LEXICONVLA: LEARNING REUSABLE ATOMIC ACTION CODEBOOKS FOR UNSEEN TASKS

Anonymous authors
Paper under double-blind review

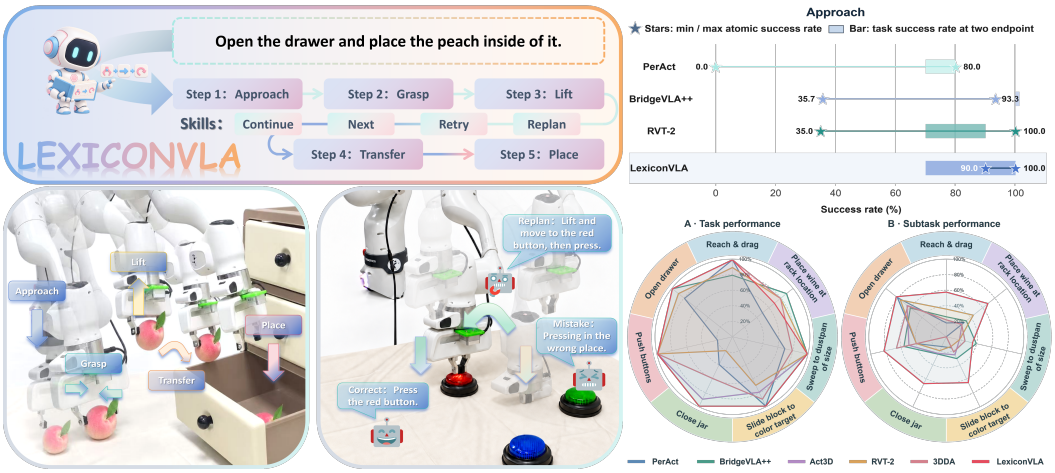


Figure 1: **LexiconVLA and Motivation. Left:** LexiconVLA decomposes an instruction into atomic-action subtasks, executes each with a VLA policy conditioned on codes from a shared codebook, and advances, retries, or replans according to execution feedback; real-robot rollouts show step-wise execution and recovery from a wrong button press (Bottom left). **Right:** A diagnostic probe on seven tasks, where each atomic subtask is prompted alone from the demonstration state. **Top right:** Lowest and highest cross-task success of the contact-free action - *approach* (stars), with full-task success on the two corresponding tasks (bars). **Bottom right:** Full-task success (A) versus mean atomic-subtask success (B). For existing policies, high full-task success does not translate into reliable or task-consistent atomic execution.

ABSTRACT

Vision-language-action (VLA) models struggle to reuse recurring interactions in unseen tasks. Our diagnostic study reveals that reliable task completion does not imply consistent execution of constituent atomic actions across task contexts. We present *LexiconVLA*, a retrievable atomic-action lexicon for cross-task reuse. Global and detail codebooks capture shared interaction structure and fine-grained execution variation, respectively, preserving both reusable patterns and execution details. Visual-Atomic Action Alignment couples trajectory reconstruction from visual state changes with visual outcome prediction from action codes, grounding the lexicon in motion and its effects. We learn these codebooks with trajectory reconstruction and visual alignment on our *AtomAction* Dataset of 57,803 segments from 69 tasks. A planner and scene-aware adapter translate new goals into code-conditioned subtasks for a shared policy, without skill-specific experts or deployment-time parameter updates. Across five policy backbones on 26 RLBench tasks, *LexiconVLA* largely maintains performance on 18 seen tasks while improving success on 8 tasks held out from policy training. With BridgeVLA, unseen-task success rises from 16.67% to 34.17% (+17.50 percentage points), and overall success reaches 71.08%, the highest among methods with reported results. Real-robot experiments demonstrate stepwise execution and failure recovery.

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

1 INTRODUCTION

VLA models enable instruction-guided manipulation by mapping visual observations and language instructions end-to-end to robot actions (Brohan et al., 2023; Kim et al., 2024; Black et al., 2026). Although scalable VLA frameworks yield powerful multi-task policies, their performance degrades sharply on out-of-distribution tasks. Such generalization failure does not stem from unfamiliar motion patterns required by novel tasks. Essentially, diverse manipulation tasks share a set of fundamental behavioral primitives, termed atomic actions, including object approaching, grasping, and transferring, as shown in Figure 1. These basic interactive units recur across different tasks, despite variations in execution scenarios, target objects, and sequential arrangements. Accordingly, unseen manipulation tasks can be essentially regarded as new combinations of learned atomic actions. Based on this insight, the core challenge of cross-task generalization lies in the acquisition and structured organization of reusable behavioral primitives. This motivates a critical question: **How can prior manipulation experience be leveraged to learn generalizable atomic-action representations that a unified policy can retrieve and recombine to accomplish novel tasks?**

To investigate whether existing policies acquire reusable atomic actions, we conduct a diagnostic probe across five pretrained manipulation policies of diverse architectures across seven tasks, contrasting full-task success against the performance of each atomic constituent executed alone from its demonstration start state (Appendix A.1). High full-task success does not guarantee reliable atomic execution: across policies, task-averaged success drops from 65.7–95.7% on complete tasks to 28.5–38.9% on isolated atomic actions (Figure 1, bottom right), and even the common non-contact primitive approach varies widely across task contexts within a single policy (top right). These diagnostic results indicate that task-level training alone fails to yield atomic actions that can be reliably invoked independently—a prerequisite for recombining them to solve unseen tasks. This motivates atomic-level supervision and explicit action representations, enabling policies to retrieve shared interaction patterns while adapting execution to scene-specific conditions.

Recent atomic-skill methods (Zhang et al., 2026b; Sun et al., 2026) emphasize skill-specialized execution or subtask-level policy optimization using predicted outcomes as rewards. In contrast, we organize recurring interactions into representations that a shared policy can retrieve across tasks. To this end, *LexiconVLA* learns an atomic-action codebook from interaction-phase trajectories and their observed visual effects. Quantization allows different trajectory instances to share a finite set of entries, providing a common action vocabulary rather than a separate representation for every instance. A scene-aware adapter retrieves these codes from the current observation and atomic subtask instruction, making trajectory-derived knowledge accessible without demonstrations at deployment. The retrieved codes condition the shared policy, while a planner sequences subtasks for new objectives, requiring neither skill-specific execution experts nor deployment-time parameter updates.

Learning a reusable action lexicon requires supervision that exposes segment-level structure shared across tasks, yet full demonstrations entangle multiple interaction phases and task-level instructions do not mark atomic-action boundaries. We therefore construct the *AtomAction* Dataset, comprising 57,803 annotated atomic-action segments from 69 tasks, each paired with its trajectory, boundary observations, and language descriptions. To capture segment-level structure while retaining execution details, we use global and detail codebooks: a global code summarizes each segment, while detail codes provide complementary information for trajectory reconstruction. Reconstruction alone provides limited supervision about an action's effects on the environment. We thus introduce Visual-Atomic Action Alignment (V3A), coupling action reconstruction from visual changes with final-observation feature prediction, conditioned on the initial observation and the global action code. These objectives encourage action-relevant visual features and action codes predictive of observable effects, grounding the lexicon in both motion and its consequences.

Our contributions are threefold:

- We introduce *AtomAction*, a dataset comprising 57,803 annotated atomic-action segments across 69 manipulation tasks. AtomAction pairs segmented interaction trajectories with boundary observations and textual descriptions. We further present a diagnostic analysis revealing a substantial gap between full-task completion and reliable standalone execution of constituent atomic actions.

- We propose *LexiconVLA*, a framework that distills recurring interaction phases into shared global and detail codebooks, grounded in their associated state transitions through Visual-Atomic Action Alignment (V3A). A planner schedules atomic subtasks and a scene-aware

adapter retrieves their codes to condition an existing VLA, forming a closed loop that composes atomic actions for unseen tasks and recovers from failures.

- Extensive experiments across five policy backbones show that LexiconVLA improves unseen-task success while largely preserving seen-task performance. With BridgeVLA, unseen-task success rises from 16.67% to 34.17% and overall success from 67.28% to 71.08%, the highest among evaluated methods. Real-robot rollouts further demonstrate sequential atomic-action execution and recovery from execution errors.

2 Related Works

2.1 Vision-Language-Action Models

Vision-language-action (VLA) models build robot policies on pretrained vision-language models (VLMs) and large multi-robot datasets (Team et al., 2024; Kim et al., 2024; Intelligence et al., 2025). Their action interfaces have evolved from autoregressively decoded discrete tokens (Brohan et al., 2023; Pertsch et al., 2025; Kim et al., 2025) to diffusion and flow-matching heads that generate continuous action chunks (Chi et al., 2024; Cheang et al., 2025; Black et al., 2026; Chen et al., 2026c), while keyframe-based 3D policies attain strong multi-task manipulation performance (Goyal et al., 2024; Li et al., 2025; 2026). Across these designs, generalization derives largely from the pretrained backbone and training data. Although VLAs transfer to novel objects, scenes, and instruction phrasings, stronger general VLM capability does not reliably translate into better control (Zhang et al., 2026a), and perturbation studies indicate that high benchmark success can reflect memorized trajectories rather than task understanding (Fei et al., 2025; Zhou et al., 2026). Recent work adds spatial representations (Qu et al., 2025; Zhang et al., 2025), subtask reasoning (Zawalski et al., 2025; Zhao et al., 2025), and predictive world models (Hu et al., 2025; Ye et al., 2026), yet still learns the mapping from instruction to motion at the task level, leaving behaviors shared across tasks implicit in policy parameters and difficult to invoke independently. LexiconVLA instead makes these behaviors explicit as retrievable atomic-action codes that condition a shared policy.

2.2 Learning Reusable Action Representations

Existing action representations primarily differ in the unit they encode. Action tokenizers (Lee et al., 2024; Zhou et al., 2025b) discretize individual motions or fixed-length segments for reconstruction and generation fidelity, without explicit alignment to recurring atomic actions. Latent action models infer actions from visual changes between unlabeled video frames (Bruce et al., 2024; Ye et al., 2025), enabling pretraining across embodiments while also capturing task-irrelevant dynamics. Task-centric variants suppress these dynamics through language conditioning (Bu et al., 2025), but still represent short frame windows rather than complete interactions. Skill-level methods (Zhang et al., 2026b; Sun et al., 2026) operate over longer horizons through specialized experts or subtask rewards during post-training. These skills are typically defined by goal-level labels that group mechanically distinct motions under terms such as *open*, and lack a shared representation that a common policy can retrieve. LexiconVLA addresses this by representing interaction phases, jointly supervised by trajectory reconstruction and observed state changes, to distinguish mechanisms that share a goal label and capture what each action changes.

3 AtomAction Dataset

Learning a discrete vocabulary of atomic actions requires supervision at a granularity that existing manipulation corpora do not provide: most VLA datasets (James et al., 2020; Liu et al., 2023; Zhou et al., 2026; Chen et al., 2026b) pair a complete demonstration with a single task instruction, so one trajectory spans several distinct interactions, while automatically derived atomic annotations remain coarse, a label such as *open* subsuming translational pulling, outward hinge rotation, and lid flipping (Zhang et al., 2026b). Segments sharing such a label stay heterogeneous in motion structure, and a codebook learned from them cannot separate mechanisms that a policy must execute differently. We therefore construct the **AtomAction Dataset**, in which every sample is a temporally localized segment realizing exactly one mechanism-level atomic action. Built from RLBench (James et al., 2020) expert demonstrations, it comprises 57,803 segments from 69 tasks spanning 18 action types (Figure 2). Each sample retains the low-dimensional state trajectory of the phase, five-view RGB, depth, and mask observations at its boundaries, the atomic-action label, and eight phase-level descriptions in different language styles, illustrated by the *flip-open* phase of *open*

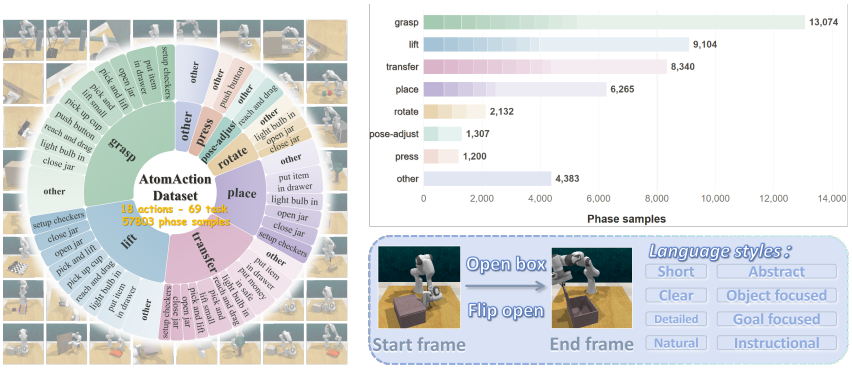


Figure 2: **The AtomAction Dataset. Left:** Atomic-action categories (inner ring) and contributing tasks (outer ring). **Top right:** Segment counts per category; coloured block widths show individual task contributions, matching the outer ring. **Bottom right:** Example segment with start/end frames and language descriptions.

box in the figure. The construction and annotation pipeline and atomic-action standard are detailed in the Appendices A.6 and A.11.

4 METHOD

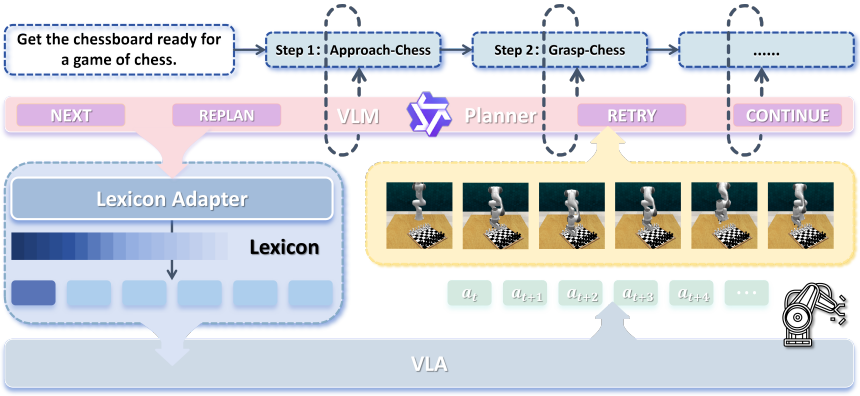


Figure 3: **Overview of LexiconVLA.** A VLM-based planner decomposes the task instruction into atomic subtasks. Given the active subtask instruction and current observations, a scene-aware adapter retrieves entries from the frozen atomic-action lexicon—comprising global and detail codes—to condition a shared VLA policy. Execution feedback drives subtask progression and recovery through four skills: NEXT, CONTINUE, RETRY, and REPLAN. Subtask transitions and replanning refresh the retrieved codes based on updated observations.

4.1 OVERVIEW

LexiconVLA conditions a shared manipulation policy on a frozen atomic-action lexicon of global and detail codes (Figure 3). We describe execution (Sec. 4.2), followed by lexicon learning with the Vector-Quantized Atomic Action Model (VQAA; Sec. 4.3 and Figure 4). VQAA combines Atomic Action-NSVQ (Sec. 4.3.1) for codebook learning via trajectory reconstruction with Visual-Atomic Action Alignment (V3A; Sec. 4.3.2) for grounding codes in observed state changes.

4.2 LEXICONVLA

Given the task instruction and current observations, a VLM planner generates atomic subtask instructions. A scene-aware adapter maps the active instruction ℓ_i and current images to codebook indices, retrieving $\mathbf{z}_g^{(i)}$ and $\mathbf{Z}_d^{(i)}$ as the corresponding lexicon entries. A lightweight injection module uses these codes to condition the shared VLA policy:

$$\mathbf{b}_t \sim \pi_\theta\Big(\cdot \mid O_t, \mathbf{s}_t, \ell_i, \mathbf{z}_g^{(i)}, \mathbf{Z}_d^{(i)}\Big), \tag{1}$$

where O_t, $\mathbf{s}_t$, and $\mathbf{b}_t$ denote visual observations, robot state, and the end-effector control command, respectively. The adapter learns this mapping using code indices assigned to demonstration trajectories by the frozen action encoder and codebooks (Appendix A.2). Within a subtask, the retrieved codes stay fixed while the policy receives updated observations. The planner assesses visual feedback and selects among four scheduling skills: NEXT advances after subtask completion; CONTINUE retains the active subtask when further execution is required; RETRY repeats a failed step, optionally returning to an earlier subtask; and REPLAN regenerates the plan when scene changes invalidate it. Monitoring rules and planner prompts are detailed in Appendices A.4 and A.12.

4.3 Vector-Quantized Atomic Action Model

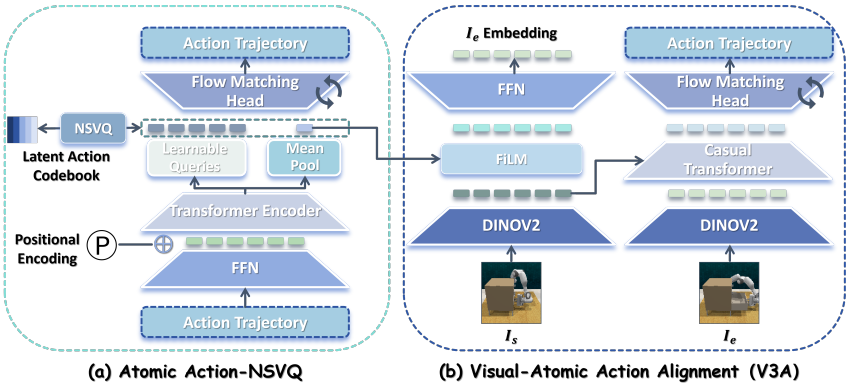


Figure 4: **Overview of the Vector-Quantized Atomic Action Model (VQAA). (a)** Atomic Action-NSVQ encodes an atomic action segment into discrete codes from two learnable codebooks using Noise Substitution in Vector Quantization (NSVQ) (Vali & Bäckström, 2022). A conditional flow matching decoder provides trajectory reconstruction supervision for learning the codebooks. **(b)** Visual-Atomic Action Alignment (V3A) incorporates visual context through two auxiliary objectives: reconstructing actions from initial-to-final visual changes and predicting final-frame features conditioned on the initial observation and the global action code.

4.3.1 Atomic Action-NSVQ

Trajectory encoding. To capture differences in joint configuration, motion dynamics, and contact conditions that are not specified by a 6-DoF end-effector pose, we augment pose information with joint and gripper measurements, providing richer supervision for codebook learning. Let $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_T]^\top \in \mathbb{R}^{T\times 39}$ denote an atomic action segment of length T from AtomAction Dataset, where each timestep is represented as

$$\mathbf{a}_t = \Big[\mathbf{p}_t,\ \mathbf{r}_t,\ \mathbf{q}_t,\ \dot{\mathbf{q}}_t,\ (\mathbf{f}_t^{\text{joint}}),\ (\mathbf{q}_t^{\text{grip}}),\ (\mathbf{f}_t^{\text{grip}}),\ g_t\Big]^\top \in \mathbb{R}^{39}. \tag{2}$$

Here, $\mathbf{p}_t \in \mathbb{R}^3$ and $\mathbf{r}_t \in \mathbb{R}^6$ encode end-effector position and rotation; $\mathbf{q}_t, \dot{\mathbf{q}}_t, \mathbf{f}_t^{\text{joint}} \in \mathbb{R}^7$ are joint positions, velocities, and torques; $\mathbf{q}_t^{\text{grip}} \in \mathbb{R}^2$, $\mathbf{f}_t^{\text{grip}} \in \mathbb{R}^6$, and $g_t \in \{0, 1\}$ specify gripper joint positions, contact forces, and state. After field-specific preprocessing (Appendix A.5), we separately embed the end-effector, joint, gripper-measurement, and gripper-state groups and concatenate their embeddings. A channel encoder E_{ch} integrates these features through gating and feed-forward layers, followed by a bidirectional Transformer $E_{\text{temp}}^{\text{RoPE}}$ with rotary positional embeddings (Su et al., 2021):

$$\mathbf{H} = E_{\text{temp}}^{\text{RoPE}}(E_{\text{ch}}(\Phi(\mathbf{A}))) \in \mathbb{R}^{T\times d}, \tag{3}$$

where Φ denotes the grouped feature projections and d is the hidden dimension. RoPE (Su et al., 2021) encodes relative temporal positions in self-attention to produce contextualized features $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_T]^\top$, with $\mathbf{h}_t \in \mathbb{R}^d$. Padding positions are excluded throughout encoding and aggregation.

Global and detail representations. Pooling $\mathbf{H}$ into a single vector may discard execution details needed for reconstruction. We therefore complement a global feature obtained by masked average pooling with detail features extracted by learnable queries attending to $\mathbf{H}$:

$$\bar{\mathbf{h}}_g = \frac{\sum_{t=1}^{T} m_t \mathbf{h}_t}{\sum_{t=1}^{T} m_t} \in \mathbb{R}^d, \qquad \bar{\mathbf{H}}_d = \text{CrossAttn}\left(\mathbf{Q}_d, \mathbf{H}, \mathbf{H}; \mathbf{m}\right) \in \mathbb{R}^{N_d\times d}. \tag{4}$$

Here, $m_t \in \{0, 1\}$ indicates a valid timestep, and $\mathbf{Q}_d \in \mathbb{R}^{N_d \times d}$ contains N_d learnable queries. Pooling summarizes the segment, while each detail query can emphasize different temporal information. Separate linear projections map both feature types to a common code dimension d_c:

$$\mathbf{u}_g = P_g(\bar{\mathbf{h}}_g) \in \mathbb{R}^{d_c}, \qquad \mathbf{U}_d = P_d(\bar{\mathbf{H}}_d) \in \mathbb{R}^{N_d \times d_c}. \tag{5}$$

We quantize $\mathbf{u}_g$ using a global codebook $\mathcal{C}_g$ and each row of $\mathbf{U}_d$ using a shared detail codebook $\mathcal{C}_d$, yielding one global code and N_d detail codes per segment.

Noise-substituted vector quantization. To learn these discrete representations through reconstruction, we use Noise Substitution in Vector Quantization (NSVQ) (Vali & Bäckström, 2022). For a projected feature $\mathbf{u} \in \mathbb{R}^{d_c}$ and its corresponding codebook $\mathcal{C} = \{\mathbf{e}_k\}_{k=1}^{K}$, the discrete representation is the nearest codebook vector:

$$k^\star = \underset{1 \le k \le K}{\arg\min} \|\mathbf{u} - \mathbf{e}_k\|_2^2, \qquad \mathbf{z} = \mathbf{e}_{k^\star}. \tag{6}$$

To address the non-differentiability of nearest-neighbor selection, NSVQ replaces $\mathbf{z}$ with a noise-perturbed feature during training:

$$\tilde{\mathbf{z}} = \mathbf{u} + \frac{\|\mathbf{u} - \mathbf{e}_{k^\star}\|_2}{\|\boldsymbol{\xi}\|_2 + \varepsilon}\boldsymbol{\xi}, \qquad \boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{d_c}), \tag{7}$$

where $\varepsilon > 0$ ensures numerical stability. The noise scales with the quantization error, enabling reconstruction gradients to update both the encoder and the selected codebook vector without a straight-through estimator (STE). Applying this operation to the two branches yields $\tilde{\mathbf{z}}_g \in \mathbb{R}^{d_c}$ and $\tilde{\mathbf{Z}}_d \in \mathbb{R}^{N_d \times d_c}$, which jointly condition the trajectory decoder.

Conditional trajectory reconstruction. A conditional flow-matching decoder reconstructs $\mathbf{A}_{\text{ee}} \in \mathbb{R}^{T \times 9}$, comprising normalized positions and 6D rotations. The global code and flow-time embedding modulate the decoder; detail codes condition it through cross-attention. For $s \sim \mathcal{U}(0, 1)$ and Gaussian noise $\boldsymbol{\epsilon}$, define $\mathbf{A}_{\text{ee}}^{(s)} = (1 - s)\boldsymbol{\epsilon} + s\mathbf{A}_{\text{ee}}$ and target velocity $\dot{\mathbf{A}}_{\text{ee}}^{(s)} = \mathbf{A}_{\text{ee}} - \boldsymbol{\epsilon}$. Following Lipman et al. (2023), we minimize

$$\mathcal{L}_{\text{FM}}^{b} = \mathbb{E}_{\mathbf{A}, s, \boldsymbol{\epsilon}, \boldsymbol{\xi}}\left[\left\| \left[v_\theta\left(\mathbf{A}_{\text{ee}}^{(s)}, s \mid \tilde{\mathbf{z}}_g, \tilde{\mathbf{Z}}_d\right) - \dot{\mathbf{A}}_{\text{ee}}^{(s)} \right]_b \right\|_{\mathbf{m}}^2\right], \quad b \in \{\text{pos}, \text{rot}\}, \tag{8}$$

where $[\cdot]_b$ selects position or rotation components and $\|\mathbf{X}\|_{\mathbf{m}}^2 = \sum_t m_t \|\mathbf{X}_t\|_2^2 / \sum_t m_t$ averages over valid timesteps. A separate gripper head uses masked binary cross-entropy $\mathcal{L}_{\text{grip}}$, while $\mathcal{L}_{\text{sep}} = \mathbb{E}[|\cos(\bar{\mathbf{h}}_g, N_d^{-1} \sum_n \bar{\mathbf{h}}_{d,n})|]$ discourages redundancy between global and detail features, with $\bar{\mathbf{h}}_{d,n}$ denoting the n-th detail feature. The combined objective is

$$\mathcal{L}_{\text{AA}} = \mathcal{L}_{\text{FM}}^{\text{pos}} + \lambda_{\text{rot}}\mathcal{L}_{\text{FM}}^{\text{rot}} + \lambda_{\text{grip}}\mathcal{L}_{\text{grip}} + \lambda_{\text{sep}}\mathcal{L}_{\text{sep}}. \tag{9}$$

The decoder is used only during codebook training.

4.3.2 Visual-Atomic Action Alignment

Trajectory reconstruction encourages the codebooks to retain motion information but provides limited supervision on objects and environmental changes. Visual-Atomic Action Alignment (V3A) adds visual supervision through action grounding from paired observations and global-code-conditioned prediction of final-frame features.

Action grounding from visual changes. Given the initial and final observations of an atomic action segment from AtomAction Dataset, a shared DINOv2 (Oquab et al., 2024) encoder extracts patch features $\mathbf{F}_{\text{start}}, \mathbf{F}_{\text{end}} \in \mathbb{R}^{P \times d_v}$, where P is the number of patches and d_v is the feature dimension. We combine cross-attention with the projected feature difference to encode visual changes:

$$\mathbf{F}_\Delta = E_\Delta(\text{CrossAttn}(\mathbf{F}_{\text{end}}, \mathbf{F}_{\text{start}}, \mathbf{F}_{\text{start}}) + P_\Delta(\mathbf{F}_{\text{end}} - \mathbf{F}_{\text{start}})). \tag{10}$$

Here, cross-attention uses final-frame features as queries and initial-frame features as keys and values; P_Δ projects the feature difference to the attention output dimension, and E_Δ comprises two self-attention blocks that produce the visual-change tokens $\mathbf{F}_\Delta$.

An independent decoder conditions on $\mathbf{F}_\Delta$ through cross-attention to reconstruct the action segment. It uses the same masked flow-matching losses for position and rotation and binary cross-entropy for gripper state as the trajectory reconstruction branch:

$$\mathcal{L}_{\text{AG}} = \mathcal{L}_{\text{vis}}^{\text{pos}} + \lambda_{\text{rot}}\mathcal{L}_{\text{vis}}^{\text{rot}} + \lambda_{\text{grip}}\mathcal{L}_{\text{vis}}^{\text{grip}}. \tag{11}$$

The reconstruction weights are shared with $\mathcal{L}_{\text{AA}}$. Gradients from this objective update the visual encoder through both observations via LoRA (Hu et al., 2021), encouraging features that capture action-relevant visual changes.

Global-code-conditioned visual prediction. To associate the global action representation with its visible effects, we predict final-frame features from the initial observation and the global code $\tilde{\mathbf{z}}_g$, following the latent prediction principle of JEPA-based world models (Zhou et al., 2025a; Maes et al., 2026; Chen et al., 2026a). The global code modulates initial-frame features via FiLM before a shared MLP predicts each final-frame patch:

$$(\boldsymbol{\gamma}, \boldsymbol{\beta}) = P_{\mathrm{FiLM}}(\tilde{\mathbf{z}}_g), \qquad \boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^{d_v},$$
$$\hat{\mathbf{f}}_{\mathrm{end},p} = \mathrm{MLP}_{\mathrm{future}}((\mathbf{1} + \boldsymbol{\gamma}) \odot \mathrm{LN}(\mathbf{f}_{\mathrm{start},p}) + \boldsymbol{\beta}), \qquad p = 1, \ldots, P. \tag{12}$$

Here, P_{FiLM} is a learned affine map, $\mathbf{f}_{\mathrm{start},p}$ is the initial feature at patch p, and $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ are shared across patches. We optimize a patch-wise cosine loss:

$$\mathcal{L}_{\mathrm{future}} = \frac{1}{P} \sum_{p=1}^{P} \left[1 - \cos\Big(\hat{\mathbf{f}}_{\mathrm{end},p}, \mathrm{sg}(\mathbf{f}_{\mathrm{end},p}) \Big) \right], \tag{13}$$

where sg denotes stop-gradient. The initial-feature and global-code paths remain differentiable, propagating visual supervision to the action encoder and global codebook and encouraging global representations predictive of the observed state transition.

Training objective. VQAA combines trajectory reconstruction with the two visual objectives:

$$\mathcal{L}_{\mathrm{VQAA}} = \mathcal{L}_{\mathrm{AA}} + \lambda_{\mathrm{AG}} \mathcal{L}_{\mathrm{AG}} + \lambda_{\mathrm{future}} \mathcal{L}_{\mathrm{future}}, \tag{14}$$

We gradually increase $\lambda_{\mathrm{future}}$ to strengthen visual supervision and set $\lambda_{\mathrm{AG}} = 0$ once the visual encoder is frozen.

5 Experiments

5.1 Simulation Experiments

Experimental Setup We evaluate on RLBench (James et al., 2020) with 18 seen and eight policy-unseen tasks. The latter are held out from policy training, allowing us to evaluate generalization from pretrained atomic knowledge. Grouped by the atomic action they involve, these tasks span three categories: articulated-object manipulation (*close drawer, open jar*), object transport and placement (*pick up cup, phone on base, basketball in hoop, knife on board*), and switch operation (*lamp on, press switch*). The codebooks and adapter are pretrained on AtomAction and subsequently frozen, while the policies are trained on demonstrations from the 18 seen tasks. At test time, the planner schedules subtasks and the adapter retrieves action codes from current observations and subtask instructions, without any parameter updates. We report full-task success over 25 episodes per task in each of three random seeds. Appendix A.8 details the training configurations, evaluation protocol and compared methods.

Results Analysis Table 1 shows that strong seen-task performance of existing policies does not carry over to policy-unseen tasks: seven of the eight baselines reach only 11.83–19.17% on unseen tasks, and BridgeVLA (Li et al., 2025), the strongest on seen tasks (89.78%), trails PerAct (Shridhar et al., 2022), the weakest (37.11%), on unseen tasks (16.67% vs. 19.00%). LexiconVLA mitigates this limitation across five backbones spanning voxel-based, multi-view, and VLM-based architectures. On every backbone, it improves unseen-task and overall success while largely preserving seen-task performance, even though it executes each task as a sequence of atomic subtasks whose success depends jointly on every subtask and transition. The unseen-task gains are largest on the two strongest backbones, BridgeVLA++ (+7.66 points) (Li et al., 2026) and BridgeVLA (+17.50 points) (Li et al., 2025). This trend is consistent with the division of labor in LexiconVLA: the retrieved codes specify which atomic action to perform, whereas the base policy executes it, so a more capable policy converts this guidance into task success more effectively. With BridgeVLA (Li et al., 2025), unseen-task success more than doubles from 16.67% to 34.17%, exceeding the strongest baseline, 3D Diffuser Actor (28.67%) (Ke et al., 2024), by 5.50 points, with gains on six of the eight unseen tasks (Appendix A.13). Overall success rises accordingly from 67.28% to 71.08%, the highest among all evaluated methods. Together, these results support combining pretrained atomic knowledge with subtask-conditioned execution as an effective approach to policy-unseen task transfer. Moreover, per-task results are reported in Tables 15 and 16.

Table 1: **Multi-task performance on RLBench** (James et al., 2020)**.** Results are presented as mean±std success rate(%) over three random seeds, with 25 evaluation episodes per task and seed. Shaded rows apply LexiconVLA to the base policy above; green numbers give gains in percentage points. Best result per column in **bold**. * denotes results we obtained by repeatedly fine-tuning and evaluating from the officially released pre-trained checkpoint.

Method	Unseen (8 tasks)		Seen (18 tasks)	Overall (26 tasks)	
Act3D (Gervet et al., 2023)	$15.67_{\pm 2.52}$		$65.19_{\pm 1.22}$	$49.95_{\pm 1.43}$	
3D Diffuser Actor (Ke et al., 2024)	$28.67_{\pm 0.29}$		$78.30_{\pm 1.92}$	$63.03_{\pm 1.34}$	
SAM2Act (Fang et al., 2025)	$15.50_{\pm 1.50}$		$84.30_{\pm 1.80}$	$63.13_{\pm 1.02}$	
PerAct (Shridhar et al., 2022)	$19.00_{\pm 1.50}$		$37.11_{\pm 2.12}$	$31.54_{\pm 1.34}$	
LexiconVLA (PerAct)	$19.33_{\pm 0.58}$		$39.48_{\pm 2.11}$	$33.28_{\pm 1.62}$	
RVT (Goyal et al., 2023)	$11.83_{\pm 2.31}$		$59.33_{\pm 2.04}$	$44.72_{\pm 2.00}$	
LexiconVLA (RVT)	$14.50_{\pm 1.80}$		$60.81_{\pm 1.14}$	$46.56_{\pm 0.99}$	
RVT-2 (Goyal et al., 2024)	$18.00_{\pm 0.50}$		$75.63_{\pm 1.00}$	$57.90_{\pm 0.64}$	
LexiconVLA (RVT-2)	$19.83_{\pm 0.76}$		$76.30_{\pm 0.34}$	$58.92_{\pm 0.31}$	
BridgeVLA++* (Li et al., 2026)	$19.17_{\pm 2.02}$		$79.70_{\pm 1.12}$	$61.08_{\pm 1.20}$	
LexiconVLA (BridgeVLA++)	$26.83_{\pm 1.15}$		$77.04_{\pm 1.68}$	$61.59_{\pm 0.99}$	
BridgeVLA (Li et al., 2025)	$16.67_{\pm 1.26}$		$\mathbf{89.78}_{\pm 1.02}$	$67.28_{\pm 0.94}$	
LexiconVLA (BridgeVLA)	$\mathbf{34.17}_{\pm 2.02}$	↑17.50	$87.48_{\pm 0.93}$	$\mathbf{71.08}_{\pm 1.16}$	↑3.80

5.2 Ablation and Analysis

Table 2 ablates the components of LexiconVLA with BridgeVLA as the base policy. Every ablation lowers unseen-task and overall success, whereas seen-task success changes little, indicating that these components mainly serve transfer to policy-unseen tasks. Without the planner, codes are retrieved once from the full task instruction, and unseen-task success drops from 34.17% to 28.50%, indicating that retrieving codes for each atomic subtask matters. Its slight seen-task gain reflects single-pass execution, which avoids the chained subtask transitions discussed above. Without the codebooks, the policy follows the planner's subtask instructions without code conditioning, and unseen-task success drops to 26.00%; subtask decomposition alone therefore does not account for the gain. Removing the global codebook, the detail codebook, or V3A from codebook learning lowers unseen-task success to 26.50%, 29.50%, and 26.50%, respectively, showing that both codebooks and the visual grounding provided by V3A contribute to transfer. Additional analyses in Appendix A.3 show that the default codebook size yields the highest unseen-task success, that the learned codes are fully used and retain atomic-action information on held-out tasks, and that stronger planners further raise unseen-task success without retraining any component.

Table 2: Success rates (%) in the ablation study.

Method	Unseen (8)	Seen (18)	Overall (26)
LexiconVLA (BridgeVLA)	**34.17**	87.48	**71.08**
w/o Planner	28.50	**88.22**	69.85
w/o Codebooks	26.00	87.78	68.77
w/o Global Codebook	26.50	85.33	67.23
w/o Detail Codebook	29.50	87.11	69.38
w/o V3A	26.50	84.89	66.92

5.3 Real-World Experiments

Table 3: Real-world success rates (%) on sim-to-real and generalization tasks.

Methods	Standard					Generalization				
	Sim-Short		Sim-Long	Real only-Short	Real only-Long	Distractor		Height	Category	Instruction
	Open drawer	Put item in drawer	Push buttons	Pour water into the bowl	Place fruits on the plate	Push buttons	Place fruits on the plate	Pour water into the bowl	Put item in drawer	Open drawer
BridgeVLA (Li et al., 2025)	50.0	0.0	60.0	20.0	0.0	40.0	0.0	0.0	0.0	40.0
$\pi_{0.5}$ (Intelligence et al., 2025)	60.0	0.0	**80.0**	0.0	0.0	40.0	0.0	0.0	0.0	20.0
LexiconVLA (Ours)	**100.0**	**40.0**	**80.0**	**40.0**	**50.0**	**60.0**	**20.0**	**80.0**	**40.0**	**80.0**

Experimental Setup We compare LexiconVLA (BridgeVLA) with its base policy BridgeVLA (Li et al., 2025) and the generalist VLA $\pi_{0.5}$ (Intelligence et al., 2025), all fine-tuned on the same 30 real-robot demonstrations per task. The *Standard* setting evaluates five tasks under training conditions: three have RLBench counterparts (*Sim*), and two have none (*Real-only*). Following prior real-robot evaluations (Li et al., 2025; Sun et al., 2026), the *Generalization* setting introduces distractors, elevated objects, unseen object categories, and paraphrased instructions, all absent from

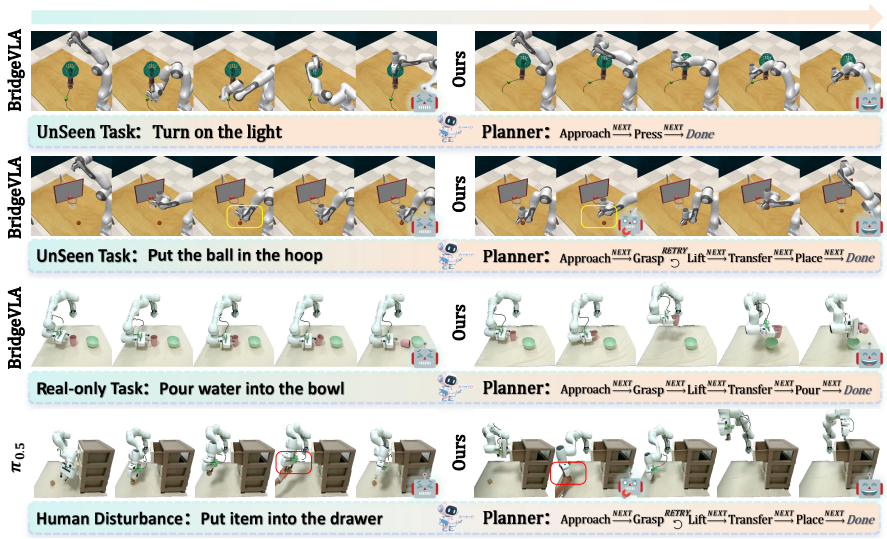


Figure 5: **Qualitative rollouts in simulation and the real world.** *Ours*: LexiconVLA. Rows 1–2: policy-unseen RLBench tasks; rows 3–4: real-robot tasks. After a missed grasp (yellow box) or human disturbance (red box), our planner triggers RETRY, re-executing the failed subtask with codes re-retrieved from the current observation, whereas the baselines fail to recover.

the demonstrations, without further training. Each task is evaluated over 10 trials per setting, and a trial succeeds only if the entire instruction is completed. Appendix A.9 details the hardware, tasks, perturbations, and evaluation protocol.

Results Analysis Table 3 shows that LexiconVLA achieves the the best or tied-best result under all ten conditions. Averaged over the five tasks, it reaches 62.0% in the Standard setting versus 26.0% for BridgeVLA (Li et al., 2025) and 28.0% for $\pi_{0.5}$ (Intelligence et al., 2025). The gain is most pronounced on multi-step tasks: both baselines fail every trial of put item in drawer and place fruits on the plate, whereas LexiconVLA reaches 40.0% and 50.0%. Although the baselines open drawers in 50.0–60.0% of open-drawer trials, they never complete drawer placement, mirroring the context-dependent atomic execution revealed by our diagnostic probe (Appendix A.1). Under perturbations absent from the demonstrations, LexiconVLA maintains 56.0%, whereas BridgeVLA (Li et al., 2025) and $\pi_{0.5}$ (Intelligence et al., 2025) fall to 16.0% and 12.0%. Together with its lead on both real-only tasks, these results show that the atomic-action lexicon, learned solely from simulated segments, remains effective on a physical robot, both on tasks absent from simulation and under unseen conditions. In Figure 5, the planner decomposes each instruction into atomic subtasks, such as approach, grasp, and transfer, which the policy executes with codes retrieved for each subtask, completing policy-unseen RLBench tasks and real-only pouring where BridgeVLA (Li et al., 2025) fails. Step-wise execution also localizes failures. After a failed grasp in put the ball in the hoop, or a human disturbance in put item into the drawer, the planner detects the failure from visual feedback, issues RETRY, and resumes from the failed action with codes re-retrieved from the updated observation. $\pi_{0.5}$ (Intelligence et al., 2025) has no explicit notion of which step has failed, and its rollout fails under the same disturbance. Appendix A.10 details per-condition results and failure modes.

6 Conclusion

We presented LexiconVLA, which makes recurring manipulation interactions explicit and reusable across tasks. Motivated by our finding that task completion does not imply reliable atomic-action execution, we constructed the *AtomAction* Dataset and learned global and detail codebooks grounded in motion and its visual effects through Visual-Atomic Action Alignment. A planner and a scene-aware adapter retrieve codes for each atomic subtask to condition a shared policy, without skill-specific experts or deployment-time parameter updates. Across five backbones on RLBench, LexiconVLA improves unseen-task success while largely preserving seen-task performance. On a real robot, the simulation-learned lexicon remains effective on tasks absent from simulation and under unseen conditions, and stepwise execution supports recovery from failures. Extending the lexicon with real-world and cross-embodiment data is a natural next step.

AI USE STATEMENT

We used generative AI to assist author-led work, not to conduct autonomous research, with a commitment to academic integrity and research ethics. Its uses included identifying relevant literature, drafting and polishing portions of the manuscript, checking clarity and consistency, and organizing existing data and experimental results into tables. For dataset construction, GPT-4o generated eight phase-level language descriptions for each annotated atomic-action segment, grounded in the action definition, task description, and local phase context (Appendices A.6 and A.12). These descriptions constitute the AI-generated language component of AtomAction. AI models also serve as components of the reported method and evaluation, including the VLM planner and the GPT-5.4 diagnostic judge, as described in Section 4.2 and Appendix A.1.

We additionally used AI-assisted image editing to remove black curtain backgrounds from robot-manipulation photographs for clearer presentation of the robot and manipulated objects. We manually compared the edited images with the originals to verify that the depicted objects, robot actions, and task outcomes were preserved.

All AI-assisted material underwent detailed human review before inclusion. The authors checked literature references and factual claims against original sources, checked tabulated values against the underlying data and experimental records, and reviewed generated language descriptions for consistency with their associated action segments and annotations. We also reviewed manuscript text for accuracy, attribution, and fidelity to our intended meaning, correcting or excluding unsupported, misleading, or fabricated content. AI output was not treated as evidence without verification. The authors retained responsibility for scientific judgments and conclusions and take full responsibility for the accuracy, originality, and integrity of the final manuscript, dataset, figures, and reported results.

REFERENCES

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2026. URL https://arxiv.org/abs/2410.24164.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023. URL https://arxiv.org/abs/2307.15818.

Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bechtle, Feryal Behbahani, Stephanie Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative interactive environments, 2024. URL https://arxiv.org/abs/2402.15391.

Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. Univla: Learning to act anywhere with task-centric latent actions, 2025. URL https://arxiv.org/abs/2505.06111.

Chilam Cheang, Sijin Chen, Zhongren Cui, Yingdong Hu, Liqun Huang, Tao Kong, Hang Li, Yifeng Li, Yuxiao Liu, Xiao Ma, Hao Niu, Wenxuan Ou, Wanli Peng, Zeyu Ren, Haixin Shi, Jiawen Tian, Hongtao Wu, Xin Xiao, Yuyang Xiao, Jiafeng Xu, and Yichu Yang. Gr-3 technical report, 2025. URL https://arxiv.org/abs/2507.15493.

Delong Chen, Mustafa Shukor, Theo Moutakanni, Willy Chung, Jade Yu, Tejaswi Kasarla, Yejin Bang, Allen Bolourchi, Yann LeCun, and Pascale Fung. Vl-jepa: Joint embedding predictive architecture for vision-language, 2026a. URL https://arxiv.org/abs/2512.10942.

Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Haoran Lu, Weijie Wan, Baijun Chen, Songling Liu, Haowen Yan, Honghao Su, Zhiyang Dou, Kaixuan Wang, Dandan Zhang, Yunze Liu, Yan Qin, Qiwei Liang, Qiwei Wu, Zijian Lin, Wenwei Lin, Yuran Wang, Minghua He, Tianshu Wu, Ruihai Wu, Jingquan Zhou, Kai-Chong Lei, Haibao Yu, Yuanfeng Ji, Weiyang Jin, Guanyu Lin, Xiaofan Li, Qi Xiong, Renjing Xu, Zhongyu Li, Wenhao Chai, Enze Xie, Ziwei Wang, Yao Mu, Hao Dong, Wojciech Matusik, Mingyu Ding, Wenbo Ding, Ping Luo, and Masayoshi Tomizuka. Robodojo: A unified sim-and-real benchmark for comprehensive evaluation of generalist robot manipulation policies, 2026b. URL https://arxiv.org/abs/2607.04434.

Yang Chen, Xiaoguang Ma, and Bin Zhao. Mean-flow based one-step vision-language-action, 2026c. URL https://arxiv.org/abs/2603.01469.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion, 2024. URL https://arxiv.org/abs/2303.04137.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

Haoquan Fang, Markus Grotz, Wilbert Pumacay, Yi Ru Wang, Dieter Fox, Ranjay Krishna, and Jiafei Duan. Sam2act: Integrating visual foundation model with a memory architecture for robotic manipulation, 2025. URL https://arxiv.org/abs/2501.18564.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, Jinlan Fu, Jingjing Gong, and Xipeng Qiu. Libero-plus: In-depth robustness analysis of vision-language-action models, 2025. URL https://arxiv.org/abs/2510.13626.

Theophile Gervet, Zhou Xian, Nikolaos Gkanatsios, and Katerina Fragkiadaki. Act3d: 3d feature field transformers for multi-task robotic manipulation, 2023. URL https://arxiv.org/abs/2306.17817.

Ankit Goyal, Jie Xu, Yijie Guo, Valts Blukis, Yu-Wei Chao, and Dieter Fox. Rvt: Robotic view transformer for 3d object manipulation, 2023. URL https://arxiv.org/abs/2306.14896.

Ankit Goyal, Valts Blukis, Jie Xu, Yijie Guo, Yu-Wei Chao, and Dieter Fox. Rvt-2: Learning precise manipulation from few demonstrations, 2024. URL https://arxiv.org/abs/2406.08545.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. URL https://arxiv.org/abs/1606.08415.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021. URL https://arxiv.org/abs/2106.09685.

Yucheng Hu, Yanjiang Guo, Pengchao Wang, Xiaoyu Chen, Yen-Jen Wang, Jianke Zhang, Koushil Sreenath, Chaochao Lu, and Jianyu Chen. Video prediction policy: A generalist robot policy with predictive visual representations, 2025. URL https://arxiv.org/abs/2412.14803.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner,

Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020. doi: 10.1109/LRA.2020.2974707.

Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 3d diffuser actor: Policy diffusion with 3d scene representations, 2024. URL https://arxiv.org/abs/2402.10885.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model, 2024. URL https://arxiv.org/abs/2406.09246.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025. URL https://arxiv.org/abs/2502.19645.

Seungjae Lee, Yibin Wang, Haritheja Etukuru, H. Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions, 2024. URL https://arxiv.org/abs/2403.03181.

Peiyan Li, Yixiang Chen, Hongtao Wu, Xiao Ma, Xiangnan Wu, Yan Huang, Liang Wang, Tao Kong, and Tieniu Tan. BridgeVLA: Input-output alignment for efficient 3d manipulation learning with vision-language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=ffBF6hYuQv.

Peiyan Li, Yuze Zhu, Yixiang Chen, Qisen Ma, Yuan Xu, Jiabing Yang, He Guan, Yan Huang, Hongtao Wu, Xiao Ma, Tao Kong, Liang Wang, and Tieniu Tan. Bridgevla++: A data-efficient, generalizable, and memory-augmented vision-language-action framework for 3d manipulation, 2026. URL https://arxiv.org/abs/2608.05042.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2023. URL https://arxiv.org/abs/2210.02747.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791. Curran Associates, Inc., 2023. doi: 10.52202/075280-1939. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/8c3c666820ea055a77726d66fc7d447f-Paper-Datasets_and_Benchmarks.pdf.

Lucas Maes, Quentin Le Lidec, Damien Scieur, Yann LeCun, and Randall Balestriero. Leworldmodel: Stable end-to-end joint-embedding predictive architecture from pixels, 2026. URL https://arxiv.org/abs/2603.19312.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2024. URL https://arxiv.org/abs/2304.07193.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models, 2025. URL https://arxiv.org/abs/2501.09747.

Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, and Xuelong Li. Spatialvla: Exploring spatial representations for visual-language-action model, 2025. URL https://arxiv.org/abs/2501.15830.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation, 2022. URL https://arxiv.org/abs/2209.05451.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021. URL https://arxiv.org/abs/2104.09864.

Xiaoquan Sun, Zetian Xu, Chen Cao, Zonghe Liu, Yihan Sun, Jingrui Pang, Ruijian Zhang, Zhen Yang, Kang Pang, Dingxin He, Mingqi Yuan, and Jiayu Chen. Atomvla: Scalable post-training for robotic manipulation via predictive latent world models, 2026. URL https://arxiv.org/abs/2603.08519.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy, 2024. URL https://arxiv.org/abs/2405.12213.

Mohammad Hassan Vali and Tom Bäckström. Nsvq: Noise substitution in vector quantization for machine learning. *IEEE Access*, 10:13598–13610, 2022. doi: 10.1109/ACCESS.2022.3147670.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Yi Wang, Xinhao Li, Ziang Yan, Yinan He, Jiashuo Yu, Xiangyu Zeng, Chenting Wang, Changlian Ma, Haian Huang, Jianfei Gao, Min Dou, Kai Chen, Wenhai Wang, Yu Qiao, Yali Wang, and Limin Wang. Internvideo2.5: Empowering video mllms with long and rich context modeling, 2025. URL https://arxiv.org/abs/2501.12386.

Seonghyeon Ye, Joel Jang, Byeongguk Jeon, Sejune Joo, Jianwei Yang, Baolin Peng, Ajay Mandlekar, Reuben Tan, Yu-Wei Chao, Bill Yuchen Lin, Lars Liden, Kimin Lee, Jianfeng Gao, Luke Zettlemoyer, Dieter Fox, and Minjoon Seo. Latent action pretraining from videos, 2025. URL https://arxiv.org/abs/2410.11758.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi "Jim" Fan, and Joel Jang. World action models are zero-shot policies, 2026. URL https://arxiv.org/abs/2602.15922.

Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, Yu Su, Wenhu Chen, and Graham Neubig. Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark, 2025. URL https://arxiv.org/abs/2409.02813.

Michał Zawalski, William Chen, Karl Pertsch, Oier Mees, Chelsea Finn, and Sergey Levine. Robotic control via embodied chain-of-thought reasoning, 2025. URL https://arxiv.org/abs/2407.08693.

Jiahui Zhang, Yurui Chen, Yueming Xu, Ze Huang, Yanpeng Zhou, Yu-Jie Yuan, Xinyue Cai, Guowei Huang, Xingyue Quan, Hang Xu, and Li Zhang. 4d-vla: Spatiotemporal vision-language-action pretraining with cross-scene calibration, 2025. URL https://arxiv.org/abs/2506.22242.

Jianke Zhang, Xiaoyu Chen, Qiuyue Wang, Mingsheng Li, Yanjiang Guo, Yucheng Hu, Jiajun Zhang, Shuai Bai, Junyang Lin, and Jianyu Chen. Vlm4vla: Revisiting vision-language-models in vision-language-action models, 2026a. URL https://arxiv.org/abs/2601.03309.

Likui Zhang, Tao Tang, Zhihao Zhan, Xiuwei Chen, Zisheng Chen, Jianhua Han, Jiangtong Zhu, Pei Xu, Hang Xu, Hefeng Wu, Liang Lin, and Xiaodan Liang. Atomicvla: Unlocking the potential of atomic skill learning in robots. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20743–20754, June 2026b.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, Ankur Handa, Ming-Yu Liu, Donglai Xiang, Gordon Wetzstein, and Tsung-Yi Lin. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models, 2025. URL https://arxiv.org/abs/2503.22020.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *Forty-second International Conference on Machine Learning*, 2025a. URL https://openreview.net/forum?id=D5RNACOZEI.

Hongyi Zhou, Weiran Liao, Xi Huang, Yucheng Tang, Fabian Otto, Xiaogang Jia, Xinkai Jiang, Simon Hilber, Ge Li, Qian Wang, Ömer Erdinç Yağmurlu, Nils Blank, Moritz Reuss, and Rudolf Lioutikov. Beast: Efficient tokenization of b-splines encoded action sequences for imitation learning, 2025b. URL https://arxiv.org/abs/2506.06072.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. Libero-pro: Towards robust and fair evaluation of vision-language-action models beyond memorization, 2026. URL https://arxiv.org/abs/2510.03827.

A Appendix

A.1 Diagnostic Probe of Atomic-Action Invocation

Protocol. Diagnostic probes five pretrained policies. We fix ten validation demonstrations per task for reach-and-drag, open-drawer, place-wine, sweep-to-dustpan, close-jar, push-buttons, and slide-block: 70 episodes, 290 segments, and 13 atomic labels. Selection requires complete keyframe coverage. Full-task evaluation uses RLBench success detection and 25 policy calls. For each isolated atom, we reset the demonstration, replay its ground-truth (GT) prefix, verify the handoff (position error $< 2\,\text{cm}$ and orientation error $< 5^\circ$, matching gripper state). Invalid handoffs are excluded; segments are evaluated independently. GPT-5.4 judges front, wrist, and left-shoulder views with robot-state information, without policy identities. Strict scoring requires completed execution relative to the GT endpoint. Detailed results are provided in Table.4.

Table 4: Per-task results corresponding to Figure 1. Each cell reports Full task SR / Atomic SR, in %. Atomic SR pools valid segments within the task; full-task SR uses RLBench success detection.

Task	PerAct	BridgeVLA++	Act3D	RVT-2	3DDA	LexiconVLA
Reach & drag	100.0 / 18.0	80.0 / 26.0	90.0 / 22.0	90.0 / 36.0	100.0 / 22.0	100.0 / 58.0
Place wine	50.0 / 29.7	90.0 / 33.3	80.0 / 26.3	80.0 / 45.0	80.0 / 25.6	70.0 / 69.2
Sweep to dustpan	70.0 / 0.0	100.0 / 40.0	100.0 / 17.5	100.0 / 17.5	80.0 / 25.0	100.0 / 35.0
Slide block	100.0 / 16.0	100.0 / 31.4	80.0 / 26.0	70.0 / 14.0	90.0 / 22.0	100.0 / 66.0
Close jar	40.0 / 14.0	100.0 / 25.0	90.0 / 24.0	20.0 / 20.0	60.0 / 0.0	100.0 / 67.5
Push buttons	20.0 / 40.0	100.0 / 56.7	100.0 / 50.0	100.0 / 40.0	100.0 / 66.7	100.0 / 83.3
Open drawer	80.0 / 81.5	100.0 / 60.0	90.0 / 65.5	90.0 / 76.7	100.0 / 64.3	100.0 / 83.3

A.2 Adapter Architecture and Training

Architecture. Frozen DINOv2 (Oquab et al., 2024), ViT-B/14 (Dosovitskiy et al., 2021) and CLIP-ViT-B/16 (Radford et al., 2021) encoders tokenize the initial front/wrist images and subtask instruction. Pooling yields 64 visual tokens per camera, with learned spatial and camera embeddings. Together with 77 text positions, one global query, and $N_d = 9$ detail queries, they form a 215-token sequence. A four-layer, 512-dimensional Transformer (Vaswani et al., 2023) with eight heads processes the sequence. Observation tokens attend only to observations; the global query

also attends to itself; detail queries attend to all valid tokens. The final query features produce code logits:

$$\mathbf{o}_g = f_g(\mathbf{h}_g^{\text{ada}}) \in \mathbb{R}^{K_g},$$
$$\mathbf{o}_{d,n} = f_d\big([\mathbf{h}_{d,n}^{\text{ada}}; \mathbf{h}_g^{\text{ada}}]\big) \in \mathbb{R}^{K_d}, \qquad n = 1, \ldots, N_d. \tag{15}$$

Here, $K_g = 36$ and $K_d = 192$ are the codebook sizes. Both heads are two-layer GELU (Hendrycks & Gimpel, 2023) MLPs, with f_d shared across detail slots. Argmax yields the predicted indices.

Training. Frozen Atomic Action-NSVQ supplies nearest-neighbor code indices from demonstration trajectories as fixed labels. The adapter learns to predict them from initial observations and instructions:

$$\mathcal{L}_{\text{adapter}} = \mathbb{E}_{\mathcal{D}_{\text{ada}}}\left[\text{CE}_\eta(\mathbf{o}_g, k_g^\star) + \frac{\lambda_d}{N_d}\sum_{n=1}^{N_d}\text{CE}_\eta(\mathbf{o}_{d,n}, k_{d,n}^\star)\right], \qquad \lambda_d = 1, \quad \eta = 0.05, \tag{16}$$

where $\mathcal{D}_{\text{ada}}$ denotes training data, starred indices are labels, and CE_η uses label smoothing η. The adapter is trained for 100 epochs with AdamW, batch size 64, and learning rate 3×10^{-4}, using three warmup epochs followed by cosine decay. The image/text encoders and codebooks remain frozen.

Deployment. The frozen adapter retrieves code vectors by predicted indices. CONTINUE reuses the codes, while subtask transitions, retries, and replanning refresh them from current observations.

A.3 Additional Analyses

Codebook size. We scale the global and detail codebooks jointly from the default 36/192 entries to $\times\frac{1}{4}$ (9/48), $\times\frac{1}{2}$ (18/96), and $\times 2$ (72/384), retrain the codebooks with the default recipe, retrain the adapter and policy on each, and evaluate with BridgeVLA (Figure 6). Seen-task success varies by less than 3 points across sizes, whereas unseen-task success peaks at the default size (34.17%). Doubling the codebooks lowers it to 24.50%, a drop of 9.67 points that far exceeds the run-to-run standard deviation of the default configuration (2.02 points), and both smaller codebooks also fall below the default (32.00% and 28.50%). All entries remain in use at every size, so these differences reflect how the codebooks partition atomic actions rather than codebook collapse. The trend is consistent with the role of the codebooks as a shared lexicon: a compact codebook lets different instances of the same interaction share entries, an overly large one spreads them across more entries that are shared less across tasks, and an overly small one, with fewer global entries than annotated atomic-action types at $\times\frac{1}{4}$, must merge distinct actions.

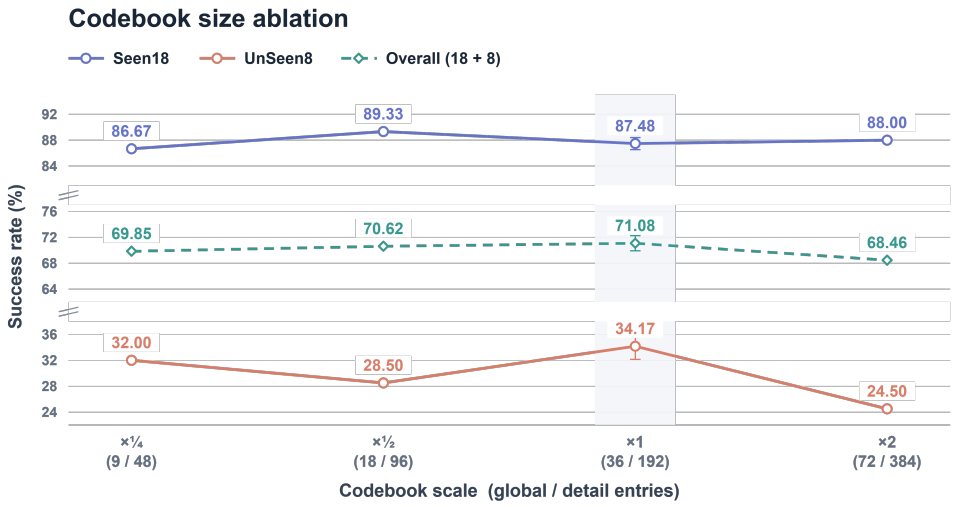


Figure 6: **Codebook size ablation.** Success rates (%) of LexiconVLA (BridgeVLA) with the global and detail codebooks scaled jointly (global/detail entries in parentheses). The default size ($\times 1$, shaded) reports mean$\pm$std over three runs; other sizes are evaluated once.

Choice of planner. We replace the model behind the planner with five alternatives while keeping the codebooks, adapter, policy (BridgeVLA), prompts, and evaluation episodes fixed (Figure 7). Since the planner decomposes the task and monitors execution from visual observations, we relate success to each model's multimodal reasoning ability, measured by MMMU-Pro (Yue et al., 2025). Unseen-task success rises with MMMU-Pro from 27.50% with gpt-5.6-luna to 35.50% with gemini-3.8-flash, and overall success follows the same trend (68.46% to 72.46%), whereas seen-task success stays between 86.67% and 88.89%, consistent with seen tasks being largely handled by the base policy. Even the weakest planner improves unseen-task success over BridgeVLA by 10.83 points (16.67% to 27.50%), so the gains of LexiconVLA do not hinge on a particular planner, and stronger planners raise them further without retraining any component.

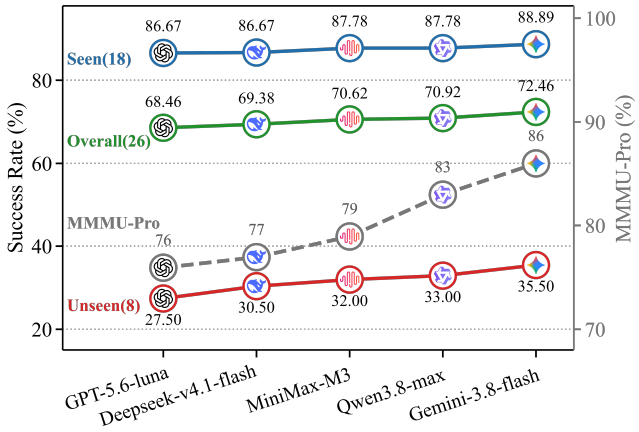


Figure 7: **Effect of the planner model.** Success rates (%) of LexiconVLA (BridgeVLA) with different planner models, ordered by MMMU-Pro score (dashed line, right axis). All other components are fixed; each planner is evaluated once.

Learned codebook structure. We embed encoder features and all codewords of the default codebooks with t-SNE, coloring them by clusters of codeword vectors (Figure 8). All 36 global and 192 detail codewords are active, with normalized perplexities of 0.657 and 0.870, so neither codebook collapses onto a few entries. Global features form compact, well-separated groups around distinct codewords, indicating that the global codebook abstracts segments into a finite set of recurring patterns, while detail codewords densely cover a continuous region and capture the finer execution variation that complements the global code. To test whether the codes carry atomic-action information that generalizes across tasks, we fit a linear probe on the frozen codes with five-fold cross-validation grouped by task: on tasks held out from probe training, global and detail codes predict the annotated atomic action with 41.38% and 31.08% accuracy, respectively, compared with 23.58% for the majority class. The codes thus retain atomic-action information that transfers across tasks, the property LexiconVLA relies on when retrieving codes for unseen tasks.

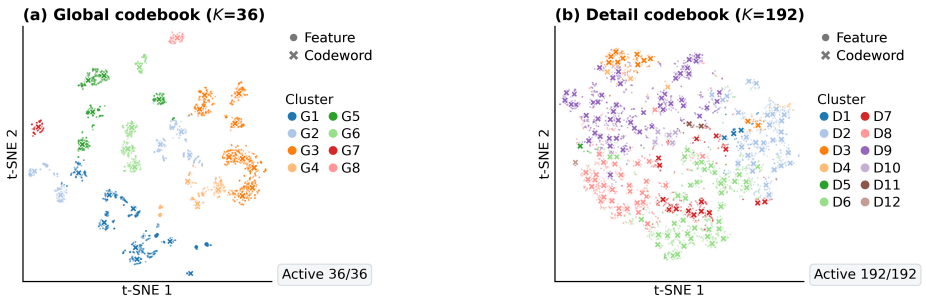


Figure 8: **t-SNE of the learned (a) global and (b) detail codebooks.** Dots denote encoder features and crosses denote codewords; cross area scales with usage frequency.

A.4 LexiconVLA Scheduling

A.4.1 Closed-loop planning and execution

Execution and monitoring. LexiconVLA coordinates task planning, lexicon-conditioned execution, and visual feedback through the closed-loop procedure in Algorithm 1, drawing on feedback-driven orchestration frameworks such as OpenClaw and Hermes Agent. PLAN receives current images, the task instruction, and available decomposition and instruction priors, and produces atomic subtask instructions. A local state machine maintains the plan, active index, completion records, execution counters, and cached lexicon entries. The adapter retrieves codes for the active subtask, and the VLA executes end-effector commands using the latest observations, task/subtask instructions, and code context. After each keyframe action, LexiconVLA updates the execution state and checks whether monitoring is required. MONITOR receives the current images, plan, active subtask, elapsed steps, robot-state information, and trigger reason; it returns a scheduling decision, an optional target index, and a rationale. This permits selective VLM assessment while the policy continues to respond to observations.

Algorithm 1 LexiconVLA: online planning and execution

Input: Task instruction ℓ, frozen codebooks, adapter, and VLA policy π_θ.
Observe $(O, \mathbf{s})$; $P \leftarrow \mathrm{PLAN}(O, \ell)$; initialize active index $i \leftarrow 1$.
Initialize execution records and counters; initialize the retrieval key and code history.
while the episode is active **do**
  if the active subtask key (i, ℓ_i) has changed and lexicon use is enabled **then**
    Predict codebook indices from (O, ℓ_i) using the adapter.
    Retrieve and cache $(\mathbf{z}_g^{(i)}, \mathbf{Z}_d^{(i)})$ from the frozen lexicon; update the retrieval key and code history.
  end if
  Execute the policy with current observations, task/subtask instructions, and cached code context; disable injection if lexicon use is off.
  Observe updated $(O, \mathbf{s})$ and execution feedback; update counters.
  Select the first eligible event or heartbeat trigger, if any.
  if a trigger is present and monitoring is enabled within budget **then**
    $(a, j, r) \leftarrow \mathrm{MONITOR}(O, \ell, P, i, \text{execution state}, \text{trigger})$.
    Apply skill a through the local state machine:
    NEXT: record completion and advance; extend the plan at its end.
    CONTINUE: retain the active subtask and cached codes.
    RETRY: retry step j (default i); revoke rolled-back completions.
    REPLAN: $P \leftarrow \mathrm{PLAN}(O, \ell)$; reset i and completion records.
    Apply recovery budgets and terminal plan-extension rules.
  end if
end while

ℓ_i is the instruction for the active atomic-action subtask in P; j and r are the optional target index and decision reason. The code context includes retrieved entries and causal global-code history. Trigger eligibility, code reuse, and budget-dependent overrides are specified below.

A.4.2 Monitoring and recovery rules

Monitor triggers. Local checks run at each keyframe action step, rather than each simulation frame. In priority order, monitoring is triggered by:
Only the first eligible trigger invokes MONITOR, with the heartbeat providing periodic assessment when no higher-priority event applies. At the final planned subtask, only prolonged execution and inactivity triggers remain active. A trigger requests assessment; it does not itself advance the plan.

Scheduling and recovery. The state machine applies the four core skills as follows:
NEXT requires visible completion. A reachable subtask with incomplete motion calls for CONTINUE; environmental changes that invalidate the plan call for REPLAN. RETRY can return to an earlier failed subtask and revoke subsequent completion records. These decisions use observed outcomes to guide progression and recovery.

Lexicon entry reuse. The current BridgeVLA integration retrieves codes when the active subtask key (i, ℓ_i) changes and lexicon use is enabled; otherwise, it reuses the cached entries. Disabling lexicon use clears the current codes, and episode reset clears the code history. A RETRY or REPLAN that retains the same index and instruction does not necessarily trigger fresh retrieval. Likewise,

Table 5: Trigger conditions for invoking the MONITOR function

Trigger	Condition
Gripper change	Binary gripper state changes, except for actions where closure serves as preparation or tool use.
Prolonged execution	The active subtask exceeds four keyframe steps.
Persistent inactivity	Inactivity persists for two steps, using joint-speed and end-effector displacement thresholds of 0.01 rad/s and 0.005 m; detected joint motion overrides the displacement test.
Heartbeat	Three keyframe steps have elapsed since the last monitoring call.

Table 6: State-machine updates for each scheduling skill

Skill	State update
NEXT	Mark the active subtask complete, advance by one step, and reset its execution and retry counters
CONTINUE	Retain the active subtask and cached codes
RETRY	Repeat the indicated failed step; when rolling back, revoke completion records from that step onward
REPLAN	Clear completion records and regenerate the plan from the current observation

re-enabling lexicon use without changing this key does not independently trigger retrieval. These are implementation limits of the current cache, rather than guarantees of scene-dependent retrieval after every recovery decision.

Plan exhaustion and limits. The implementation additionally supports EXTEND: when the final planned step is exhausted, a final-step NEXT is converted into a request for remaining subtasks. EXTEND preserves progress and appends the new steps. Extensions are limited to two, with at most ten planned segments; an empty or blocked extension disables further monitoring. Each episode permits at most 25 VLM calls and four replans. Further replan requests become CONTINUE. After more than four retries, the controller switches the decomposition prior where available and advances beyond the repeatedly failed step. These limits bound planner usage and recovery attempts.

A.5 Action Preprocessing and Normalization

The encoder input contains 39 scalar components: 3 for end-effector position, 6 for rotation, 21 for joint positions, velocities, and torques, 8 for gripper joint positions and contact forces, and 1 for the binary gripper state. These components describe complementary aspects of the recorded robot state. In particular, the six rotation components encode a three-degree-of-freedom orientation; the input dimension does not denote the robot's number of degrees of freedom.

Table 7: Preprocessing of the 39-dimensional encoder input.

Input field	**Dim.**	**Preprocessing**
End-effector position $\mathbf{p}_t$	3	Quantile normalization
End-effector rotation $\mathbf{r}_t$	6	Quaternion-to-6D conversion; no statistical normalization
Joint positions $\mathbf{q}_t$	7	Z-score normalization
Joint velocities $\dot{\mathbf{q}}_t$	7	Quantile normalization
Joint torques $\mathbf{f}_t^{\text{joint}}$	7	Quantile normalization
Gripper joint positions $\mathbf{q}_t^{\text{grip}}$	2	Z-score normalization
Gripper contact forces $\mathbf{f}_t^{\text{grip}}$	6	Quantile normalization
Gripper state g_t	1	Binary embedding; no statistical normalization

Field-specific normalization. We normalize each scalar dimension independently using dataset-level statistics shared across all action categories. The current preprocessing computes these statistics from all indexable trajectories under the dataset root, before sequence padding. The transformations are summarized in Table 7.

1. For a scalar measurement s, quantile normalization uses its first and ninety-ninth percentiles, denoted by $Q_{0.01}$ and $Q_{0.99}$:

$$\tilde{s} = 2\frac{s - Q_{0.01}}{\max(Q_{0.99} - Q_{0.01}, \epsilon_{\text{norm}})} - 1, \qquad \epsilon_{\text{norm}} = 10^{-6}. \tag{17}$$

   This scaling limits the influence of extreme observations on the normalization range. The percentile endpoints map to -1 and 1 whenever their separation is at least ϵ_{norm}. We do not clip the normalized values, so observations outside the percentile interval retain their relative magnitudes and may fall outside $[-1, 1]$.

2. Joint and gripper joint positions are standardized using their per-dimension mean μ and population standard deviation σ:

$$\tilde{s} = \frac{s - \mu}{\max(\sigma, \epsilon_{\text{norm}})}. \tag{18}$$

Both transformations impose a lower bound on the denominator for numerical stability. The normalization statistics are fixed during model training, and the same position transformation is used for encoder inputs and trajectory reconstruction targets.

Rotation and gripper representations. The recorded end-effector quaternion is converted to a rotation matrix $\mathbf{R}_t \in \mathrm{SO}(3)$. We concatenate its first two columns to obtain

$$\mathbf{r}_t = \left[(\mathbf{R}_t)_{:,1}^{\top},\ (\mathbf{R}_t)_{:,2}^{\top}\right]^{\top} \in \mathbb{R}^6. \tag{19}$$

These components lie in $[-1, 1]$ and receive no further statistical normalization. The binary gripper state g_t is mapped to a learnable 64-dimensional embedding before its feature projection. Padding is introduced after preprocessing, and the resulting validity mask excludes padded timesteps from encoding, aggregation, and reconstruction losses.

A.6 Dataset Construction and Annotation Pipeline

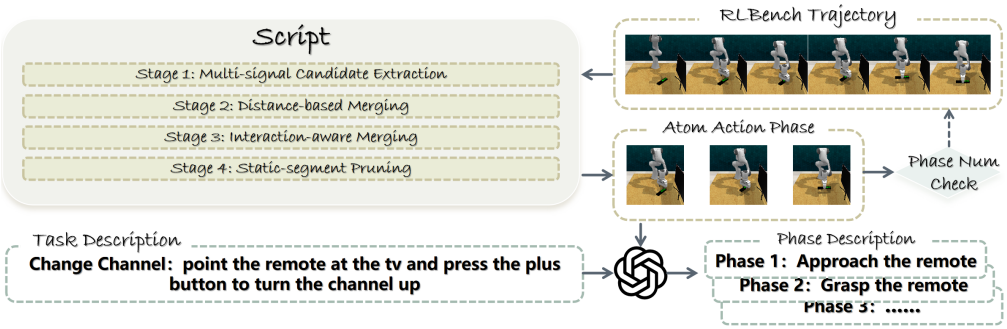


Figure 9: AtomAction Dataset construction pipeline. Complete RLBench (James et al., 2020) expert trajectories are segmented by a rule-based procedure, checked against expert-provided phase-count priors, and annotated with atomic-action labels and phase-level language descriptions.

Figure 9 presents the construction pipeline of the AtomAction Dataset. Starting from complete RLBench (James et al., 2020) expert trajectories, the pipeline first applies a rule-based script to divide each continuous demonstration into temporally localized manipulation phases. The script performs four successive operations: multi-signal candidate extraction, distance-based merging, interaction-aware merging, and static-segment pruning.

Stage 1: Multi-signal candidate extraction. The script identifies candidate transition points from complementary trajectory signals, including gripper-state changes, velocity stop/start events, and contact-related changes. The union of these events forms an inclusive set of candidate phase boundaries.

Stage 2: Distance-based merging. The initial candidate set may contain several boundaries around the same physical transition. To reduce such over-segmentation, adjacent candidates separated by fewer than five frames are merged, with the later boundary retained. This operation suppresses short fragments caused by multiple signals responding to the same transition.

Stage 3: Interaction-aware merging. The resulting intervals are further refined using contact, gripper, motion, and torque-related information. Consecutive intervals belonging to the same object interaction are merged into a coherent phase. Direction-aware processing is also applied to non-interaction intervals: approach and intermediate transit motion are preserved when relevant, whereas task-completion reset motion can be removed.

Stage 4: Static-segment pruning. Intervals containing a stationary run of at least eight frames are removed unless they still exhibit meaningful accumulated positional or rotational changes. This step suppresses low-information pauses while retaining slow but purposeful manipulation. The final retained interval is extended when necessary to cover the end of the episode.

Phase-count checking and refinement. For each task variation, experts first inspect representative demonstration videos and use task-specific knowledge to determine the expected number of phases in its episodes. This expert-provided phase-count prior is incorporated into the automated pipeline as an acceptance criterion. After each segmentation attempt, the script compares the detected phase count with the corresponding prior; a mismatch automatically triggers another segmentation cycle, up to the configured retry limit. This iterative process reduces over- and under-segmentation caused by transient gripper, motion, contact, or collection noise. Most episodes can therefore be processed automatically, while only the remaining ambiguous cases require lightweight manual boundary refinement, substantially improving segmentation reliability with limited annotation effort. After the phase boundaries are finalized, experts assign an atomic-action type to each phase according to the definitions in Appendix A.11.

Language annotation and dataset assembly. Each annotated phase segment and its corresponding atomic-action definition are combined with the task description and local phase context to form the language-generation prompt. GPT-4o then generates eight phase-specific English descriptions for every atomic-action phase. The complete prompt structure and an example output are provided in Appendix A.12.

Finally, the annotated phases are reorganized into an action-first hierarchy indexed by atomic action, task, variation, and phase. Each AtomAction sample contains the complete low-dimensional robot-state trajectory of the phase; five-view RGB, depth, and mask observations at the phase boundaries; task and variation information; the atomic-action label and phase metadata; and eight phase-level English descriptions. Together, these components constitute the final AtomAction Dataset.

A.7 Comparison with AtomicVLA Data

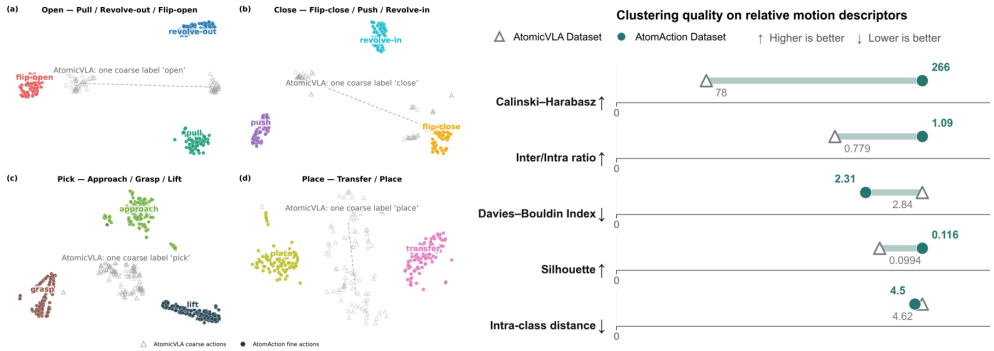


Figure 10: Motion-structure comparison between AtomicVLA (Zhang et al., 2026b) and AtomAction. Left: four action families. Right: clustering metrics. Gray triangles denote AtomicVLA (Zhang et al., 2026b) coarse labels; colored circles denote AtomAction fine-grained labels.

AtomicVLA (Zhang et al., 2026b) constructs atomic-action annotations by analyzing relative end-effector translation, rotation, and gripper changes within short motion windows. Principal-axis analysis yields initial temporal segments and coarse labels, which are then refined from video clips using InternVideo2.5 (Wang et al., 2025). In its LIBERO (Liu et al., 2023) setting, trajectories are organized into five action abstractions: Pick, Place, Open, Close, and Turn. Each segment retains its visual observations, robot actions, task instruction, temporal interval, and coarse label.

However, this representation remains coarse-grained. A single label often mixes different motion mechanisms or execution stages. For example, Open combines translational pulling, outward hinge rotation, and lid opening, whereas Pick combines approaching, grasping, and lifting. Segments sharing an AtomicVLA (Zhang et al., 2026b) label therefore remain heterogeneous in motion structure, limiting annotation precision at the mechanism level. AtomAction addresses this with 18 phase-level action types, including Pull, Revolve-out, Flip-open, Approach, Grasp, Lift, Transfer, and Place.

We further conduct a quantitative comparison. From the Open, Close, Pick, and Place families in both datasets, we extract independent trajectory segments, convert them into 32-step, seven-channel relative end-effector sequences, and construct 52-dimensional motion descriptors. The left panels illustrate how AtomAction distinguishes motion mechanisms and execution stages within coarse action families: Open separates into Pull, Revolve-out, and Flip-open, while Pick separates into Approach, Grasp, and Lift. These finer labels correspond to distinct motion groups in the visualization. Consistent with this pattern, the quantitative results on the right show higher Calinski–Harabasz (266 vs. 78), inter-/intra-class ratio (1.09 vs. 0.779), and silhouette (0.116 vs. 0.0994) scores for AtomAction, together with a lower Davies–Bouldin index (2.31 vs. 2.84) and a modest reduction in intra-class distance (4.50 vs. 4.62). These results indicate greater between-class separability and slightly improved within-class compactness under the selected subsets and shared motion representation.

Taken together, these findings suggest that, compared with AtomicVLA's (Zhang et al., 2026b) coarse action annotations, AtomAction provides finer distinctions between manipulation mechanisms and execution stages, with labels that better reflect the structure of the underlying motion. This finer granularity supports more precise grouping and retrieval of atomic actions and offers a basis for aligning related actions across tasks. AtomAction thus provides a structured source of data for future research on atomic action representation learning, cross-task skill transfer, and compositional robot learning.

A.8 Simulation Setup and Evaluation Protocol

Task Split and Data Coverage. The seen set consists of the standard 18 RLBench tasks used for policy fine-tuning. The policy-unseen set contains eight additional tasks excluded from policy training. This setting evaluates whether pretrained atomic knowledge can support downstream execution of policy-unseen tasks. Table 8 lists the task groups; policy-unseen tasks are grouped by the atomic action they involve.

Table 8: **Simulation task groups and training coverage.**

Seen tasks (18)		
Close jar	Light bulb in	Open drawer
Place cups	Place shape in shape sorter	Push buttons
Put groceries in cupboard	Reach and drag	Slide block to color target
Stack blocks	Place wine at rack location	Sweep to dustpan of size
Insert onto square peg	Meat off grill	Put item in drawer
Put money in safe	Stack cups	Turn tap
Policy-unseen tasks (8)		
Articulated-object manip. (2)	*Object transport & placement (4)*	*Switch operation (2)*
Close drawer	Pick up cup	Lamp on
Open jar	Phone on base	Press switch
	Basketball in hoop	
	Put knife on chopping board	

Atomic Pretraining and Policy Adaptation. We pretrain the codebooks and adapter on Atom-Action and freeze both during downstream policy adaptation. The codebooks contain 36 global entries and 192 detail entries, each with 512 dimensions. During policy fine-tuning, LexiconVLA conditions execution on the task instruction, the current subtask instruction, and retrieved action codes. Code retrieval uses observations available at the subtask boundary, without access to future observations.

BridgeVLA Fine-tuning. All BridgeVLA comparison arms initialize from the official RLBench checkpoint. Training starts with 100 demonstrations for each seen task. The base policy, subtask-only control, and full LexiconVLA use the same demonstrations, sample order, and optimization budget. PaliGemma remains frozen; the action head and, where present, the code injector are trainable. Table 9 summarizes the configuration.

Table 9: Fine-tuning configuration for the BridgeVLA comparison arms. These settings are specific to BridgeVLA.

Setting	**Value**
Optimizer	AdamW
Learning rate	8×10^{-5}
Batch size	8
Training epochs	3
Optimizer steps	17,904
Training seed	20260914
Frozen modules	PaliGemma, codebooks, adapter
Trainable modules	Action head and optional code injector

Test-Time Execution. Each task is evaluated on 25 episodes initialized from test demonstrations, with identical initial scenes across comparison arms. Each episode permits at most 25 keyframe actions. LexiconVLA uses an online planner to schedule subtasks. At each subtask entry, the adapter retrieves codes from the current observation and subtask instruction; these codes are reused within the subtask. All parameters remain fixed throughout evaluation. An episode succeeds only when RLBench's task-level success condition is satisfied. Timeouts and execution failures count as unsuccessful episodes and remain in the denominator. Unseen, seen, and overall success rates average per-task success rates with equal weight across tasks, and we report their mean and standard deviation over three random seeds.

Compared Methods

- **Simulation Baselines.** All simulation baselines are keyframe-based 3D manipulation policies developed for, and evaluated on, the multi-task RLBench (James et al., 2020) setting. We apply LexiconVLA to five of them, covering three architectural families: PerAct (Shridhar et al., 2022), a Perceiver transformer over voxelized observations; RVT (Goyal et al., 2023) and RVT-2 (Goyal et al., 2024), multi-view transformers over virtual renderings of the scene point cloud, with RVT-2 adding coarse-to-fine refinement; and BridgeVLA (Li et al., 2025) and BridgeVLA++ (Li et al., 2026), which build on a pretrained VLM by projecting 3D inputs to multi-view images and predicting 2D heatmaps, with BridgeVLA++ adding memory augmentation. Each base policy and its LexiconVLA variant are evaluated on identical initial scenes. Act3D (Gervet et al., 2023), a 3D feature-field transformer; 3D Diffuser Actor (Ke et al., 2024), a diffusion policy over 3D scene representations; and SAM2Act (Fang et al., 2025), a multi-view policy built on SAM2 visual features, serve as standalone references for absolute performance.
- **Generalist VLAs.** Generalist VLAs such as π_0 (Black et al., 2026) and $\pi_{0.5}$ (Intelligence et al., 2025) are not included in the simulation comparison. Their public releases provide no RLBench checkpoints, and they predict continuous action chunks at control frequency, whereas the RLBench protocol followed by all compared methods predicts sparse keyframe poses executed by a motion planner. Including them would require re-designing their action interface and training them on RLBench under our own choices, so the results would reflect our adaptation rather than the released models. On the real robot, this mismatch does not arise: $\pi_{0.5}$ can be fine-tuned with its standard pipeline on the same Franka Research 3 demonstrations as the other real-world methods. We therefore include it in the real-world experiments as a generalist VLA reference with a different architecture and action representation, complementing BridgeVLA, which serves as the common base policy in both settings.

A.9 Real-World Setup and Evaluation Protocol

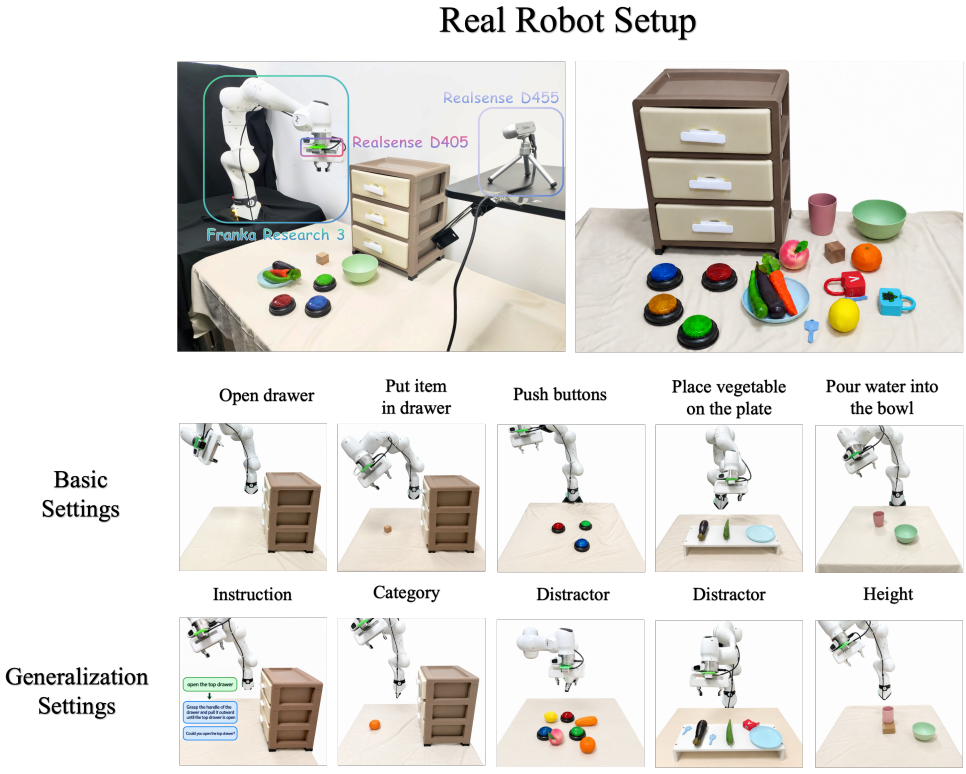


Figure 11: Real-robot platform and experimental settings. Top: the robot setup and all objects used in the experiments. Middle, from left to right: open drawer, put item in drawer, push buttons, place items on a plate, and pour water into a bowl. Bottom: the corresponding generalization settings involving instruction, category, distractor, and height.

Platform and Demonstrations. The real-world experimental setup is illustrated in Fig. 11. All experiments are conducted on a 7-DoF Franka Research 3 (FR3) robotic arm equipped with a parallel-jaw gripper. Visual observations are captured from two complementary viewpoints: a RealSense D455 camera provides a fixed third-person view of the workspace, while a RealSense D405 camera is mounted on the robot wrist to capture close-range interaction details. To improve visual clarity and better emphasize the robot, manipulated objects, and task-relevant regions, we remove irrelevant background content and retain only the principal foreground elements in the presented figures. This post-processing is applied solely for visualization and does not alter the visual observations used by the policy during training or evaluation.We collect 30 demonstrations per task, covering all task variants in Table 10. The Standard setting uses the objects, scene layout, and instructions of the demonstrations.

Compared Methods and Training. All methods are fine-tuned on the same demonstrations. BridgeVLA and LexiconVLA (BridgeVLA) are initialized from the same pretrained checkpoint, and $\pi_{0.5}$ is fine-tuned with its standard pipeline from its released base checkpoint; Appendix A.8 explains why $\pi_{0.5}$ is included only in real-world experiments. LexiconVLA keeps the codebooks learned on AtomAction frozen and uses the same planner and scheduling skills as in simulation (Appendix A.4). No parameters are updated at test time in either setting.

Standard Tasks. Table 10 lists the five tasks and their success criteria . Fig. 11 shows the real-world scenes for the five standard tasks. Open drawer, put item in drawer, and push buttons have counterparts among the seen RLBench tasks (Sim), whereas pour water into the bowl and place fruits on the plate have none (Real-only). Push buttons and place fruits on the plate are treated as long-horizon because their instructions specify multiple targets that must be handled in sequence, so a failure at any target fails the trial.

Table 10: **Real-world tasks, variants, and success criteria.** Groups follow Table 3.

Task	Group	Variants	Success criterion
Open drawer	Sim-Short	Top / middle drawer	The specified drawer is pulled open.
Put item in drawer	Sim-Short	Top / middle drawer	The specified drawer is opened and the block rests inside it.
Push buttons	Sim-Long	Red; red→blue; red→blue→green	All specified buttons are pressed in the given order.
Pour water into the bowl	Real-only-Short	Single setup	The cup is grasped, carried to the bowl, and tilted to pour into it without being knocked over.
Place fruits on the plate	Real-only-Long	Pepper + eggplant; pepper + carrot; eggplant + carrot	Both items rest on the plate at the end of the trial.

Generalization Settings. Each perturbation is applied to the task listed in Table 3 and is absent from the demonstrations.Fig 11 shows the real-world scenes under the generalization settings.

- **Distractor** (push buttons; place fruits on the plate). One or two distractors are placed near the targets, including objects of the target category (an additional yellow button; additional fruits such as an orange and a peach) and unrelated objects (fruits among the buttons; toys near the plate). This setting tests whether execution remains bound to the instructed targets.
- **Height** (pour water into the bowl). The manipulated objects are raised on supports of two different heights, shifting the grasping and pouring poses away from those demonstrated.
- **Category** (put item in drawer). The wooden block used in the demonstrations is replaced by objects of unseen categories (an orange and a lemon), which differ from the training item in shape, size, and surface.
- **Instruction** (open drawer). The training instruction is replaced by a detailed procedural command, "Grasp the handle of the drawer and pull it outward until the top/middle drawer is open," and a conversational request, "Could you open the drawer?"

Evaluation Protocol. Each method is evaluated over 10 trials per task and setting. When a task has several variants, trials alternate among them, and all methods follow the same variant sequence. A trial succeeds only if the entire instruction is completed; partial completion, such as pressing only part of a button sequence or leaving one item off the plate, counts as failure. Trials terminated by collisions, joint or workspace limits, or controller faults also count as failures and remain in the denominator. The human-disturbance rollouts in Figure 5 are qualitative and are not included in Table 3.

A.10 Detailed Real-World Analysis

Table 11: **Aggregated real-world success rates (%).** Group averages of Table 3. Each task appears once in each setting, so both All columns cover the same five tasks.

Method	Standard			Generalization
	Sim (3)	Real-only (2)	All (5)	All (5)
BridgeVLA	36.7	10.0	26.0	16.0
$\pi_{0.5}$	46.7	0.0	28.0	12.0
LexiconVLA (Ours)	**73.3**	**45.0**	**62.0**	**56.0**

Standard Setting. As summarized in Table 11, LexiconVLA outperforms both baselines on tasks with and without simulation counterparts, and its advantage grows with the number of chained atomic actions. On open drawer, LexiconVLA succeeds in every trial, compared with 50.0% for BridgeVLA and 60.0% for $\pi_{0.5}$. Put item in drawer appends grasping, transferring, and placing an item to the same opening interaction. Here both baselines fail in all trials, often already at the opening stage by missing the handle or pulling a different drawer. An interaction a policy

performs in one task is thus not reliably invoked within another, in line with our diagnostic probe (Appendix A.1). Most remaining failures of LexiconVLA occur after the drawer is opened, while grasping or transferring the item. On place fruits on the plate, which repeats pick-and-place for two items, several LexiconVLA failures are partial completions in which the second item falls off the plate; these count as failures under our criterion. On pour water into the bowl, BridgeVLA frequently knocks the cup over or reaches joint limits, and $\pi_{0.5}$ fails in all trials, in some cases executing the pouring motion without a stable grasp on the cup. Push buttons is the only task on which a baseline matches LexiconVLA ($\pi_{0.5}$, 80.0%), and this parity disappears under distractors.

Generalization Setting. Because each task is perturbed exactly once, the Generalization averages are directly comparable with the Standard ones: LexiconVLA retains about 90% of its Standard success rate (56.0% vs. 62.0%), compared with about 62% for BridgeVLA and 43% for $\pi_{0.5}$.

- **Instruction.** Paraphrases lower $\pi_{0.5}$ from 60.0% to 20.0% and BridgeVLA from 50.0% to 40.0% on open drawer, whereas LexiconVLA keeps 80.0%. Several $\pi_{0.5}$ failures pull a drawer other than the instructed one. For LexiconVLA, the planner maps both paraphrases onto the same atomic subtasks, so the subtask instructions and retrieved codes that condition execution remain largely unchanged.
- **Category.** Replacing the wooden block with an orange or a lemon leaves LexiconVLA at 40.0%, its Standard success on this task, with failures arising mainly from grasping the round, smooth objects. As both baselines already fail in the Standard setting, this condition mainly shows that the success of LexiconVLA is not tied to the demonstrated object, consistent with codes that encode the interaction rather than object identity.
- **Height.** LexiconVLA reaches 80.0% on raised objects, whereas both baselines fail in all trials. Its success exceeds that in the Standard setting partly because the raised placement reduces reachability failures at table height; the relevant comparison is therefore across methods under the same condition. Baseline failures, such as closing the gripper before reaching the cup, colliding with the support, or pouring in place, suggest motions tied to the demonstrated heights, whereas LexiconVLA retrieves codes from the current observation at each subtask boundary.
- **Distractor.** Distractors are the most challenging perturbation for LexiconVLA, lowering its success from 80.0% to 60.0% on push buttons and from 50.0% to 20.0% on place fruits on the plate. It nevertheless leads on both tasks, while $\pi_{0.5}$ drops from 80.0% to 40.0% on push buttons. Failures of all methods mainly involve acting on a distractor, such as pressing a fruit placed among the buttons or confusing the pepper with a similar-looking item. The codes specify which interaction to perform but not which object to act on, so target binding under same-category distractors still relies on the planner and base policy; object-centric grounding is a natural extension.

Failure Recovery. Figure 5 shows two forms of recovery. Without external intervention, a failed grasp in put the ball in the hoop is detected by the planner, which issues RETRY and re-retrieves codes from the updated observation. Under human disturbance in put item into the drawer, the planner likewise returns to the grasp subtask and completes the task, whereas the $\pi_{0.5}$ rollout fails under the same disturbance. In contrast, baselines can stall after a failed interaction; for example, $\pi_{0.5}$ repeatedly closes its gripper near a missed drawer handle without re-approaching it. Recovery does not guarantee completion, however, as subsequent grasps or transfers can still fail.

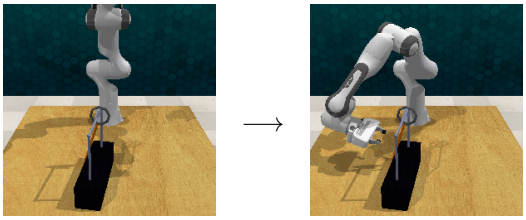
A.11 Atomic-Action Standard

This appendix specifies the operational meaning of each released action label. Each entry pairs a normalized definition with the start and terminal frames of the reference example provided in the annotation workbook. The images illustrate representative state changes; the label definitions, rather than scene appearance alone, determine annotation.

A.11.1 APPROACH

Reference task: *Beat the buzz*

Definition: The end effector moves toward a target object or contact site without making physical contact. Its orientation may change during the motion, and the phase ends at a preparatory pose for the next interaction.

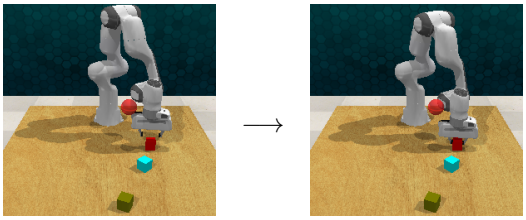


Start frame End frame

A.11.2 GRASP

Reference task: *pick and lift*

Definition: From a pre-grasp pose, the end effector advances until contact and closes the gripper around the target object. The phase establishes a stable hold through friction or geometric constraint.

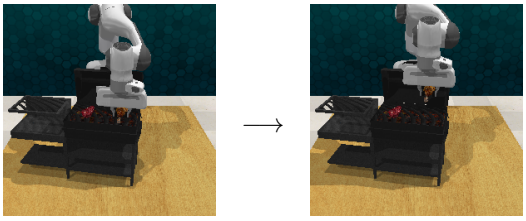


Start frame End frame

A.11.3 LIFT

Reference task: *meat off grill*

Definition: While maintaining a stable grasp, the end effector moves the target object upward until it is fully separated from its initial support surface.

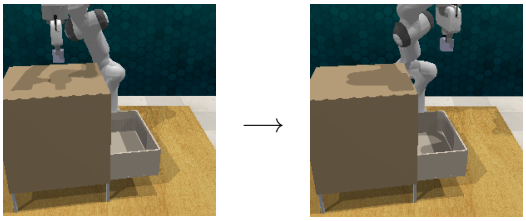


Start frame End frame

A.11.4 TRANSFER

Reference task: *put item in drawer*

Definition: While maintaining a stable grasp after lift, the end effector carries the object from its current location toward a pre-placement pose near the target region.

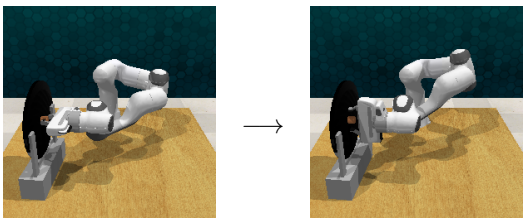


Start frame End frame

A.11.5 ROTATE

Reference task: *change clock*

Definition: While maintaining a stable grasp, the end effector changes the object's orientation through angular motion about an axis. The object's overall position remains approximately fixed unless a more specific interaction label applies.

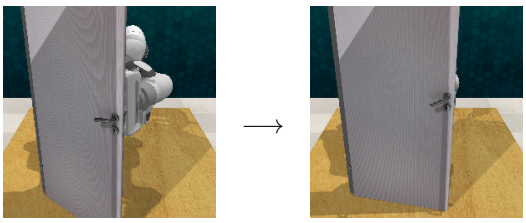


Start frame End frame

A.11.6 REVOLVE-OUT

Reference task: *close door*

Definition: The end effector applies tangential motion to a component constrained by a fixed axis, such as a hinge, causing it to rotate outward and away from its closed or initial configuration.

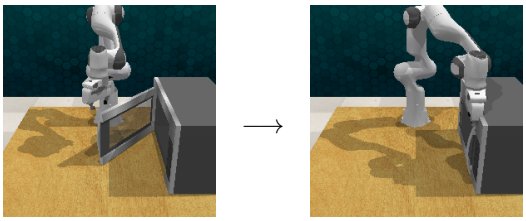


Start frame End frame

A.11.7 REVOLVE-IN

Reference task: *close microwave*

Definition: The end effector applies tangential motion to a component constrained by a fixed axis, causing it to rotate inward toward its closed configuration. Contact may be maintained without a grasp when pushing is sufficient.

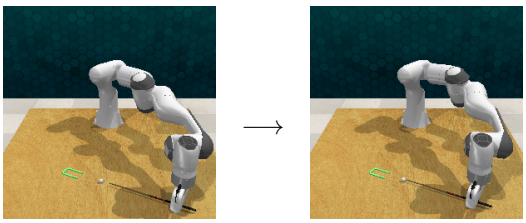


Start frame End frame

A.11.8 PUSH

Reference task: *close drawer*

Definition: After contact is established, the end effector applies force away from itself along a constrained direction, producing translational or mechanically constrained displacement of the target object or mechanism.

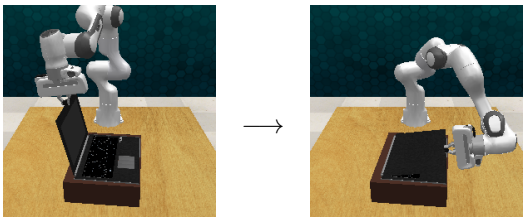


Start frame End frame

A.11.9 FLIP-CLOSE

Reference task: *Close laptop lid*

Definition: The end effector acts on a component that rotates about a predominantly horizontal hinge or pivot, moving it downward from an open state until it reaches the closed configuration.

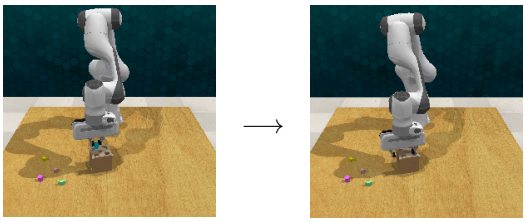


Start frame End frame

A.11.10 PLACE

Reference task: *place shape*

Definition: From a pre-placement pose, the end effector aligns and lowers a held object onto a target support or into a target region, establishing the intended final support relationship.

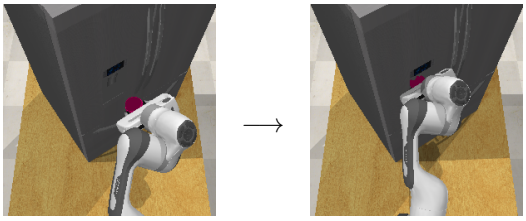


Start frame End frame

A.11.11 PRESS

Reference task: *get ice from fridge*

Definition: After establishing contact with a control or surface, the end effector applies a short displacement and controlled force along the contact normal, typically until a mechanical travel limit is reached.

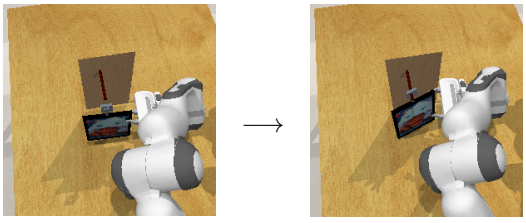


Start frame End frame

A.11.12 HANG

Reference task: *hang frame on hanger*

Definition: The end effector aligns an object's hook, loop, hole, or mounting feature with a fixed support and seats it so that the object remains suspended by that support.

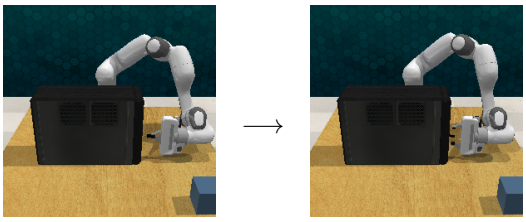


Start frame End frame

A.11.13 INSERT

Reference task: *insert usb in computer*

Definition: The end effector aligns a held object with a compatible hole, slot, or interface and advances it along the insertion axis until the intended geometric engagement is established.

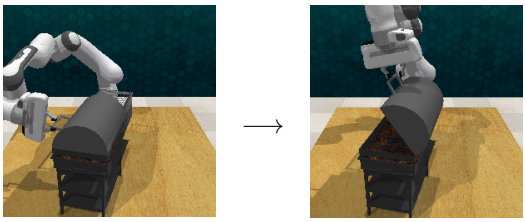


Start frame End frame

A.11.14 FLIP-OPEN

Reference task: *Close grill*

Definition: The end effector maintains contact with a lid, cover, flap, or similar component and rotates it about a predominantly horizontal hinge or pivot from a closed state to an open state.

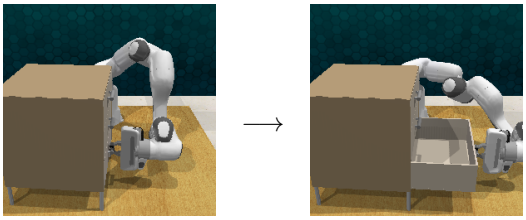


Start frame End frame

A.11.15 PULL

Reference task: *open drawer*

Definition: While maintaining a stable grasp on an object or mechanism handle, the end effector moves toward itself along a constrained axis, opening, extending, or extracting the target.

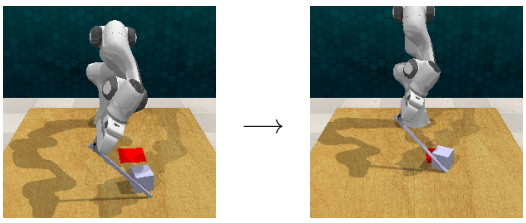


Start frame End frame

A.11.16 SLIDE

Reference task: *reach and drag*

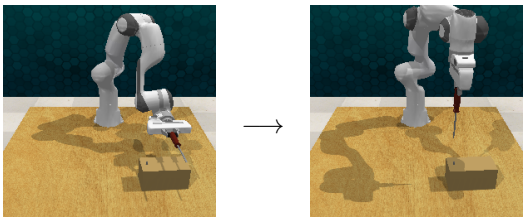
Definition: The end effector maintains a grasp or firm contact while moving an object or sliding mechanism along its constrained linear axis without lifting it from that constraint.



Start frame End frame

A.11.17 POSE-ADJUST

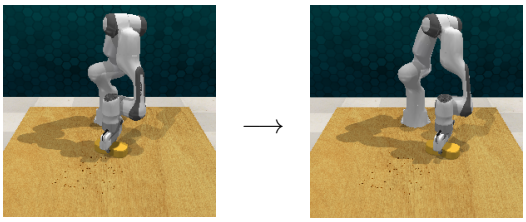
Reference task: *screw nail*
Definition: The end effector performs a local correction to its pose or to the pose of a grasped object so that the following interaction can proceed from a better-aligned configuration.



Start frame End frame

A.11.18 WIPE

Reference task: *wipe desk*
Definition: The end effector maintains continuous surface contact, directly or through a held tool, while sweeping across a region to produce a covering or cleaning motion.



Start frame End frame

A.12 PROMPT TEMPLATES

This appendix collects the prompt templates used in this work, all abbreviated for presentation. Table 12 presents the phase-level language generation prompt, which produces the eight English descriptions attached to every atomic-action segment of the dataset. The remaining two templates drive the online closed loop execution, which queries the VLM planner with two prompts: PLAN(Table 13) decomposes the remaining task into atomic-action subtasks, and MONITOR (Table 14) selects a scheduling skill for the active subtask. The scheduling skills and the call, retry, and extension limits referenced below follow Appendix A.4.

A.13 PER-TASK SIMULATION RESULTS

Tables 15 and 16 list the per-task success rates underlying Table 1. Each entry is the mean$_{\pm\text{std}}$ over three random seeds with 25 episodes per task and seed. The Avg. column averages the per-task means with equal weight and therefore reproduces the Unseen and Seen columns of Table 1 up to rounding; its std is computed across the three seed-level averages.

Table 12: Prompt template and example for phase-level language generation.

Role:

You are annotating phases for an RLBench robot manipulation dataset. Generate reusable English descriptions for each exact phase. You must be accurate, diverse in wording and form, and strictly output valid JSON.

Input:

The input is a JSON object containing:

1. Task and variation information, including task, variation index, and variation descriptions.
2. The complete phase label sequence and one item for every phase. Each item contains phase index, action label, previous action label, next action label, repeated action label, and same action phase indices.
3. The descriptions per phase value and the list of required styles.

Example input: task = open drawer; variation = 1; phase sequence = [approach, grasp, pull]; descriptions per phase = 8.

Task:

Generate phase-level English descriptions for every item in the input JSON.

Requirements:

1. Return JSON only, with the top-level key phase descriptions.
2. phase descriptions must contain one entry for every input item, ordered by phase index.
3. Each entry must contain phase index, action label, and descriptions.
4. descriptions must contain exactly one object for each required style.
5. Each description must stay correct for this exact phase only, not the whole task.
6. Keep the writing varied in wording, sentence form, and abstraction level.
7. Mention the action, manipulated object, local target, or immediate subgoal when supported.
8. Do not invent unsupported details such as color, material, left/right, exact spatial layout, or hidden states.
9. If the same action label appears in multiple phases, distinguish the target phase using its phase index and neighboring labels.
10. All text must be English.

Required styles:

short, clear, object focused, goal focused, detailed, abstract, natural, and instructional.

Output Format:

The response contains the top-level key phase descriptions. Each phase entry contains phase index, action label, and eight {style, text} description objects.

Example Output:

Task: open drawer **Phase:** 2 **Action:** pull

Short: Pull the middle drawer open.
Clear: Draw the middle drawer outward using the grasped handle.
Object-focused: The middle drawer is pulled outward by its handle.
Goal-focused: Open the middle drawer by pulling on the handle.
Detailed: With the handle already grasped, move the gripper outward to slide the middle drawer open.
Abstract: Convert the secured grip into an opening motion.
Natural: Pull back on the handle to open the middle drawer.
Instructional: Pull the grasped middle drawer handle outward until the drawer opens.

Table 13: Abbreviated PLAN prompt used to decompose a task instruction into atomic-action subtasks.

Role:

You are a task planner for a robot arm working on a tabletop. Given the current front and wrist views, decompose the remaining task into an ordered sequence of atomic actions. Return a single JSON object only.

Constraints:

1. Choose action labels from the predefined vocabulary: approach, grasp, (remaining atomic action labels omitted). Use pose-adjust only when no other action applies.
2. Each instruction describes one step: lowercase, no trailing punctuation, 3–8 words, imperative mood. Preserve distinguishing colours, ordinals, and spatial references.
3. Use the prescribed initial verbs and do not substitute synonyms: approach → move toward, approach; grasp → grasp; (remaining action-to-verb mappings omitted).
4. Plan only unfinished work, typically 3–8 steps. Treat the reference sequence as a soft prior, and obey any supplied repetition count exactly.

Action definitions:

approach: Move toward the target without contact, stopping at a preparatory pose. grasp: Establish contact and close the gripper to secure the object. (Remaining action definitions omitted.)

Instruction selection: prefer A over B over C.

A. Reuse a listed instruction via its phrasing identifier.
B. Reuse its structure and substitute only the differing object or variation words.
C. If no entry applies, imitate the listed grammar, length, and vocabulary.

Output format:

{"plan": [<step>, ...]}, where each step takes one of the following forms:

{"action": "grasp", "phrasing_id": <id>}

{"action": "grasp", "phrasing_id": <id>, "substitute": {"<old>": "<new>"}}

{"action": "<action>", "instruction": "<subtask instruction>"}

User message:

task: {task_name}

task instruction: {task_instruction}

reference action sequence (soft prior): {reference_sequence}

phrasings from the training cache:

[0] {action_0}: {instruction_0}

[1] {action_1}: {instruction_1}

(Remaining candidate entries omitted.)

{repetition_constraint_if_applicable}

{completed_steps_if_extending}

{attempt_history_if_enabled}

front view: {front_image} wrist view: {wrist_image}

Table 14: Abbreviated MONITOR prompt used to assess execution of the active subtask and to select a scheduling skill.

Role:

You are a progress monitor for a robot arm executing a plan. Assess the current images, the active step, the elapsed keyframes, and the gripper state. Return a single JSON object only.

Scheduling skills:

CONTINUE: The active step is still in progress and remains achievable.

NEXT: Its goal is visibly achieved; advance to the next step.

RETRY: A step failed. Provide an index to repeat an earlier step, or omit it to retry the current step.

REPLAN: The environment changed and invalidated the plan; regenerate it from the current scene, discarding progress.

EXTEND: Available on the final step when the task remains unfinished. Append the remaining steps while preserving completed work, and use EXTEND instead of NEXT at the end of the plan.

Guidance:

Judge completion from the images rather than from step counts. Do not replan merely because the arm has not reached its target. Avoid indefinite CONTINUE decisions and advance once completion is visible. Prefer NEXT or CONTINUE when either applies.

Output format:

`{"decision": "<skill>", "index": <optional step index>, "reason": "<at most 12 words>"}`

User message:

`task: {task_name}` `task instruction: {task_instruction}`
`gripper: {gripper_state}` `keyframes spent on the current step: {elapsed_keyframes}`
`plan:`
`[0] {action_0}: {instruction_0}`
`[1] {action_1}: {instruction_1}`
(Remaining steps omitted; the active step is marked with `<-- CURRENT`.)
`{gripper_semantics_note_if_applicable}`
`{final_step_note_if_applicable}`
`{progress_warning_if_applicable}`
`{attempt_history_if_enabled}`
`front view: {front_image}` `wrist view: {wrist_image}`

Table 15: **Per-task success rate (%) on the 8 unseen tasks.** No method is trained on these tasks. Mean$_{\pm\text{std}}$ over three random seeds. Best result per column in bold. Shaded rows are LexiconVLA variants.

Method	Avg.	Basketball in Hoop	Close Drawer	Knife on Board	Lamp On	Open Jar	Phone on Base	Pick up Cup	Press Switch
Act3D	$15.7_{\pm 2.5}$	$1.3_{\pm 2.3}$	$68.0_{\pm 4.0}$	$1.3_{\pm 2.3}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$18.7_{\pm 4.6}$	$36.0_{\pm 10.6}$
3D Diffuser Actor	$28.7_{\pm 0.3}$	$0.0_{\pm 0.0}$	$\mathbf{94.7}_{\pm 4.6}$	$5.3_{\pm 2.3}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$2.7_{\pm 2.3}$	$42.7_{\pm 2.3}$	$\mathbf{84.0}_{\pm 6.9}$
SAM2Act	$15.5_{\pm 1.5}$	$0.0_{\pm 0.0}$	$42.7_{\pm 2.3}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$1.3_{\pm 2.3}$	$65.3_{\pm 10.1}$	$14.7_{\pm 8.3}$
PerAct	$19.0_{\pm 1.5}$	$0.0_{\pm 0.0}$	$45.3_{\pm 8.3}$	$6.7_{\pm 6.1}$	$6.7_{\pm 6.1}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$20.0_{\pm 4.0}$	$73.3_{\pm 8.3}$
LexiconVLA (PerAct)	$19.3_{\pm 0.6}$	$0.0_{\pm 0.0}$	$45.3_{\pm 9.2}$	$4.0_{\pm 4.0}$	$6.7_{\pm 4.6}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$30.7_{\pm 4.6}$	$68.0_{\pm 6.9}$
RVT	$11.8_{\pm 2.3}$	$0.0_{\pm 0.0}$	$52.0_{\pm 8.0}$	$\mathbf{8.0}_{\pm 4.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$28.0_{\pm 4.0}$	$6.7_{\pm 4.6}$
LexiconVLA (RVT)	$14.5_{\pm 1.8}$	$0.0_{\pm 0.0}$	$49.3_{\pm 6.1}$	$5.3_{\pm 4.6}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$33.3_{\pm 12.2}$	$28.0_{\pm 4.0}$
RVT-2	$18.0_{\pm 0.5}$	$0.0_{\pm 0.0}$	$64.0_{\pm 4.0}$	$4.0_{\pm 0.0}$	$1.3_{\pm 2.3}$	$0.0_{\pm 0.0}$	$6.7_{\pm 8.3}$	$42.7_{\pm 6.1}$	$25.3_{\pm 4.6}$
LexiconVLA (RVT-2)	$19.8_{\pm 0.8}$	$1.3_{\pm 2.3}$	$73.3_{\pm 2.3}$	$4.0_{\pm 4.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$1.3_{\pm 2.3}$	$62.7_{\pm 10.1}$	$16.0_{\pm 6.9}$
BridgeVLA++*	$19.2_{\pm 2.0}$	$0.0_{\pm 0.0}$	$70.7_{\pm 10.1}$	$5.3_{\pm 6.1}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$2.7_{\pm 4.6}$	$54.7_{\pm 8.3}$	$20.0_{\pm 6.9}$
LexiconVLA (BridgeVLA++)	$26.8_{\pm 1.2}$	$22.7_{\pm 12.2}$	$84.0_{\pm 0.0}$	$6.7_{\pm 2.3}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$9.3_{\pm 2.3}$	$62.7_{\pm 12.2}$	$29.3_{\pm 8.3}$
BridgeVLA	$16.7_{\pm 1.3}$	$9.3_{\pm 4.6}$	$44.0_{\pm 0.0}$	$4.0_{\pm 0.0}$	$2.7_{\pm 2.3}$	$0.0_{\pm 0.0}$	$4.0_{\pm 4.0}$	$57.3_{\pm 12.9}$	$12.0_{\pm 4.0}$
LexiconVLA (BridgeVLA)	$\mathbf{34.2}_{\pm 2.0}$	$\mathbf{36.0}_{\pm 6.9}$	$61.3_{\pm 2.3}$	$\mathbf{8.0}_{\pm 4.0}$	$\mathbf{56.0}_{\pm 0.0}$	$0.0_{\pm 0.0}$	$\mathbf{24.0}_{\pm 8.0}$	$\mathbf{82.7}_{\pm 8.3}$	$5.3_{\pm 2.3}$

Table 16: **Per-task success rate (%) on the 18 seen tasks.** Mean$_{\pm\text{std}}$ over three random seeds. Best result per column in bold. Shaded rows are LexiconVLA variants. The Avg. column covers all 18 tasks.

Method	Avg.	Close Jar	Drag Stick	Insert Peg	Meat off Grill	Open Drawer	Place Cups	Place Wine	Push Buttons	Put in Cupboard
Act3D	$65.2_{\pm 1.2}$	$\mathbf{100.0}_{\pm 0.0}$	$66.7_{\pm 8.3}$	$13.3_{\pm 2.3}$	$\mathbf{100.0}_{\pm 0.0}$	$90.7_{\pm 9.2}$	$8.0_{\pm 4.0}$	$65.3_{\pm 4.6}$	$\mathbf{100.0}_{\pm 0.0}$	$65.3_{\pm 6.1}$
3D Diffuser Actor	$78.3_{\pm 1.9}$	$96.0_{\pm 0.0}$	$98.7_{\pm 2.3}$	$60.0_{\pm 8.0}$	$\mathbf{100.0}_{\pm 0.0}$	$77.3_{\pm 12.2}$	$33.3_{\pm 6.1}$	$90.7_{\pm 6.1}$	$\mathbf{100.0}_{\pm 0.0}$	$81.3_{\pm 6.1}$
SAM2Act	$84.3_{\pm 1.8}$	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{100.0}_{\pm 0.0}$	$80.0_{\pm 0.0}$	$\mathbf{100.0}_{\pm 0.0}$	$90.7_{\pm 4.6}$	$33.3_{\pm 4.6}$	$96.0_{\pm 0.0}$	$\mathbf{100.0}_{\pm 0.0}$	$65.3_{\pm 8.3}$
PerAct	$37.1_{\pm 2.1}$	$34.7_{\pm 4.6}$	$90.7_{\pm 2.3}$	$0.0_{\pm 0.0}$	$88.0_{\pm 4.0}$	$81.3_{\pm 8.3}$	$0.0_{\pm 0.0}$	$50.7_{\pm 15.1}$	$26.7_{\pm 2.3}$	$6.7_{\pm 2.3}$
LexiconVLA (PerAct)	$39.5_{\pm 2.1}$	$9.3_{\pm 2.3}$	$81.3_{\pm 4.6}$	$4.0_{\pm 4.0}$	$78.7_{\pm 2.3}$	$76.0_{\pm 10.6}$	$0.0_{\pm 0.0}$	$50.7_{\pm 2.3}$	$37.3_{\pm 2.3}$	$17.3_{\pm 6.1}$
RVT	$59.3_{\pm 2.0}$	$36.0_{\pm 4.0}$	$92.0_{\pm 0.0}$	$26.7_{\pm 8.3}$	$97.3_{\pm 2.3}$	$77.3_{\pm 6.1}$	$1.3_{\pm 2.3}$	$92.0_{\pm 4.0}$	$\mathbf{100.0}_{\pm 0.0}$	$36.0_{\pm 8.0}$
LexiconVLA (RVT)	$60.8_{\pm 1.1}$	$28.0_{\pm 4.0}$	$94.7_{\pm 2.3}$	$13.3_{\pm 2.3}$	$93.3_{\pm 2.3}$	$82.7_{\pm 4.6}$	$1.3_{\pm 2.3}$	$88.0_{\pm 6.9}$	$\mathbf{100.0}_{\pm 0.0}$	$56.0_{\pm 8.0}$
RVT-2	$75.6_{\pm 1.0}$	$18.7_{\pm 6.1}$	$98.7_{\pm 2.3}$	$26.7_{\pm 6.1}$	$\mathbf{100.0}_{\pm 0.0}$	$85.3_{\pm 6.1}$	$50.7_{\pm 6.1}$	$92.0_{\pm 8.0}$	$96.0_{\pm 0.0}$	$57.3_{\pm 2.3}$
LexiconVLA (RVT-2)	$76.3_{\pm 0.3}$	$24.0_{\pm 4.0}$	$94.7_{\pm 2.3}$	$44.0_{\pm 4.0}$	$\mathbf{100.0}_{\pm 0.0}$	$88.0_{\pm 6.9}$	$38.7_{\pm 6.1}$	$93.3_{\pm 4.6}$	$96.0_{\pm 0.0}$	$54.7_{\pm 6.1}$
BridgeVLA++*	$79.7_{\pm 1.1}$	$\mathbf{100.0}_{\pm 0.0}$	$44.0_{\pm 6.9}$	$34.7_{\pm 10.1}$	$\mathbf{100.0}_{\pm 0.0}$	$96.0_{\pm 4.0}$	$\mathbf{56.0}_{\pm 4.0}$	$\mathbf{97.3}_{\pm 4.6}$	$\mathbf{100.0}_{\pm 0.0}$	$81.3_{\pm 8.3}$
LexiconVLA (BridgeVLA++)	$77.0_{\pm 1.7}$	$92.0_{\pm 0.0}$	$58.7_{\pm 4.6}$	$36.0_{\pm 8.0}$	$98.7_{\pm 2.3}$	$\mathbf{97.3}_{\pm 2.3}$	$44.0_{\pm 4.0}$	$\mathbf{97.3}_{\pm 2.3}$	$\mathbf{100.0}_{\pm 0.0}$	$76.0_{\pm 0.0}$
BridgeVLA	$\mathbf{89.8}_{\pm 1.0}$	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{98.7}_{\pm 2.3}$	$\mathbf{100.0}_{\pm 0.0}$	$96.0_{\pm 0.0}$	$45.3_{\pm 4.6}$	$90.7_{\pm 6.1}$	$\mathbf{100.0}_{\pm 0.0}$	$84.0_{\pm 4.0}$
LexiconVLA (BridgeVLA)	$87.5_{\pm 0.9}$	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{100.0}_{\pm 0.0}$	$85.3_{\pm 6.1}$	$\mathbf{100.0}_{\pm 0.0}$	$96.0_{\pm 0.0}$	$54.7_{\pm 2.3}$	$93.3_{\pm 8.3}$	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{85.3}_{\pm 4.6}$

Method	Put in Drawer	Put in Safe	Screw Bulb	Slide Block	Sort Shape	Stack Blocks	Stack Cups	Sweep to Dustpan	Turn Tap
Act3D	$76.0_{\pm 4.0}$	$\mathbf{100.0}_{\pm 0.0}$	$41.3_{\pm 12.9}$	$96.0_{\pm 0.0}$	$45.3_{\pm 2.3}$	$5.3_{\pm 6.1}$	$12.0_{\pm 6.9}$	$93.3_{\pm 6.1}$	$94.7_{\pm 6.1}$
3D Diffuser Actor	$88.0_{\pm 0.0}$	$94.7_{\pm 2.3}$	$82.7_{\pm 2.3}$	$\mathbf{100.0}_{\pm 0.0}$	$12.0_{\pm 8.0}$	$\mathbf{84.0}_{\pm 4.0}$	$40.0_{\pm 4.0}$	$72.0_{\pm 4.0}$	$\mathbf{98.7}_{\pm 2.3}$
SAM2Act	$97.3_{\pm 4.6}$	$98.7_{\pm 2.3}$	$94.7_{\pm 2.3}$	$76.0_{\pm 4.0}$	$42.7_{\pm 8.3}$	$73.3_{\pm 4.6}$	$\mathbf{77.3}_{\pm 10.1}$	$98.7_{\pm 2.3}$	$93.3_{\pm 2.3}$
PerAct	$21.3_{\pm 2.3}$	$34.7_{\pm 10.1}$	$8.0_{\pm 0.0}$	$57.3_{\pm 10.1}$	$0.0_{\pm 0.0}$	$17.3_{\pm 2.3}$	$5.3_{\pm 2.3}$	$58.7_{\pm 6.1}$	$86.7_{\pm 6.1}$
LexiconVLA (PerAct)	$58.7_{\pm 8.3}$	$40.0_{\pm 6.9}$	$22.7_{\pm 8.3}$	$61.3_{\pm 6.1}$	$1.3_{\pm 2.3}$	$17.3_{\pm 6.1}$	$1.3_{\pm 2.3}$	$62.7_{\pm 2.3}$	$90.7_{\pm 6.1}$
RVT	$86.7_{\pm 11.5}$	$65.3_{\pm 4.6}$	$48.0_{\pm 6.9}$	$78.7_{\pm 6.1}$	$33.3_{\pm 4.6}$	$37.3_{\pm 9.2}$	$9.3_{\pm 4.6}$	$58.7_{\pm 6.1}$	$92.0_{\pm 0.0}$
LexiconVLA (RVT)	$85.3_{\pm 2.3}$	$64.0_{\pm 4.0}$	$49.3_{\pm 6.1}$	$85.3_{\pm 4.6}$	$33.3_{\pm 6.1}$	$33.3_{\pm 2.3}$	$22.7_{\pm 4.6}$	$74.7_{\pm 6.1}$	$89.3_{\pm 2.3}$
RVT-2	$93.3_{\pm 6.1}$	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{97.3}_{\pm 2.3}$	$70.7_{\pm 6.1}$	$46.7_{\pm 4.6}$	$76.0_{\pm 4.0}$	$61.3_{\pm 4.6}$	$96.0_{\pm 0.0}$	$94.7_{\pm 2.3}$
LexiconVLA (RVT-2)	$\mathbf{98.7}_{\pm 2.3}$	$98.7_{\pm 2.3}$	$94.7_{\pm 4.6}$	$72.0_{\pm 6.9}$	$42.7_{\pm 4.6}$	$73.3_{\pm 6.1}$	$72.0_{\pm 4.0}$	$\mathbf{100.0}_{\pm 0.0}$	$88.0_{\pm 4.0}$
BridgeVLA++*	$93.3_{\pm 4.6}$	$96.0_{\pm 4.0}$	$85.3_{\pm 2.3}$	$94.7_{\pm 6.1}$	$58.7_{\pm 4.6}$	$68.0_{\pm 6.9}$	$37.3_{\pm 11.5}$	$98.7_{\pm 2.3}$	$93.3_{\pm 2.3}$
LexiconVLA (BridgeVLA++)	$96.0_{\pm 0.0}$	$97.3_{\pm 2.3}$	$80.0_{\pm 4.0}$	$\mathbf{100.0}_{\pm 0.0}$	$61.3_{\pm 6.1}$	$70.7_{\pm 6.1}$	$12.0_{\pm 6.9}$	$76.0_{\pm 4.0}$	$93.3_{\pm 4.6}$
BridgeVLA	$97.3_{\pm 2.3}$	$98.7_{\pm 2.3}$	$\mathbf{97.3}_{\pm 2.3}$	$97.3_{\pm 4.6}$	$\mathbf{70.7}_{\pm 2.3}$	$77.3_{\pm 8.3}$	$70.7_{\pm 8.3}$	$\mathbf{100.0}_{\pm 0.0}$	$92.0_{\pm 0.0}$
LexiconVLA (BridgeVLA)	$96.0_{\pm 6.9}$	$\mathbf{100.0}_{\pm 0.0}$	$92.0_{\pm 0.0}$	$92.0_{\pm 0.0}$	$57.3_{\pm 8.3}$	$76.0_{\pm 6.9}$	$56.0_{\pm 6.9}$	$\mathbf{100.0}_{\pm 0.0}$	$90.7_{\pm 4.6}$