, and (10) low resolution. These degradations are applied to camera frames either globally (for the entire episode) or locally (for specific time windows), producing raw images \tilde{I}_t that are significantly harder to interpret. The corresponding visual enhancement tools in ART are then responsible for restoring the images to a usable quality level.

Affordance disturbances. Affordance disturbances are designed to remove critical geometric or semantic information that is necessary to solve the task. We consider two main types: (i) missing depth information, and (ii) missing object target information. To instantiate missing depth information, we modify the instructions so that the intended target is specified by spatial relations (e.g., “*place the farthest object into the drawer*”) and perturb the scene so that depth ordering cannot be inferred reliably from RGB alone. To instantiate missing target information, we rewrite the language to describe the goal through abstract attributes or roles (e.g., “*place the object that matches the reference*”), while the original object labels are hidden. In both cases, the depth estimation and object detection tools are required to recover the missing affordance cues.

Embodiment disturbances. Embodiment disturbances perturb the camera viewpoint or the robot’s initial configuration, so that the base demonstration is no longer directly executable. We explicitly distinguish six embodiment tool-task types: (1) camera zoom-in, (2) camera zoom-out, (3) camera roll, (4) camera yaw, (5) camera pitch, and (6) abnormal arm initialization / arm reset. During data generation, we randomly sample combinations of these perturbations and apply them at the beginning of an episode or at intermediate keyframes. The corresponding embodiment tools must be invoked by the policy to recover a configuration compatible with the original actions.

Combinatorial disturbance patterns. To encourage long-horizon tool reasoning, we combine the three disturbance families in a Cartesian-product-like manner. For each base demonstration, we first sample the intended tool-chain length $L \in \{2, \dots, 4\}$ (cf. Sec. 8.4), and then select L concrete disturbance types from the union of visual, affordance, and embodiment issues without replacement. This procedure yields tasks where the robot simultaneously faces multiple challenges (e.g., low-light + zoom-out + missing target identity + missing depth + arm mis-initialization), and must

compose multiple tools in the correct order to restore a solvable state.

8.3. Tool-Chain and Trajectory Synthesis

Given a disturbed episode specification, we synthesize an augmented trajectory that interleaves the original low-level actions with explicit tool-use reasoning. The process consists of three stages and ultimately yields a ReAct-style sequence of Observation–Reason–Action entries.

Step 1: structured task generation. We convert each disturbed episode into a structured text description that includes: (i) the high-level instruction, (ii) a natural-language explanation of the injected disturbances (e.g., “*the scene is under low light and the camera is zoomed out*”), (iii) the list of available tools and their informal documentation, and (iv) the original action demonstration as a sequence of action chunks. This representation serves as the input context for generating tool-use plans.

Step 2: reasoning and tool-chain specification. Inspired by T3-Agent [18], we prompt a large language model with the structured description and ask it to propose a step-by-step tool-use plan that specifies *when* and *which* tools should be invoked. The model outputs a natural-language sketch such as “*first enhance the low-light image, then zoom out the camera, then run object detection, then estimate depth, finally reset the arm before executing the grasp*”. We do *not* store this long reasoning as a single text field. Instead, we parse it into a structured tool-chain specification (implemented as a JSON-like object in our code), which contains L entries ($f^{(1)}, \dots, f^{(L)}$). Each entry records the tool identity (e.g., low-light-enhance, depth-estimation), its position in the chain, and a short local description of what the tool is expected to fix.

Step 3: ReAct-style trajectory generation. We then align the structured tool chain with the original action trajectory and serialize the whole episode into a ReAct-style sequence. Concretely, each AT trajectory is stored as a list of interleaved entries:

```
Observation: ...
Reason: ...
Action: <tool use: low-light enhance>
Observation: ...
Reason: ...
Action: <embodied action tokens>
...
```

The initial Observation block encodes the multi-modal input \tilde{o}_0 : the disturbed images, the (possibly modified) language instruction, and the initial proprioceptive

state. For each subsequent step, we consult the structured tool-chain specification (for tool calls) or the original demonstration (for embodied actions), and generate a *local* Reason sentence that explains only the next decision, rather than the entire chain. Each Action block then contains either (i) a tool invocation derived from the corresponding entry in the tool-chain specification, or (ii) an embodied action chunk taken from the original trajectory.

For tool steps, the Action field is a symbolic tool call in text form, such as Action: <tool use: low-light enhance> or Action: <tool use: depth estimation>. This is later mapped to the corresponding discrete tool token $a_{t,t}$ in the vocabulary. The preceding Reason describes why this particular tool is needed at the current moment, providing supervision for reasoning tokens $a_{r,t}$. For embodied steps, the Action field stores the FAST-encoded action chunk [35], serialized as Action: <embodied action tokens>, i.e., the sequence of action tokens corresponding to $A_t = [a_t, \dots, a_{t+H-1}]$. In this case, the preceding Reason explains the high-level intent of the motor command (e.g., “*move the gripper towards the detected cup*”).

In summary, each AT trajectory can be viewed as a ReAct-like [46] unrolling of the augmented policy: starting from an initial Observation, the agent alternates between producing language Reason tokens and Action tokens, where the latter may be either tool calls or embodied action chunks. When training ART, we linearize this sequence into a single token stream and supervise the model with next-token prediction, while the environment and tool backends provide the corresponding observation updates. This serialization makes the dataset format simple and explicit, while the underlying structured tool-chain specification from Step 2 keeps the global plan well organized and consistent with the long-horizon tool-use behavior depicted in Fig. 3.

8.4. Statistics and Examples

We next summarize key statistics of AT.

Tool-task coverage. Because a single trajectory may invoke multiple tools, the total number of tool usages exceeds the number of trajectories. Table 5 reports the frequency of each tool-task type across the entire dataset. On average, each trajectory uses 3.2 tools, leading to 96,000 tool calls over 30,000 trajectories. AT covers 18 distinct tool types in total: 10 visual enhancement tools, 2 affordance tools, and 6 embodiment tools. Within the vision and embodiment categories, we distribute disturbances evenly over the corresponding tool types, while affordance tools are skewed towards object detection to reflect the prevalence of target-identification tasks.

Aggregated at the category level, visual enhancement

Category	Tool-task type	# Tool usages	Share of all tool usages
Visual	Low-light enhancement	3,000	3.1%
	Gaussian denoising	3,000	3.1%
	Color consistency correction	3,000	3.1%
	JPEG artifact removal	3,000	3.1%
	Quantization artifact removal	3,000	3.1%
	Defocus deblurring	3,000	3.1%
	Motion deblurring	3,000	3.1%
	Over-exposure correction	3,000	3.1%
	Contrast enhancement	3,000	3.1%
	Super-resolution (low-res fix)	3,000	3.1%
Affordance	Depth completion / estimation	21,000	21.9%
	Object detection / target identification	30,000	31.3%
Embodiment	Camera zoom-in	2,500	2.6%
	Camera zoom-out	2,500	2.6%
	Camera roll	2,500	2.6%
	Camera yaw	2,500	2.6%
	Camera pitch	2,500	2.6%
	Arm reset from abnormal initialization	2,500	2.6%
Total		96,000	100%

Table 5. Tool-task statistics in the AT dataset. Each trajectory may use multiple tools; the table counts individual tool usages across all 30,000 trajectories. On average, each trajectory contains $96,000/30,000 = 3.2$ tool calls. In total, AT covers 18 distinct tool types (10 visual, 2 affordance, 6 embodiment).

Chain length	# Trajectories	Percentage
2 tools	7,000	23.3%
3 tools	10,000	33.3%
4 tools	13,000	43.3%
Total	30,000	100%

Table 6. Distribution of tool-chain lengths in AT. Each chain length L indicates that the trajectory contains L distinct tool calls arranged along the temporal axis. The resulting average chain length is 3.2 tools per trajectory.

tools are used 30,000 times ($\approx 31.3\%$ of all tool calls), affordance tools 51,000 times ($\approx 53.1\%$), and embodiment tools 15,000 times ($\approx 15.6\%$). In terms of trajectory coverage, a large fraction of trajectories contain affordance-related tools, reflecting the emphasis on challenging target-identification and spatial-reasoning tasks in AT. Note that embodiment disturbances are only applied to simulated LIBERO trajectories; for real-world datasets such as DROID and BridgeData-v2, the camera placement and robot setup are hardware-fixed in the logs, so we do not synthesize additional viewpoint or initialization perturbations there, which also explains the smaller share of embodiment tool usages in Table 5.

Tool-chain length. Table 6 shows the distribution of tool-chain lengths. We explicitly bias the dataset towards non-trivial chains: every trajectory contains at least two tool calls, and more than forty percent of the trajectories use four tools in a single episode. Overall, the average chain length is 3.2 tools per trajectory, indicating that AT systematically targets long-horizon tool-use scenarios where multiple tools must be composed in sequence (e.g., “*low-light enhancement* \rightarrow *object detection* \rightarrow *depth estimation* \rightarrow *arm reset*”).

Comparison to similar work in VLM research. T3-Agent [18] introduces MM-Traj, a multi-modal tool-usage dataset with 20K trajectories for generic VLM-driven agents. AT is complementary and pushes this line of work towards embodied, robot-centric settings. In particular, AT increases both the scale and structural complexity of trajectories: it contains 30K robot manipulation trajectories, covers 18 distinct tools (10 visual, 2 affordance, 6 embodiment), and achieves an average tool-chain length of 3.2, which is longer than the typical chains in MM-Traj. This makes AT a challenging testbed for studying long-horizon tool-use in Vision-Language-Action models. Detailed qualitative rollouts and visualizations are provided in Sec. 9.

Instruction: Pick the right item on the plate, and pick up the left item. (Over-Exposure)

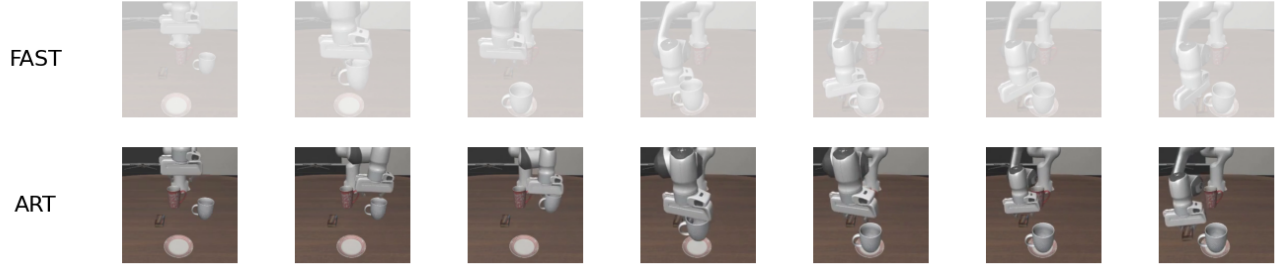


Figure 5. Qualitative comparison under **over-exposure**. The instruction is “*Pick the right item on the plate, and pick up the left item.*”. The FAST baseline fails to localize the objects in the second instruction under severe over-exposed images, while ART first restores the visual input via tool-use and then successfully completes the two-stage pick task.

Instruction: Put the left cup in the microwave. (Low-Resolution)

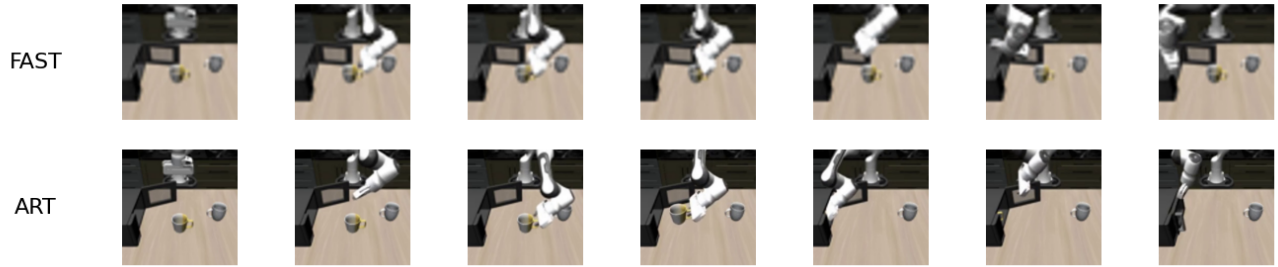


Figure 6. Qualitative comparison under **low-resolution** degradation. The instruction is “*Put the left cup in the microwave.*”. Due to the heavy down-sampling, FAST struggles to track the target cup and the microwave opening, whereas ART enhances the resolution with vision tools and executes a precise pick-and-place trajectory.

9. Extended Qualitative Experiments

In this section, we present additional qualitative rollouts that complement the quantitative results in the main paper, which are tested on LIBERO [33] with extra corruptions. Each figure compares the end-to-end finetuned FAST policy (top row) with our ART (bottom row) under different visual disturbances from AT. The instruction text and the corresponding corruption type are overlaid at the top of each image, while the temporal evolution proceeds from left to right.

Beyond these static rollouts, we provide supplementary video clips that visualize the full trajectories, including intermediate tool activations and robot motions, for a broader set of tasks and disturbance patterns. These videos offer a more detailed view of how ART composes on-the-fly tool-use with continuous action generation in long-horizon manipulation tasks.

Instruction: Pick up the nearest item in the basket. (Jittering)



Figure 7. Qualitative comparison under **jittering**. The instruction is “Pick up the nearest item in the basket.”. Strong motion jittering introduces temporal inconsistencies that confuse FAST and lead to unstable grasps. By invoking the jitter correction model, ART obtains a more consistent observation stream and reliably selects the nearest item in the basket.

Instruction: Pick the middle one on the right item, and then pick the left on the right item. (Low-Contrast)

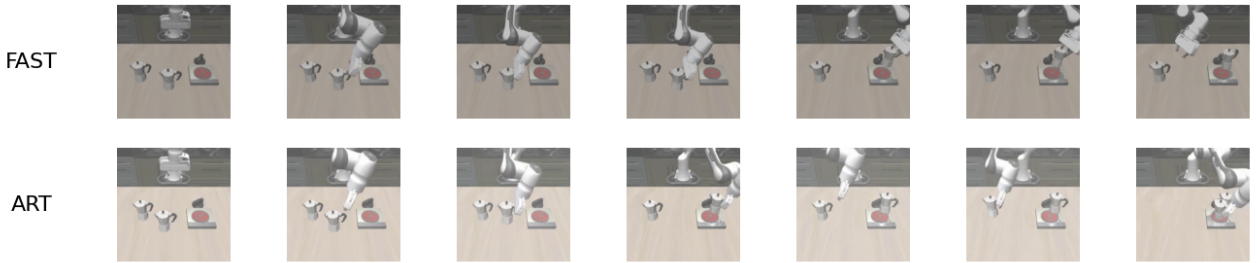


Figure 8. Qualitative comparison under **low-contrast** degradation. The instruction is “Pick the middle one on the right item, and then pick the left one on the right item.”. In this case the cups and supporting object have very similar appearance to the background. FAST mislocalizes the target in the second instruction, while ART leverages contrast-enhancement and detection tools to recover the scene structure and correctly executes the sequential pick-and-place operations.

Instruction: Pick up the nearest bowl in the plate. (Noise)

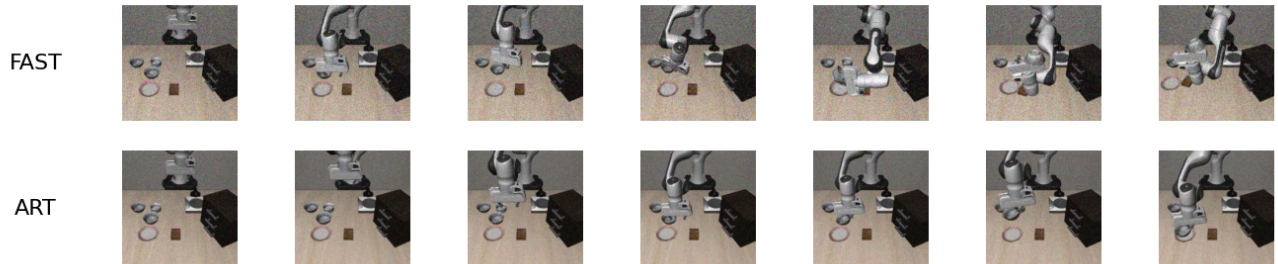


Figure 9. Qualitative comparison under **Gaussian noise**. The instruction is “Pick the nearest bowl in the plate.”. Under heavy sensor noise, FAST frequently fails to recognize the plate position and stops. In contrast, ART first denoises the scene using vision tools and then completes the instructed pick task.