

GT-VLA: Target-Conditioned Trace Guidance for Generalizable Robotic Manipulation

Ninghan Zhong^{1*}, Jing-Chen Peng^{1*}, Sriram Vishwanath¹

Abstract—Vision-Language-Action (VLA) models have shown strong performance on robotic manipulation, but they often struggle to generalize to unseen tasks, configurations, and long-horizon settings. A key challenge is that VLAs overfit to training scenes and fail to follow novel language instructions. Off-the-shelf vision-language models (VLMs) often provide stronger generalization, but cannot directly control robot actions. To combine the common sense of VLMs with VLA control, we propose Guided Trace VLA (GT-VLA), a steerable framework that accepts guidance from an external generalist VLM through trace-conditioned action generation. GT-VLA uses a generalist model to identify semantic guidance for the current skill, converts this guidance into a 2D visual trace, and conditions its action policy on the resulting trace-rendered observation. This design separates semantic target acquisition, trace generation, and low-level action execution, allowing high-level guidance to propagate to robot actions. GT-VLA uses a Mixture-of-Experts architecture with skill-specific trace and action modules for robust execution. We evaluate GT-VLA on LIBERO and a physical robot platform, showing improved generalization over recent VLA baselines in both settings. The code and additional supplemental materials are available on our project website at <https://ivaniz.github.io/gt-vla/>.

I. INTRODUCTION

Vision-Language-Action models (VLAs) integrate perception, planning, and control into unified policies, excelling at in-distribution manipulation [1], [2]. However, they frequently suffer from scene overfitting, repeating memorized behaviors rather than adapting to novel instructions or unseen configurations [3], [4]. While decomposing complex tasks into atomic subtasks [5], [6] or querying generalist VLMs for runtime steering [6] improves robustness, these high-level semantic signals are often ignored by the low-level controller. Effective generalization requires a steerable execution pipeline that propagates external semantic guidance all the way to action generation.

To bridge this gap, we propose **Guided Trace VLA** (GT-VLA), a framework that injects guidance from an off-the-shelf generalist model through an intermediate, actionable visual representation. GT-VLA first queries a generalist VLM to decompose a task into a skill plan (Fig. 1). For each skill, GT-VLA generates 2D visual traces anchored to a semantic point predicted by the generalist VLM, which subsequently guides downstream action execution. This modularizes high-level

intent and low-level control, preserving the VLA’s competence while keeping it steerable and interpretable.

At the core of GT-VLA is a Trace-Action Mixture-of-Experts (TA-MoE) architecture featuring skill-specific trace and action heads. The trace head uses conditional flow-matching to generate traces conditioned on the semantic target. This trace is then used to condition action generation. We further propose a set of data augmentations to encourage the model to respect semantic guidance and be robust to imperfect generated traces.

We evaluate GT-VLA in simulation and the real world. In simulation, we use a clean LIBERO train-test split to test generalization. We also validate GT-VLA on a physical robot platform. Across both settings, GT-VLA outperforms recent VLA baselines, demonstrating the benefit of semantic-trace conditioning for steerable and generalizable manipulation.

Our core contributions are:

- **GT-VLA**: A framework that translates high-level VLM guidance into visual traces to steer low-level VLA execution on unseen tasks and configurations.
- **Trace-Action Mixture-of-Experts (TA-MoE)**: A modular architecture coupling skill-specific trace generation with trace-guided action execution.
- **Robust Trace-Conditioning Strategies**: Guidance-conditioned generation, trace perturbation, and scene masking ensure the policy respects guidance while remaining robust to imperfect traces.

II. RELATED WORK

A. Vision-Language-Action Models

VLAs leverage large-scale pre-training and semantic priors to unify language understanding, visual perception, planning, and control within a single robotic policy [1], [2]. Recent VLAs achieve improved performance across diverse tasks [7], [8]. However, they still struggle with unseen objects, novel scene configurations, and long-horizon task compositions [3].

To improve long-horizon execution, some methods decompose tasks into short subtasks, using embodied chain-of-thought to switch between reasoning and acting [6] or directly predicting an atomic skill to execute [5]. Although these methods improve sub-task composition, VLAs still overfit to visual observations, reproducing memorized behaviors even when instructed otherwise [4].

Another line of work uses explicit high-level reasoners to generate plans or intermediate decisions that are

*Equal contribution.

¹School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA. {nzhong34}@gatech.edu

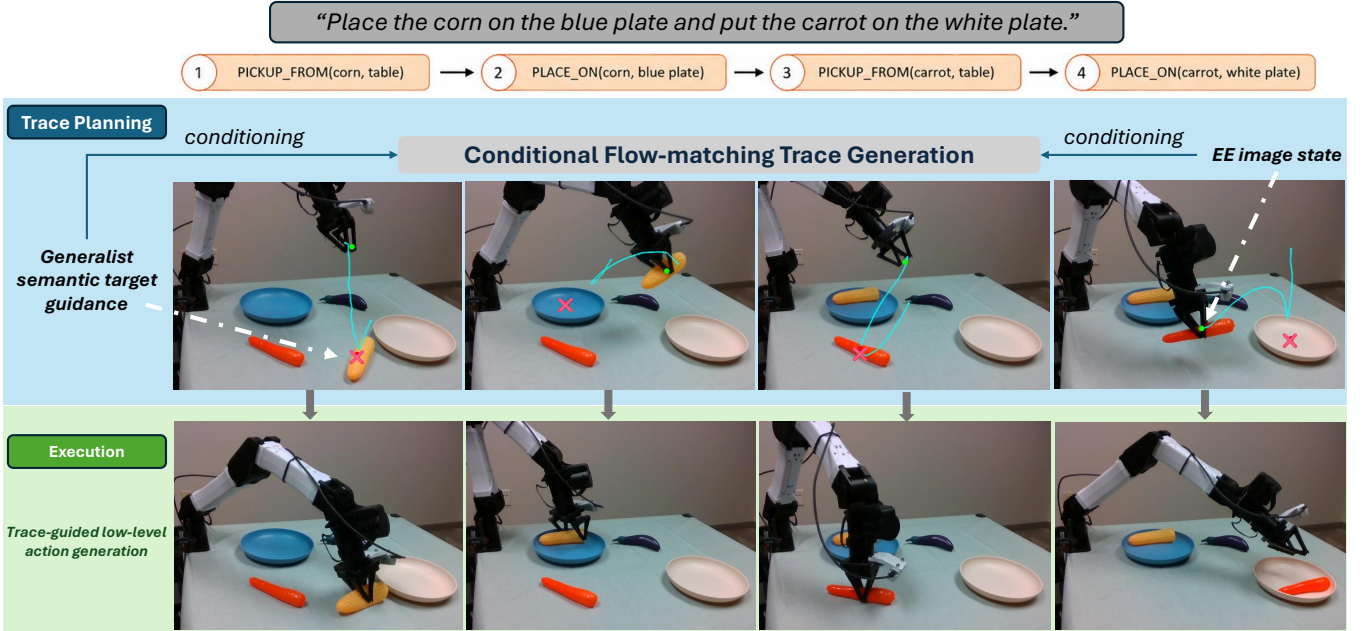


Fig. 1: GT-VLA overview. An off-the-shelf generalist VLM decomposes tasks into skills and provides skill-specific guidance. This is converted into an image-space trace conditioning for the action policy, ensuring execution follows high-level intent.

executed step by step by a lower-level VLA [7], [9]–[11]. These hierarchical designs, introduced in embodied chain-of-thought policies [11], improve generalization by separating semantic reasoning from control. Some methods freeze the VLM backbone to preserve general-purpose reasoning, but this may reduce task performance [7], [10]. Our work builds on the view that hierarchy is useful but insufficient: effective generalization requires a framework that can propagate guidance to the action generation level.

B. Runtime Verification and Steering

Runtime verification and steering improve VLA robustness by introducing inference-time signals that verify or guide robot actions. Learned critics can steer candidate actions [12], but may not generalize to novel scenes or tasks. Mechanistic interpretability methods can control fine-grained behavior attributes such as speed or clearance [13], while ReSteer [14] evaluates and improves steerability with targeted data collection.

Recent methods also use VLMs for runtime verification and guidance. RoboMonkey [15] selects among sampled actions using a fine-tuned VLM, LoHo-Manip [16] fine-tunes a VLM for task decomposition and visual guidance, and AHA [17] trains a VLM to detect and explain manipulation failures. FOREWARN [18] aligns a VLM with latent-space predictions to steer robot actions. These methods show the promise of VLM-based steering, but often require additional fine-tuning of the high-level VLM.

Off-the-shelf VLMs can also directly generate policies [19], orchestrate tools [20], or act as performance critics [6]. However, such external verification cannot correct execution if the underlying VLA fails to incorporate the guidance. Our work follows the run-

time steering approach, using an off-the-shelf generalist model for semantic guidance and designing the downstream pipeline to be steerable by this guidance.

C. Visual Prompting and Trace Guidance

Visual prompting can provide spatial guidance to generalist models. In computer vision, the SAM series shows that simple geometric point and box prompts can guide zero-shot segmentation [21]. Related robotics methods use visual hints or intermediate representations to connect semantic intent with low-level action. VLMs can generate semantic guidance and constraints for motion primitive execution [22]; MOKA [23] uses keypoint affordances from a VLM to command a low-level VLA; SpatialVLA [24] studies 3D spatial encodings of action tokens; *Fast-ThinkAct* [25] connects semantic guidance to control with visual latent planning.

A growing line of work uses traces as visual guidance for manipulation. TraceVLA [26] uses historical end-effector traces to improve spatial-temporal awareness, while RT-Trajectory [27] shows that trajectory sketches can act as steerable policy inputs. ATM [28] predicts motion flow for the entire scene. Closer to our setting, LoHo-Manip [16] uses a fine-tuned manager for subtask decomposition and image-space trace planning, followed by a trace-conditioned executor, while MolmoAct [29] jointly predicts depth, traces, and actions within a monolithic action-reasoning model. GT-VLA differs from prior work by both explicitly decomposing long-horizon tasks into skills, and accepting guidance from an external, off-the-shelf generalist VLM without fine-tuning, decoupling semantic guidance from trace generation and low-level control.

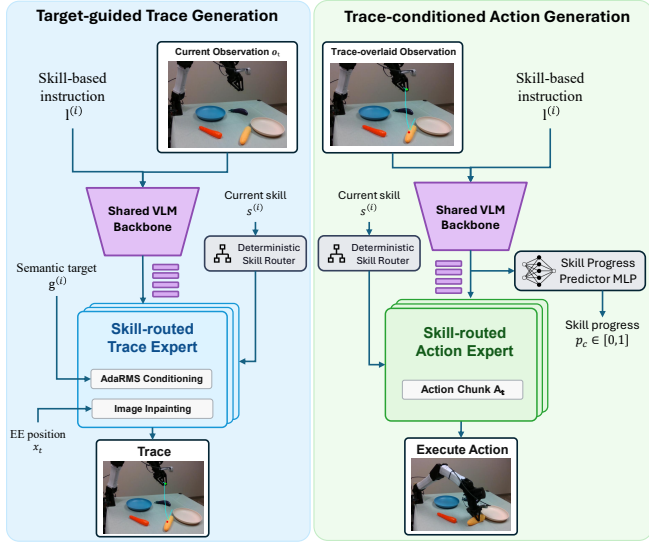


Fig. 2: GT-VLA architecture. **Left:** a skill-routed trace expert generates a 2D trace anchored to the semantic target $g^{(i)}$ and end-effector position x_t . **Right:** a skill-routed action expert predicts actions from the trace-overlaid observation, and an MLP predicts skill progress. A deterministic router selects experts for the current skill.

III. METHOD

The central goal of GT-VLA is to make VLA execution steerable by semantic guidance from an off-the-shelf generalist VLM. Instead of relying on the VLA to infer all of the task semantics, GT-VLA uses an off-the-shelf generalist VLM to provide sparse guidance at inference time. This guidance is converted into a dense visual trace to aid action generation, injecting high-level intent while preserving the VLA’s learned low-level control competence.

Given an instruction ℓ and observation o_t , the generalist VLM first decomposes the task into parameterized manipulation skills. At the start of each skill, the same generalist model identifies a semantic target point in the image, such as an object to grasp, a handle to manipulate, or a placement location. A skill-specific trace expert then generates a 2D end-effector trace conditioned on this target, and the corresponding action expert executes low-level actions given the trace and observation. Fig. 2 illustrates the architecture, and Alg. 1 gives the inference pseudocode.

A. Skill Interface and Semantic Target Acquisition

We use a finite library of manipulation skills $\mathcal{S}$, where each skill $s \in \mathcal{S}$ is represented as a semi-structured operator with arguments. For example, the *PLACE* skill may be instantiated as *PLACE(bowl, table)*. Given a task instruction ℓ and observation o_t containing camera views and robot state, we query an off-the-shelf generalist VLM Φ to produce a skill plan

$$P = \Phi_{\text{plan}}(\ell, o_t) = (s^{(1)}, s^{(2)}, \dots, s^{(k)}), \quad (1)$$

where each $s^{(i)} \in \mathcal{S}$ is an atomic skill to be executed in sequence. This decomposition reduces a long-horizon

instruction into shorter manipulation segments, but does not by itself guarantee that the low-level policy will execute each segment in the intended way. To make skill execution steerable, GT-VLA additionally queries the generalist model for a semantic target point at the start of each skill. For the current skill $s^{(i)}$, the generalist model predicts a target in normalized image coordinates:

$$g^{(i)} = \Phi_{\text{target}}(\ell, o_t, s^{(i)}) \in [0, 1]^2, \quad (2)$$

$g^{(i)}$ represents a spatial goal for the current skill. For example, it may mark the object to pick up, the handle of a drawer to open, or the desired placement location. This target serves as a semantic anchor for trace generation, allowing the policy to convert sparse high-level guidance signal into a feasible end-effector path.

B. Target-Conditioned Trace Generation

The purpose of the trace generator is to translate the sparse semantic target into an actionable visual guidance signal. Let $x_t \in [0, 1]^2$ denote the projection of the robot end-effector position into image coordinates at timestep t . We define the ground-truth remaining trace as the future portion of the image-space trajectory that completes the skill:

$$\tau_t^* = (x_t, x_{t+1}, \dots, x_{t^{\text{end}}}), \quad (3)$$

where t^{end} is the final timestep of the current skill. In practice, we resample each trace by linear interpolation along its arc length to obtain a fixed-length sequence $\tilde{\tau}_t^* \in \mathbb{R}^{H \times 2}$ where H is the trace length. We predict this semantic trace using a trace prediction model π^τ :

$$\pi^\tau(\tau_t | o_t, x_t, g^{(i)}, \ell^{(i)}). \quad (4)$$

The language input $\ell^{(i)}$ combines ℓ and $s^{(i)}$, giving both high-level and skill-specific information. π^τ is implemented as a flow-matching model, with the loss

$$\mathcal{L}^{\text{trace}} = \mathbb{E}_{(\tilde{\tau}_t^*, o_t, \ell^{(i)}, g^{(i)}) \sim \hat{\mathcal{D}}, \epsilon, \lambda} [\|\hat{v} - (\epsilon - \tilde{\tau}_t^*)\|_2^2], \quad (5)$$

$$\hat{v} \equiv v_\theta^\tau(\tilde{\tau}_t^*(1 - \lambda) + \epsilon \lambda, \lambda, o_t, x_t, g^{(i)}, \ell^{(i)})$$

where $\hat{\mathcal{D}}$ is a skill-annotated demonstration dataset, $\lambda \in [0, 1]$ is the flow-matching time, ϵ is the noise vector, and v_θ^τ is the predicted trace velocity field to be learned. We draw λ and ϵ following [7]. Ground-truth traces are obtained by projecting the end-effector position into the camera image. Details on dataset construction are provided on the project website.

The current end-effector position x_t is incorporated as an inpainting-style constraint to clamp the trace starting point. The semantic target provides a second constraint, injected through Adaptive RMSNorm conditioning (Fig. 2 left). These constraints enforce the trace to start from the robot’s current image-space position and remain anchored to the spatial goal.

C. Robust Trace-Conditioned Action Learning

The trace provides an interpretable 2D plan, but the action expert must map it to 6-DoF robot motion and gripper commands. We train the policy to follow trace guidance while remaining robust to imperfect traces and 2D–3D ambiguity. The action expert is implemented as a flow-matching model following [1], with the trace τ_t rendered onto the visual observation:

$$\pi^{\text{act}}(A_t | \text{overlay}(o_t, \tau_t), \ell^{(i)}). \quad (6)$$

To force the action expert to respect the trace guidance while considering scene geometries, and to bridge the gap between training demonstrations and imperfect inference traces, we propose three data augmentation techniques. Visual illustrations are provided in Fig. 4.

Random scene drop. To encourage the action expert to respect the trace, we randomly mask the scene observation while keeping the rendered trace visible. This forces the policy to sometimes act primarily from the trace guidance, reducing its tendency to overfit to memorized visual configurations in the training scenes.

Random trace drop. Because the trace is defined in 2D image space, blindly following it can be harmful when depth, occlusion, or contact geometry matters. We randomly remove the trace overlay during training, providing only the clean image. This preserves the policy’s ability to reason from the original scene image and robot state, reducing the learned controller’s sensitivity to inaccurate or geometrically ambiguous traces.

Low-frequency trace perturbation. To bridge the distribution gap between ground-truth traces at training time and generated traces at inference time, we perturb the rendered training trace with smooth low-frequency noise. This produces traces that remain plausible end-effector paths but are slightly shifted or bent, addressing distribution shift between the training traces and generated traces at inference time.

D. Trace-Action Mixture-of-Experts Architecture

Following the recipe from [1], we instantiate π^τ and π^{act} as small “expert” transformers attending to a shared VLM backbone. We instantiate the VLM backbone using a pretrained checkpoint of Gemma 2B [7].

One potential challenge of image-trace conditioning is ambiguity: *PICK* and *PLACE* skills have similar traces but require different actions. Inspired by [5], we instantiate π^{act} as a deterministically-routed model with one expert assigned to each skill category in our dataset. We refer to this as a skill-routed mixture-of-experts (MoE) following [5]. At training and inference time, only the expert corresponding to the skill being run is invoked. Each expert is a 300M parameter transformer, following the action expert design of [1].

We implement π^τ as a lightweight flow-matching transformer (~60M parameters), following the same skill-routed MoE design as the action generator.

We also attach a lightweight MLP head (Fig. 2 right) to the final-layer hidden states of the shared VLM backbone, which predicts the progress $p_c \in [0,1]$ of the current skill. At inference, GT-VLA advances to the next skill once $p_c \geq \eta_c$ (Alg. 1), where η_c is a chosen threshold. This controls transitions between steps during inference.

E. Inference Algorithm

Algorithm 1: GT-VLA Inference

Input: ℓ , o_{init} , $f_{\text{upd}} \geq 1$, $\eta_c \in [0,1]$

```

1  $o \leftarrow o_{\text{init}}$  // initialize obs.
2  $\{s^{(1)}, \dots, s^{(k)}\} \leftarrow \Phi_{\text{plan}}(\ell, o)$  // get skill plan (Sec. III-A)
3 for each skill  $s^{(i)} \in \{s^{(1)}, \dots, s^{(k)}\}$  do
4    $\ell^{(i)} \leftarrow \text{concat}(\ell, s^{(i)})$  // concat prompt and skill
5    $g^{(i)} \leftarrow \Phi_{\text{target}}(\ell, o, s^{(i)})$  // semantic target (Sec. III-A)
6   skillDone  $\leftarrow$  False
7   while not skillDone do
8      $x \leftarrow \text{project}(o)$  // end effector image projection
9      $\tau \sim \pi^\tau(o, x, g^{(i)}, \ell^{(i)})$  // generate trace (Sec. III-B)
10    for each  $j = 1, \dots, f_{\text{upd}}$  do
11       $\tilde{o} \leftarrow \text{overlay}(o, \tau)$  // overlay visual trace
12       $p_c \leftarrow \text{comp}(\tilde{o}, \ell^{(i)})$  // skill progress (Sec. III-D)
13      if  $p_c \geq \eta_c$  then
14        skillDone  $\leftarrow$  True
15        break
16       $A \sim \pi^{\text{act}}(\tilde{o}, \ell^{(i)})$  // sample actions (Sec. III-C)
17       $o \leftarrow \text{execute}(A)$  // execute actions, update obs.
```

GT-VLA’s inference pipeline is presented in Algorithm 1. The algorithm takes as input a natural language prompt ℓ , current observation o_t , a trace update frequency $f_{\text{upd}} \geq 1$ that controls the number of action expert executions between each trace refresh, and a skill completion threshold $\eta_c \in [0,1]$. First, GT-VLA queries a generalist VLM to decompose the task into a skill plan (Line 2). For each skill, the algorithm constructs a structured skill command and queries the VLM for a semantic target (Lines 4, 5). Execution then proceeds via a nested loop. The outer loop regenerates a visual trace every f_{upd} action chunks (Lines 8, 9). The inner loop first obtains the trace overlaid visual observation (Line 11) and predicts the current skill progress $p_c \in [0,1]$ (Line 12). If the predicted progress reaches the completion threshold η_c (Lines 13–15), the algorithm proceeds to the next skill. Otherwise, the algorithm samples and executes actions, and updates current observation (Lines 16, 17).

In our current formulation, the high-level skill plan is executed open-loop. Algorithm 1 can be modified to execute the skill plan in a closed-loop manner, where the high-level generalist VLM is periodically queried to replan the skill sequence to improve failure recovery. We leave evaluation of this extension to future work.

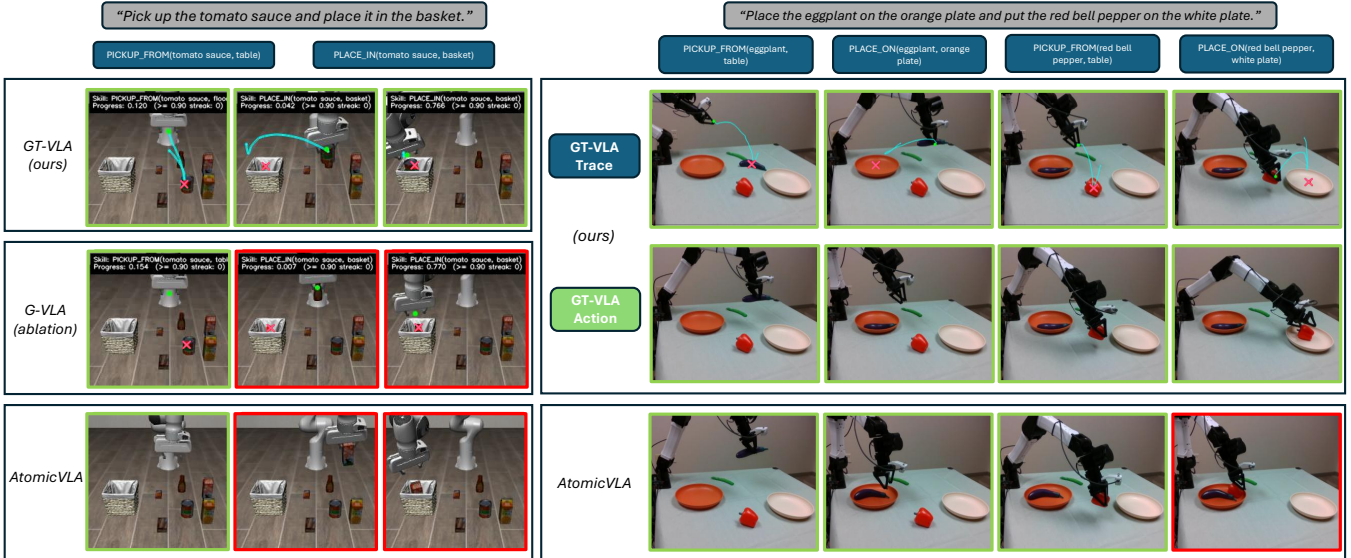


Fig. 3: **Comparing steerability of GT-VLA against baselines and ablations.** On the left we show a task from Libero-Object, which is an OOD task. GT-VLA (top-left) is steerable by the external semantic target. G-VLA executes memorized actions despite having correct semantic guidance (mid-left). AtomicVLA (bottom-left) also fails to generalize to this instruction. In real-world experiments, we show a 4-step OOD task. GT-VLA completes all skills, while AtomicVLA fails to adapt.

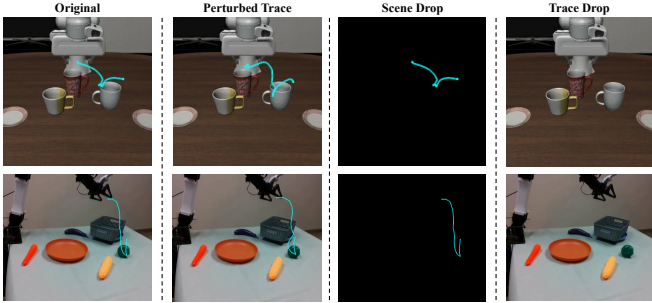


Fig. 4: Trace-conditioning augmentations (Sec. III-C), shown on LIBERO (top row) and the real robot (bottom row).

IV. EXPERIMENTS

Our evaluation involves four major components: (1) comparisons against recent baselines (Sec. IV-A–IV-D), (2) ablation studies on proposed designs (Sec. IV-E), (3) pilot studies on GT-VLA’s robustness to guidance noise and generalist VLM modularity (Sec. IV-F, IV-G), and finally (4) failure analysis of GT-VLA (Sec. IV-H).

Comparisons against Recent Baselines: We evaluate GT-VLA in both simulation and on a physical robot platform. In this part, our experiments are designed to answer two questions. First, does GT-VLA improve generalization to unseen tasks, object configurations, and task compositions compared with recent VLA baselines? Second, does converting semantic guidance from an off-the-shelf VLM into an explicit and trace-conditioned execution signal improve steerability beyond directly relying on the generalist model for steering or verification without explicit design integrations?

We compare GT-VLA against three baselines with publicly released implementations. The base $\pi_{0.5}$ model [7], on which our method and the other baselines are built, directly maps vision-language inputs to actions. AtomicVLA [5] predicts which skill is being

executed at every step, and uses a MoE to learn each skill. SEAL [6] is a VLA steering framework using an external VLM verifier. We use Gemini 3.1 Pro to implement both SEAL and GT-VLA.

We additionally provide a best-effort reproduction of LoHo-Manip [16] on our project website; since no official implementation is available, we report it as a supplementary comparison.

Ablation Studies: We conduct ablation studies to evaluate two core design choices in GT-VLA: the use of a trace-generation stage, the TA-MoE architecture, and the three data augmentation techniques for trace-conditioned learning. Fig. 3 illustrates GT-VLA’s steerability, showing how GT-VLA adapts to unseen configurations compared to baseline and ablation methods.

A. Simulation Setup

We evaluate GT-VLA on LIBERO [30]. To evaluate generalizability to unseen tasks, we forgo per-suite in-distribution finetuning. Instead, we use a clean train-test split where methods are trained on Libero-10 and Libero-90 demonstrations [6] and evaluated on the Libero-Goal, Libero-Spatial, and Libero-Object suites [30]. We preprocess the dataset to obtain task decompositions, semantic targets, and ground truth traces. The annotation procedure is similar to approaches from recent work [5], [6], with details on our project website. We train all the methods for 100k iterations with batch size 64 on 4 GH200 GPUs.

B. Simulation Results

Table I reports the simulation results. GT-VLA achieves the best performance across all three LIBERO evaluation suites, with an average success rate of 43.8%. This corresponds to a 28.0 percentage

TABLE I: Success rate on LIBERO out-of-distribution evaluation suites, with 50 trials $\times$ 10 tasks per category (%). 95% Wilson confidence intervals are shown in parentheses.

Models	Goal	Spatial	Object	Avg.
$\pi_{0.5}$	16.2 (13.1–19.7)	29.6 (25.6–33.8)	1.6 (0.7–3.1)	15.8
AtomicVLA	30.2 (26.2–34.4)	46.8 (42.4–51.3)	24.6 (20.9–28.6)	33.9
SEAL	31.0 (27.0–35.3)	45.0 (40.6–49.5)	6.0 (4.1–8.5)	27.3
GT-VLA (ours)	38.8 (34.5–43.2)	50.8 (46.3–55.3)	41.8 (37.4–46.3)	43.8

point improvement over vanilla $\pi_{0.5}$, a 9.9 point improvement over AtomicVLA, and a 16.5 point improvement over SEAL.

Compared to GT-VLA, AtomicVLA also uses a skill-routed MoE to handle different skills, but does not support high-level generalist guidance or trace conditioning. The stronger performance of GT-VLA, particularly on LIBERO-Object, suggests that generalist information and trace guidance help performance when faced with novel objects. Although both SEAL and GT-VLA use an external generalist VLM, GT-VLA achieves substantially higher performance. This suggests that external guidance is not sufficient for steering: The VLA must also be designed to accept guidance effectively.

C. Real-world Experiment Setup

We further evaluate GT-VLA on a physical tabletop manipulation platform. The setup consists of a YAM robot arm equipped with side-view and wrist cameras. The workspace contains plates, baskets, and fake vegetables with different colors, shapes, and sizes. The tasks are long-horizon pick-and-place instructions such as “place the red bell pepper into the green plate”, typically involving two to six PICK and PLACE skills.

For real-world experiments, we compare against $\pi_{0.5}$ and AtomicVLA. We do not include SEAL because it requires a high-fidelity world model or digital twin for forward simulation, which is not available for our hardware setup. We collect 299 complete episodes for training, each containing two to six PICK and PLACE skills. Task definitions and dataset annotations are detailed on our project website. All methods are trained for 30k iterations with batch size 64 on 4 GH200 GPUs. Inference is performed on a single GH200 GPU.

To evaluate real-world generalization, we construct five evaluation suites. The first two are **composition** suites, where each task is composed of skill configurations that appear in training, but arranged in unseen orders. This tests whether the model can compose familiar behaviors into new long-horizon instructions. We evaluate 4- and 6-skill variants, denoted as **composition-4** and **composition-6**. The other three are **OOD** suites, where the skill configurations themselves are unseen. We evaluate 2-, 4-, and 6-skill variants, denoted **OOD-2**, **OOD-4**, and **OOD-6**, respectively. Each suite contains 10 tasks. For each task, we run 5 trials with different initial object placements, resulting in 50 trials for each method-suite pair. We report episode-level success: A trial is successful only if the robot completes the full task.

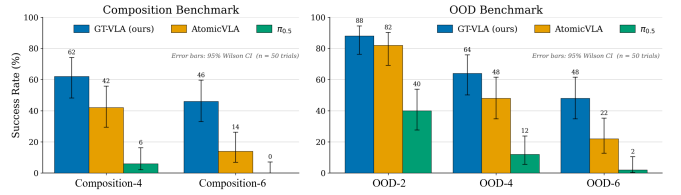


Fig. 5: Real-world results. 5 trials $\times$ 10 tasks per category.

TABLE II: Ablation results on LIBERO (%). Results use 50 trials $\times$ 10 tasks per category, except the generalist VLM swap rows, which use 20 trials $\times$ 10 tasks. Parentheses show 95% Wilson confidence intervals.

Models	LIBERO Goal	LIBERO Spatial	LIBERO Object	Avg.
GT-VLA-single	27.6 (23.7–31.7)	40.6 (36.3–45.0)	27.8 (23.9–32.0)	32.0
G-VLA	32.2 (28.1–36.5)	39.6 (35.3–44.0)	20.8 (17.3–24.6)	30.9
No S. Drop	38.4 (34.2–42.7)	29.0 (25.2–33.1)	24.4 (20.8–28.4)	30.6
No T. Drop	36.6 (32.5–40.9)	41.8 (37.6–46.2)	21.4 (18.0–25.2)	33.3
No Perturb.	34.8 (30.8–39.1)	40.6 (36.4–45.0)	23.8 (20.3–27.7)	33.1
GPT-5.6	33.0 (26.9–39.8)	44.5 (37.8–51.4)	35.0 (28.7–41.8)	37.5
Qwen 3.8	35.0 (28.7–41.8)	37.0 (30.6–43.9)	30.5 (24.5–37.2)	34.2
Gemini 3.6	31.0 (25.0–37.7)	46.5 (39.7–53.4)	33.5 (27.3–40.3)	37.0
GT-VLA (ours)	38.8 (34.5–43.2)	50.8 (46.3–55.3)	41.8 (37.4–46.3)	43.8

D. Real-world Experiment Results

Fig. 5 shows our real-world experiment results. GT-VLA outperforms baselines across 5 real-world suites, showing target-conditioned trace guidance’s effectiveness in real settings. GT-VLA’s advantage over AtomicVLA widens in both the composition and OOD settings as task horizon increases, supporting our hypothesis that complex manipulation requires both continuous semantic alignment and subtask decomposition.

E. Ablation Studies

We perform five ablation studies to isolate the contribution of the key components in GT-VLA: the trace generation stage, the MoE architecture, and the three data augmentation techniques. All ablations are evaluated in simulation on LIBERO following Sec. IV-A.

The *G-VLA* removes the trace module while keeping the semantic target guidance. We retain the same MoE action architecture and inject the target directly into the action heads using AdaRMS conditioning. This tests if converting the target into an explicit trace can provide additional benefit over using the target directly. The *GT-VLA-single* removes the skill-routed MoE, using only a single trace-generation head and a single action head. This tests the impact of using skill-specific experts to model each skill separately.

To evaluate the contributions from the three data augmentation techniques for trace-conditioned learning, we train a leave-one-out ablation for each augmentation. *No S. Drop* removes the random scene drop, *No T. Drop* removes the random trace drop, and *No Perturb.* removes the low-frequency trace perturbation.

Table II reports the ablation results. GT-VLA achieves substantially higher success rates compared to G-VLA. This suggests that directly injecting a semantic target into the action model is not enough, and the target is much more useful if it is converted into an actionable

trace that the policy can visually follow. This is also visualized in Fig. 3 (mid-left) where G-VLA fails to be steered by semantic guidance alone. Further, the gap between GT-VLA-single and GT-VLA indicates that the MoE structure is beneficial, substantially improving performance. Lastly, GT-VLA-single slightly outperforms G-VLA, suggesting that the trace guidance layer can overcome a lower parameter count.

Further, the three leave-one-out ablations show that each augmentation matters, as removing any one costs 10.5% - 13.2% average performance drop. The trace guidance yields limited benefit unless all three augmentations are present, suggesting the augmentations are complementary rather than redundant. The ablations, however, do not show a clear rank among them.

F. Pilot Studies on Noisy Semantic Guidance

Because GT-VLA relies on a generalist VLM to provide semantic target points for trace generation and downstream action execution, its performance is inherently tied to the accuracy of these targets. In real-world or out-of-distribution deployments, the generalist VLM may output imperfect or noisy target predictions. To evaluate GT-VLA’s robustness to such inaccuracies, we conducted an additional noise-injection benchmark.

In this evaluation, we systematically perturb the semantic targets generated by the VLM. For a given original target point, we define a noise radius r (in pixels) to form a bounding disk around it. We then uniformly sample a new coordinate from within this disk to serve as the perturbed target, ensuring the noise is not biased toward the original center. We test $r \in \{4, 8, 16, 32, 64\}$. Since the input observation image size is 224×224 pixels, the maximum perturbation radius of $r = 64$ encompasses over a quarter of the image width, representing severe spatial drift. Visual illustrations of these noise levels are provided in Fig. 6.

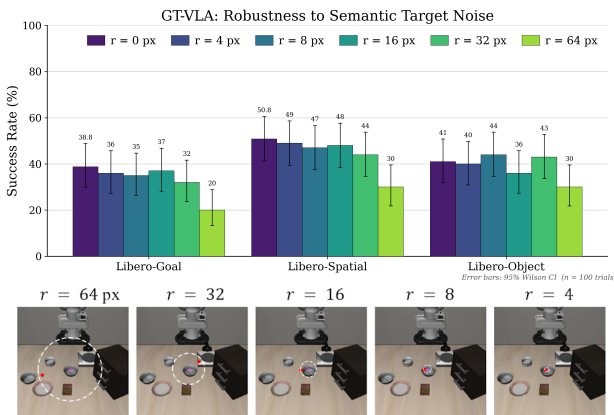


Fig. 6: **Semantic target perturbations.** Top: GT-VLA success rates across LIBERO suites when conditioned on noisy semantic targets. Bottom: Visual illustration of perturbations. Pink cross indicates given point, white circle indicates perturbation max. radius, red dot indicates perturbed final point.

We ran 100 trials for each noise radius across the three LIBERO out-of-distribution evaluation suites (see

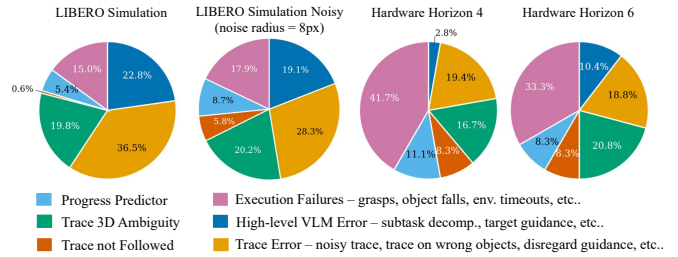


Fig. 7: GT-VLA failure breakdowns for simulation, hardware, and noise injections.

Sec. IV-A). The results, reported in Fig. 6, demonstrate that GT-VLA maintains robust performance for noise radii up to $r = 32$ pixels. Remarkably, even with this degree of perturbation, GT-VLA still generally outperforms the baselines presented in Table I.

The sharp performance drop observed at $r = 64$ pixels highlights a natural boundary for trace-conditioning robustness. As visualized in Fig. 6, perturbed targets at or below $r = 32$ pixels typically remain physically grounded on or immediately adjacent to the intended object. GT-VLA is robust to this localized noise because the target still provides a broadly correct semantic direction. However, at $r = 64$ pixels, the target point frequently drifts onto entirely incorrect objects or empty background space, misguiding the trace generation and significantly degrading performance.

G. Pilot Studies on Generalist VLM Swap

In principle, GT-VLA’s design allows it to take guidance from any high-quality off-the-shelf VLM generalist. To test this hypothesized modularity, we evaluate the impact of swapping the guidance VLM at test time to Gemini 3.6 Flash, Qwen 3.8 Max, and GPT-5.6 Terra, while freezing learned components. As shown in Table II, GT-VLA remains competitive with or outperforms the baselines in Table I, despite using guidance VLMs unseen during training. This also highlights the key benefit of GT-VLA over related works that leverage a finetuned VLM [16]. The benefit of using an off-the-shelf generalist VLM goes beyond finetuning cost: GT-VLA is not tied to a single generalist and may benefit from advances in frontier models out-of-the-box.

H. Failure Analysis

We assign each failed episode a single primary failure source, the earliest pipeline stage that deviated (Fig. 7). The results support three design choices. First, an off-the-shelf generalist suffices for sparse high-level guidance without fine-tuning. The high-level VLM and progress-prediction errors remain a minority error source across categories. Second, the trace-conditioned policy is reliably steerable. Failures due to not following the guidance trace range from 0.6% to 8.3% only, in contrast to G-VLA, which executes memorized motions despite correct targets (Fig. 3). Third, errors do not strongly cascade through the hierarchy and model components: perturbing the semantic target (Sec. IV-F)

and extending real-world tasks from four to six skills leave the failure profile largely unchanged, so noisier and longer tasks generally fail for the similar reasons rather than through new compounding failure modes.

The remaining failures lie downstream of the generalist and point to concrete improvements. In LIBERO, inaccurate traces are the largest source (36.5%). On hardware, low-level execution failures such as grasp slips dominate (41.7% for 4-skill tasks); these occur despite correct guidance and would benefit directly from stronger low-level controllers. Finally, 2D–3D ambiguity consistently accounts for around 20% of failures, indicating a limitation of image-space guidance itself and motivating multi-view or 3D traces.

V. CONCLUSION

We present GT-VLA, a VLA framework converting off-the-shelf VLM semantic guidance into actionable image-space traces for control. Ablations show that this intermediate trace interface surpasses direct target guidance, enhancing steerability and out-of-distribution performance. GT-VLA outperforms recent baselines on unseen tasks, novel configurations, and long-horizon task composition across simulated and real environments. In future work, we aim to incorporate 3D traces for more accurate guidance. Further, we note that our MoE closely follows [5], which supports continual expansion by adding a new expert and extending a new router while preserving existing experts. We expect GT-VLA’s design to support continual skill acquisition but this is left to future work.

REFERENCES

- [1] K. Black *et al.*, “ π_0 : A vision-language-action flow model for general robot control,” *arXiv preprint arXiv:2410.24164*, 2024.
- [2] M. J. Kim *et al.*, “Openvla: An open-source vision-language-action model,” in *Proceedings of The 8th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, P. Agrawal, O. Kroemer, and W. Burgard, Eds., vol. 270. PMLR, 2025, pp. 2679–2713. [Online]. Available: <https://proceedings.mlr.press/v270/kim25c.html>
- [3] S. Fei *et al.*, “Libero-plus: In-depth robustness analysis of vision-language-action models,” 2025. [Online]. Available: <https://arxiv.org/abs/2510.13626>
- [4] Z. Zhan *et al.*, “Stable language guidance for vision-language-action models,” 2026. [Online]. Available: <https://arxiv.org/abs/2601.04052>
- [5] L. Zhang *et al.*, “Atomicvla: Unlocking the potential of atomic skill learning in robots,” 2026, accepted at CVPR 2026. [Online]. Available: <https://arxiv.org/abs/2603.07648>
- [6] Y. Wu, A. Li, T. Hermans, F. Ramos, A. Bajcsy, and C. Pérez-D’Arpino, “Do what you say: Steering vision-language-action models via runtime reasoning-action alignment verification,” *2026 IEEE International Conference on Robotics and Automation (ICRA)*, 2026. [Online]. Available: <https://arxiv.org/abs/2510.16281>
- [7] K. Black *et al.*, “ $\pi_{0.5}$: a vision-language-action model with open-world generalization,” in *9th Annual Conference on Robot Learning*, 2025.
- [8] B. Zitkovich *et al.*, “Rt-2: Vision-language-action models transfer web knowledge to robotic control,” in *Conference on Robot Learning*. PMLR, 2023, pp. 2165–2183.
- [9] Physical Intelligence *et al.*, “ $\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities,” 2026. [Online]. Available: <https://arxiv.org/abs/2604.15483>
- [10] Z. Zhou *et al.*, “ChatVLA-2: Vision-language-action model with open-world reasoning,” in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. [Online]. Available: <https://openreview.net/forum?id=1lyKfIUOhp>
- [11] M. Zawalski, W. Chen, K. Pertsch, O. Mees, C. Finn, and S. Levine, “Robotic control via embodied chain-of-thought reasoning,” 2025. [Online]. Available: <https://arxiv.org/abs/2407.08693>
- [12] M. Nakamoto, O. Mees, A. Kumar, and S. Levine, “Steering your generalists: Improving robotic foundation models via value guidance,” in *8th Annual Conference on Robot Learning*, 2024. [Online]. Available: <https://openreview.net/forum?id=6FGlpzC9Po>
- [13] B. Häon, K. C. Stocking, I. Chuang, and C. Tomlin, “Mechanistic interpretability for steering vision-language-action models,” in *Proceedings of The 9th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, J. Lim, S. Song, and H.-W. Park, Eds., vol. 305. PMLR, 2025, pp. 2743–2762. [Online]. Available: <https://proceedings.mlr.press/v305/haon25a.html>
- [14] Z. Chen *et al.*, “Resteer: Quantifying and refining the steerability of multitask robot policies,” 2026. [Online]. Available: <https://arxiv.org/abs/2603.17300>
- [15] J. Kwok *et al.*, “Robomonkey: Scaling test-time sampling and verification for vision-language-action models,” in *Proceedings of The 9th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, J. Lim, S. Song, and H.-W. Park, Eds., vol. 305. PMLR, 2025, pp. 3200–3217. [Online]. Available: <https://proceedings.mlr.press/v305/kwok25a.html>
- [16] I. Liu *et al.*, “Long-horizon manipulation via trace-conditioned vla planning,” *arXiv preprint arXiv:2604.21924*, 2026.
- [17] J. Duan *et al.*, “Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation,” *ICLR 2025*, 2024.
- [18] Y. Wu, R. Tian, G. Swamy, and A. Bajcsy, “From foresight to forethought: Vlm-in-the-loop policy steering via latent alignment,” 2025. [Online]. Available: <https://arxiv.org/abs/2502.01828>
- [19] J. Liang *et al.*, “Code as policies: Language model programs for embodied control,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 9493–9500.
- [20] J. Shi *et al.*, “Maestro: Orchestrating robotics modules with vision-language models for zero-shot generalist robots,” in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. [Online]. Available: <https://openreview.net/forum?id=oykeBpHHXE>
- [21] N. Carion *et al.*, “Sam 3: Segment anything with concepts,” *arXiv preprint arXiv:2511.16719*, 2025.
- [22] J. Zhu *et al.*, “Bridging vlm and kmp: Enabling fine-grained robotic manipulation via semantic keypoints representation,” 2025. [Online]. Available: <https://arxiv.org/abs/2503.02748>
- [23] K. Fang, F. Liu, P. Abbeel, and S. Levine, “Moka: Open-world robotic manipulation through mark-based visual prompting,” *Robotics: Science and Systems (RSS)*, 2024.
- [24] D. Qu *et al.*, “Spatialvla: Exploring spatial representations for visual-language-action model,” 2025. [Online]. Available: <https://arxiv.org/abs/2501.15830>
- [25] C.-P. Huang *et al.*, “Fast-thinkact: Efficient vision-language-action reasoning via verbalizable latent planning,” *arXiv preprint arXiv:2601.09708*, 2026, accepted at CVPR 2026.
- [26] R. Zheng *et al.*, “Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies,” in *International Conference on Learning Representations*, 2025. [Online]. Available: <https://openreview.net/forum?id=b1CVu9l5GO>
- [27] J. Gu *et al.*, “Rt-trajectory: Robotic task generalization via hindsight trajectory sketches,” 2023.
- [28] C. Wen *et al.*, “Any-point trajectory modeling for policy learning,” *Robotics: Science and Systems*, 2024. [Online]. Available: <https://www.roboticsproceedings.org/rss20/p092.pdf>
- [29] J. Lee *et al.*, “Molmoact: Action reasoning models that can reason in space,” 2025. [Online]. Available: <https://arxiv.org/abs/2508.07917>
- [30] B. Liu *et al.*, “Libero: Benchmarking knowledge transfer for lifelong robot learning,” *arXiv preprint arXiv:2306.03310*, 2023.

A. Skill Definition

For the LIBERO domain, we defined 7 skills, in 5 categories:

- **PICKUP_FROM(object, location)**: Pick up an object from a surface or a container.
- **PLACE_ON / PLACE_IN(object, destination)**: Place an object in/onto a destination. These two language skills are routed to a single expert.
- **OPEN(container)**: Open a container (drawer or microwave).
- **CLOSE(container)**: Close a container (drawer or microwave).
- **TURN_ON / TURN_OFF(item)**: Turn an item on or off (stove knob). These two language skills are routed to a single expert.

This results in 5 experts for the LIBERO setting.

For our real-world experiments, we use the first 2 skill categories (PICKUP_FROM, PLACE_ON/IN), corresponding to two experts.

The discrete skill representation provides a compact interface between high-level reasoning and low-level execution: The skill name determines which trace expert and action expert are selected, while the natural language arguments specify the task-relevant objects or regions that the VLM backbone can understand.

B. Data Annotation Pipeline

Each demonstration consists of a high-level instruction ℓ and a trajectory of robot observations and actions. We first decompose ℓ into an ordered sequence of skills $P = (s^{(1)}, \dots, s^{(K)})$ using the same generalist-VLM procedure described in Sec. III-A. The demonstration is then partitioned into the corresponding contiguous skill segments. For each segment, the training instruction $\ell^{(i)}$ is formed by concatenating the episode instruction with the corresponding skill $s^{(i)}$.

For every timestep within a skill segment, the ground-truth remaining trace is obtained by projecting the future end-effector trajectory up to the end of that skill into image space. We additionally query Gemini 3.1 Pro for one semantic target $g^{(i)}$ per skill and associate this target with all samples in the corresponding segment. This produces the skill-annotated demonstrations used to train the trace and action experts.

C. Parameter Allocation Details

Following the mixture of experts implementation from AtomicVLA [5], we implement the expert weights as separate dense layers for each head. Each head ends up with some portion of shared weights (attention mechanism + AdaRMS conditioning), and some portion of weights that increases with the number of experts. SEAL [6] does not change the $\pi_{0.5}$ architecture, only the training and inference pipelines. AtomicVLA uses a number of experts based on skill categories, plus one additional expert as a “shared expert”.

TABLE III: Approximate parameter allocations for GT-VLA and baselines.

Part	Parameters
Vision encoder (Gemma) [1]	400M
Shared VLM (Gemma) [1]	2.6B
Action heads (shared)	200M
Action head (per expert)	230M
Trace heads (shared)	72M
Trace head (per expert)	57M
Progress predictor	0.6M
$\pi_{0.5}$ / SEAL (total)	3.4B
AtomicVLA (LIBERO; 5 experts + 1 shared)	4.6B
GT-VLA (LIBERO; 5 experts)	4.7B
AtomicVLA (real; 2 experts + 1 shared)	3.9B
GT-VLA (real; 2 experts)	3.8B

TABLE IV: Inference time for different components of GT-VLA.

Component	Runtime	Frequency
VLM planning	~ 7s	1/task
VLM guidance	~ 5s	1/skill
Trace generation	~ 52ms	Every 2 action chunks
Action generation	~ 74ms	N/A

D. Approximate Inference Time Latency Breakdown

Table IV provides a rough latency breakdown of the main inference components in GT-VLA. In our experiments, the generalist VLM calls for task planning and semantic guidance were queried through OpenRouter using gemini-3.1-pro-preview. Therefore, the absolute latency of the VLM components can vary with the queried model, serving platform, and network conditions. In contrast, trace generation and action generation are local model forward passes and are much faster than the external VLM queries.

The generalist VLM calls dominate the wall-clock latency, but they are invoked at relatively low frequency: planning is performed once per task, and semantic guidance is queried once per skill. The trace and action modules are invoked during closed-loop execution, and their runtimes are on the order of tens of milliseconds. If lower latency is required, one practical option is to replace the generalist VLM with a faster model or a lower-latency serving backend. This may trade semantic accuracy for inference speed. Nonetheless, the robustness study in Sec. IV-F suggests that GT-VLA can tolerate a moderate amount of noise in the predicted semantic target.

E. LoHo-Manip Reproduction

LoHo-Manip [16] is closely related to GT-VLA, as it also uses image-space trace guidance for long-horizon manipulation. Since no official implementation was available at the time of our evaluation, we provide a best-effort reproduction based on the architecture and training procedure described in the paper. We evaluate it under the same out-of-distribution protocol as Table I: the model is trained on LIBERO-10 and LIBERO-

TABLE V: Supplementary comparison with LoHo-Manip under the same LIBERO OOD protocol as Table I. Results use 50 trials $\times$ 10 tasks per suite (%). Parentheses show 95% Wilson confidence intervals. The GT-VLA row is repeated from Table I for direct comparison.

Model	Goal	Spatial	Object	Avg.
LoHo-Manip (reprod.)	30.8 (26.9–35.0)	23.2 (19.7–27.1)	12.8 (10.2–16.0)	22.3
GT-VLA (ours)	38.8 (34.5–43.2)	50.8 (46.3–55.3)	41.8 (37.4–46.3)	43.8

90, and evaluated without per-suite fine-tuning on LIBERO-Goal, LIBERO-Spatial, and LIBERO-Object.

For the high-level manager, we initialize from Qwen3-VL-4B-Instruct and fine-tune for three epochs, by which point the validation loss had plateaued. The vision encoder is frozen, while the language model and multimodal projector are fully fine-tuned using next-token cross-entropy on the generated response. Given the task instruction, current image observation, and a textual memory of completed subtasks, the manager predicts both the completed/remaining subtask split and a 20-waypoint 2D trace, with coordinates normalized to $[0, 1000]$, following the output representation in LoHo-Manip [16]. Based on validation loss, we selected the epoch-1.5 and epoch-2 manager checkpoints for downstream evaluation and report the epoch-1.5 checkpoint, which achieves the stronger aggregate performance across the three suites.

The $\pi_{0.5}$ -based low-level executor is trained for 100k steps with batch size 64, matching the training budget used for the baselines in Table I. We also apply the same trace augmentations used in GT-VLA during executor training since we found that to be helpful from our ablation studies (Sec. IV-E). Both the manager and executor are trained using 4 GH200 GPUs.

As shown in Table V, the reproduced LoHo-Manip achieves an average success rate of 22.3%, compared with 43.8% for GT-VLA. The difference is relatively smaller on LIBERO-Goal (30.8% vs. 38.8%) and substantially larger on LIBERO-Spatial (23.2% vs. 50.8%) and LIBERO-Object (12.8% vs. 41.8%). While this comparison is necessarily limited by the absence of an official LoHo-Manip implementation, the results provide an additional comparison against a closely related trace-conditioned hierarchical approach under our OOD evaluation protocol.

F. Real-world Experiment Training and Evaluation Tasks

1) *Training Task Setup*: The training dataset is split across 4 settings, with different objects on the table, described in Table VI. For each setting, 3 high-level tasks were demonstrated, described in Table VII. For each high-level task, the table was shuffled 5 times to mix up object positions, then the task was demonstrated 5 times for each shuffle. This gives $5 \times 5 \times 3 \times 4 = 300$ demonstrations nominally, with some human errors resulting in 299 total episodes.

2) *Evaluation Task Setup*: The **composition** suites evaluate the model’s ability to chain familiar skills

TABLE VI: Real-world training-data settings.

Setting	Objects
table1	orange plate, blue plate, basket, eggplant, corn, carrot, green bell pepper
table2	white plate, green plate, orange plate, eggplant, corn, carrot, green bell pepper, asparagus
table3	blue plate, green plate, red bell pepper, green bell pepper, corn
table4	white plate, basket, green bell pepper, carrot, eggplant, asparagus

into novel sequences. While all individual skill configurations are encountered during training, they are drawn from different training tasks to form completely unseen combinations. For example, Fig. 8 illustrates a **composition-6** task, which requires the execution of six previously learned skills in a novel sequence.

The **OOD** suites evaluate the model on tasks containing novel skill configurations not present in the training data. For example, Fig. 9 illustrates an **OOD-6** task requiring a sequence of six skills, where the configurations boxed in red are unseen skill configurations.

We evaluate each task over five episodes. Each episode initializes with random object placements and may include random visual distractors. To ensure fair comparisons across all methods, the object layouts are kept visually identical for each corresponding episode.

In the following, we present all **composition** and **OOD** task instructions. Visual demonstrations of select tasks can be found in the supplemental video.

Composition-4

- 1) *place the corn on the blue plate and the carrot on the white plate*
- 2) *put the eggplant into the basket and put the basket onto the blue plate*
- 3) *move the green bell pepper into the basket, and put the blue plate onto the basket*
- 4) *place the red bell pepper onto the blue plate and put the corn on the white plate*
- 5) *move the carrot onto the green plate and place the green bell pepper onto the green plate*
- 6) *place the carrot into the basket and place the red bell pepper onto the blue plate*
- 7) *put the eggplant on the white plate and put the carrot on the green plate*
- 8) *move the green bell pepper onto the green plate and place the asparagus onto the white plate*
- 9) *put the asparagus on the white plate and place the carrot into the basket*
- 10) *place the green bell pepper into the basket and put the red bell pepper onto the blue plate*

Composition-6

- 1) *put the corn on the blue plate, and put both the carrot and the asparagus onto the white plate*
- 2) *put the eggplant and the green bell pepper into the basket, and put the basket onto the blue plate*
- 3) *put the eggplant and the green bell pepper in the basket, and place the blue plate onto the basket*

TABLE VII: Real-world training tasks. Objects are initially placed on the table unless otherwise specified; “–” denotes no additional initial-state constraint.

ID	Task Instruction	Setting	Initial State
0	move the basket onto the blue plate and put the green bell pepper in the basket	table1	–
1	move the orange plate onto the basket	table1	–
2	move the eggplant into the basket and place the blue plate on the basket	table1	eggplant on blue plate
3	move the green plate onto the orange plate and move the asparagus and corn onto the white plate	table2	–
4	put the carrot onto the green plate	table2	green bell pepper on green plate
5	move the eggplant and carrot onto the white plate	table2	corn on white plate
6	put the blue plate on the green plate and place the corn and red bell pepper on the blue plate	table3	–
7	place the green bell pepper on the green plate	table3	green plate on blue plate
8	put the red bell pepper onto the blue plate and move the green bell pepper and corn to the green plate	table3	green bell pepper on blue plate
9	place the eggplant, asparagus, and carrot onto the white plate	table4	–
10	move the carrot, eggplant, and green bell pepper into the basket	table4	–
11	place the eggplant and green bell pepper onto the white plate	table4	–

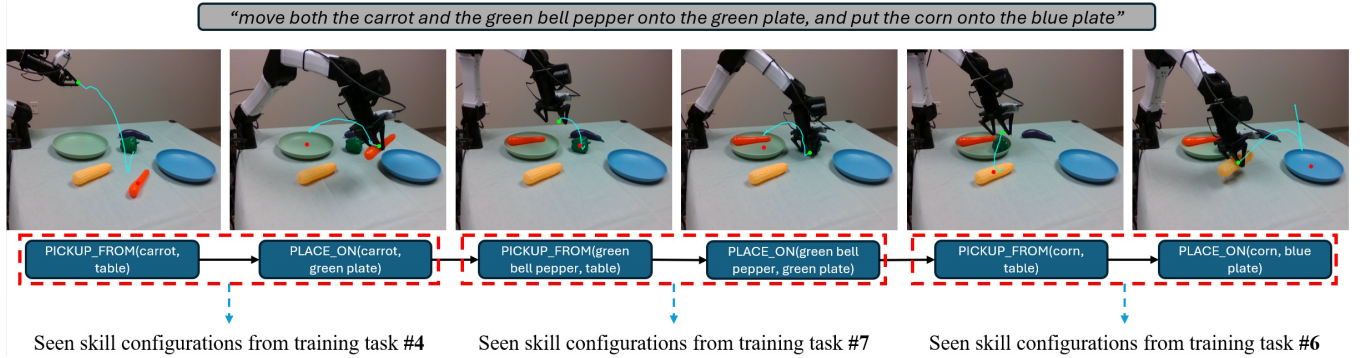


Fig. 8: Visual illustration of a task setup from the **composition-6** test suite. The seen skill configurations identified in the figure can be found in Table VII.

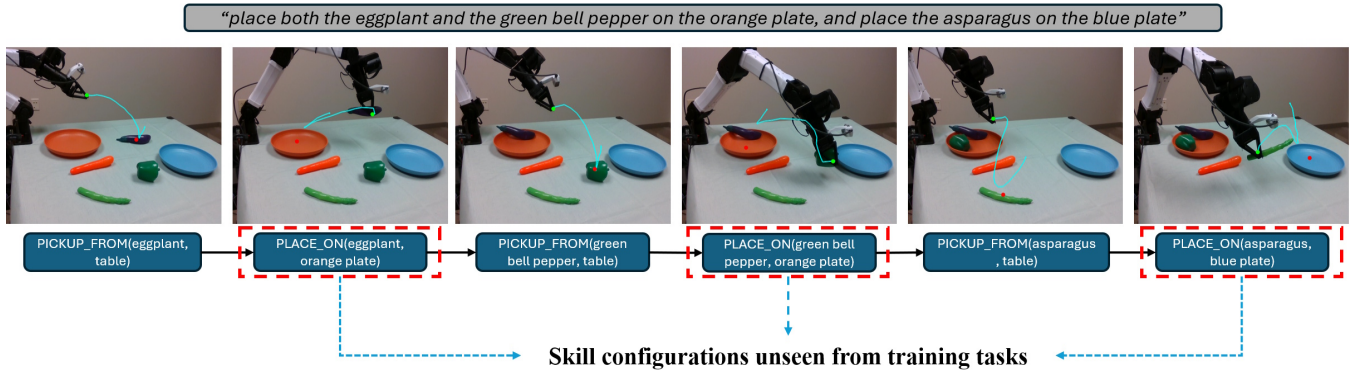


Fig. 9: Visual illustration of a task setup from the **OOD-6** test suite. The boxed skill configurations are unseen from the training dataset.

- | | |
|--|---|
| <p>4) place the eggplant and the corn on the white plate, and put the red bell pepper onto the blue plate</p> <p>5) move both the carrot and the green bell pepper onto the green plate, and put the corn onto the blue plate</p> <p>6) place the carrot into the basket, and place the corn and the red bell pepper onto the blue plate</p> | <p>7) put the corn and the carrot on the green plate, and place the eggplant onto the white plate</p> <p>8) move the green bell pepper onto the green plate, and place both the carrot and the asparagus onto the white plate</p> <p>9) move both the green bell pepper and the asparagus</p> |
|--|---|

onto the white plate, and put the carrot into the basket

- 10) *put the corn and the red bell pepper on the blue plate, and place the green bell pepper into the basket*

OOD-2

- 1) *place the blue plate on the orange plate*
- 2) *put the eggplant on the green plate*
- 3) *place the green bell pepper on the orange plate*
- 4) *put the asparagus onto the green plate*
- 5) *place the corn on the orange plate*
- 6) *place the carrot onto the orange plate*
- 7) *place the red bell pepper onto the orange plate*
- 8) *put the orange plate on the white plate*
- 9) *place the green plate on the blue plate*
- 10) *put the eggplant onto the blue plate*

OOD-4

- 1) *place the carrot onto the blue plate and put the corn into the basket*
- 2) *put both the green bell pepper and the eggplant on the blue plate*
- 3) *place the eggplant on the green plate and put the asparagus in the basket*
- 4) *place the asparagus on the green plate and put the corn on the orange plate*
- 5) *place the red bell pepper on the green plate and put the carrot on the orange plate*
- 6) *place the green plate on the white plate and place the red bell pepper on the green plate*
- 7) *place the orange plate on the green plate and put the blue plate on the orange plate*
- 8) *put both the green bell pepper and the carrot on the orange plate*
- 9) *place the red bell pepper into the basket, and put the corn into the basket*
- 10) *place the eggplant on the orange plate, and put the red bell pepper on the white plate*

OOD-6

- 1) *put both the eggplant and the green bell pepper on the blue plate, and place the corn into the basket*
- 2) *put both the eggplant and the red bell pepper on the green plate, and place the asparagus into the basket*
- 3) *put the carrot on the blue plate and place both the corn and the eggplant onto the orange plate*
- 4) *place both the eggplant and the green bell pepper on the orange plate, and place the asparagus on the blue plate*
- 5) *place both the red bell pepper and the corn into the basket, and place the carrot onto the blue plate*
- 6) *put the corn on the orange plate, and put the eggplant onto the green plate, and place the green bell pepper on the orange plate*
- 7) *place the asparagus and the carrot on the blue plate, and place the eggplant on the green plate*
- 8) *put the orange plate onto the blue plate, and place both the asparagus and the corn on the orange plate*

- 9) *place the corn on the orange plate, and put both the eggplant and the carrot on the blue plate*
- 10) *place the red bell pepper on the white plate, and place both the asparagus and the eggplant onto the green plate*

G. Prompt Templates

1) *Skill Planning Prompt Template:* As described in Sec. III-A, given a task instruction ℓ and the current observation o_t , we query an off-the-shelf generalist VLM Φ to produce a skill plan using the following prompt template, along with the side-view image.

```
You are planning a tabletop robot manipulation task.
You are given:
- the natural-language task instruction for this episode
- the observation of the INITIAL scene

Your task:
1. Decide the ordered sequence of atomic skills that, executed in order,
   will accomplish
   the instruction from the visible initial state.
2. Use ONLY the allowed skills below and follow the syntax exactly.

Task instruction: {task_instruction}

{SKILL_DEFINITIONS_TEXT}

Planning rules:
- The first skill must be PICKUP_FROM - at the start the gripper holds
  nothing, so it
  cannot PLACE_ON or PLACE_IN.
- A PLACE_ON / PLACE_IN of an object must come right after the PICKUP_FROM
  of that object
  (pick, then place, then pick the next object).
- Object descriptions must be short, specific, contain no commas, and
  reuse the
  appearance / color cues from the instruction (e.g. "green bell pepper",
  "blue plate").
- Keep the plan as short as possible while still completing the
  instruction.

Output format:
- Return JSON only.
- "plan" must be a single numbered string of the exact form:
  "1. PICKUP_FROM(green bell pepper, table) 2. PLACE_ON(green bell pepper,
  blue plate)"

Example:
{"reasoning": "the carrot is on the table and must go on the blue plate;
the corn goes in the basket",
"plan": "1. PICKUP_FROM(carrot, table) 2. PLACE_ON(carrot, blue plate)
3. PICKUP_FROM(corn, table) 4. PLACE_IN(corn, basket)"}

```

2) *Target Acquisition Prompt Template:* GT-VLA queries the generalist model for a semantic target at the start of each skill. Given the current skill, the generalist model provides a target in the normalized image coordinates. We use the following prompt for target acquisition.

```
You are annotating a tabletop robot manipulation skill from the
observation of the current scene.

Episode instruction: {task_instruction}
Current skill (the next one to execute):
- skill: {skill_text}
- sent image size: width={image_width}, height={image_height}

Semantic hints:
1. For PICKUP_FROM, point at the center of the visible body of the object
to be picked up.
2. For PLACE_ON / PLACE_IN, point at the receiving surface or the inside
of the receiving container.

Coordinate rules:
- Return coordinates on a fixed output grid width={coordinate_grid},
height={coordinate_grid}.
- The full image spans x=0..{coordinate_grid} left-to-right and y=0..{
coordinate_grid} top-to-bottom.
- Return point_2d in row-column order: [y, x], not [x, y].
- The point must be on the visible object (or contact location), strictly
inside the image bounds.
- "label" should be the literal string "semantic_target".
- Return JSON only - no markdown, prose, or code fences outside the JSON
object.

Output shape:
{"status": "OK",
"reasoning": "brief visual justification",
"label": "semantic_target",
"point_2d": [y, x]}
```