ACTATLAS: LEARNING ADAPTIVE EXECUTION ATLASES FOR LONG-HORIZON ROBOTIC MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon robot manipulation requires a policy to perform a sequence of connected operations. Diffusion policies can replan from new observations, but they often fail when two states look similar yet belong to different task stages and therefore need different actions. We argue that a policy should know not only which local pattern the current state belongs to, but also how this pattern is connected to the other patterns of the task. To this end, we propose **ActAtlas**, which learns a data-adaptive graph of task stages from demonstrations and uses it to condition a diffusion policy, so that the robot generates actions with knowledge of both the current stage and how it connects to the rest of the task. Specifically, in the first stage, ActAtlas learns a set of *execution anchors* from demonstrations with self-supervision. Each anchor is a prototype shared by similar states across demonstrations, and the number of anchors is adjusted during training by growing, merging, and splitting. Transitions between anchors form a directed *execution-flow graph*. In the second stage, a graph attention network combines the current anchor with related anchors into a fixed-size context vector, which conditions a single diffusion policy. Because the graph weights are further trained end-to-end with the action denoising loss, these two stages allow ActAtlas to recognize the current stage of the task and use the learned stage-to-stage relations to choose the correct next action. Across four simulation benchmarks, ActAtlas improves the average success rate by 12.4% over the strongest baseline, with the largest LIBERO gain (16.2%) on the long-horizon suite. On four real-world tasks, it improves the average success rate by 25.0 %. More broadly, our results suggest that learning how task stages connect to each other, rather than only what the current state looks like, is a simple and effective way to make generative robot policies more reliable on long, multi-step tasks.

1 INTRODUCTION

Many useful robot tasks, such as assembly and object arrangement, are long-horizon: the robot must complete several operations in the right order, and each operation needs precise motion (Kroemer et al., 2021). Generative imitation learning has made strong progress on this problem. For example, action chunking reduces the number of decisions by predicting several actions at once (Zhao et al., 2023), and diffusion models generate action distributions through iterative denoising (Chi et al., 2025). These policies usually run in a receding-horizon loop: they predict an action chunk, execute its first few actions, and then predict again from the new observation.

This loop has a known weakness in long-horizon tasks. Making or breaking contact changes the local dynamics (Kroemer et al., 2021; Sidhik et al., 2021), so approaching a peg, touching it, and correcting the alignment need different controls. In the meanwhile, the same local motion can appear in different parts of a task (Zhu et al., 2022), and visually similar observations should require different next actions at different stages (Hu et al., 2026). A policy that is conditioned only on the current observation may therefore choose the wrong continuation. To avoid this, the policy needs two essential pieces of information: **what** local pattern the current state belongs to, and **where** this pattern sits in the flow of the task.

Prior remedies are in several forms. (i) Hierarchical policies use subgoals or skills (Ma et al., 2024; Liang et al., 2024). (ii) Skill- and concept-learning methods group recurring behavior into discrete

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.

or continuous representations (Xu et al., 2023; Liu et al., 2024; Zhou et al., 2025; Liu et al., 2026; Huang et al., 2026). (iii) History-conditioned policies add information about past motion (Zheng et al., 2025; Hu et al., 2026). However, two questions remain open. **(i) Granularity:** Which demonstration states should share one representation? Most concept methods fix the number of concepts in advance; too few concepts merge different stages, and too many split the data into small pieces. **(ii) Relations:** how should the connections between these representations be used when generating actions? These methods usually condition the policy on one concept at a time and do not explicitly model how concepts follow each other.

To answer both above questions, we propose ActAtlas, learning a data-adaptive stage graph from demonstrations to condition the generative model, allowing the robot to act with awareness of the current stage and its links to the full task. (Figure 1). Our key idea is: **an execution pattern should be described by both its local state content and its transitions to other patterns.** ActAtlas represents recurring patterns with *execution anchors* and connects them with a directed *execution-flow graph*. Together, the anchors and the graph form an *execution atlas*.

In detail, ActAtlas is trained in two stages. In **Stage 1 (Anchor-graph (atlas) learning)**, a state encoder assigns demonstration states to a shared set of anchor prototypes, similar to vector quantization (Van Den Oord et al., 2017). The encoder is trained with two self-supervised losses: a state reconstruction loss and a contrastive loss that matches each anchor to the observation and action at the same time step. To answer the question regarding granularity, the anchor set can be adaptively regulated by growing when states are poorly covered, merging when too similar, and splitting when assigned states are too spread out. In **Stage 2 (Atlas-conditioned policy learning)**, the current state selects an anchor. A graph attention network (Veličković et al., 2017) aggregates features from related anchors, and the result is fused with the selected anchor into a fixed-size context vector. This vector conditions one shared diffusion policy through FiLM layers (Perez et al., 2018). The initial graph weights start from the transition counts in the demonstrations and are then trained together with the policy by the denoising loss, so the graph keeps the relations that are most useful for predicting actions, which answers the above-mentioned question 2.

We evaluate ActAtlas on four simulation benchmarks (LIBERO (Liu et al., 2023), RoboTwin 2.0 (Chen et al., 2025), RoboMimic (Mandlekar et al., 2021), and D3IL (Jia et al., 2024)) and four real-world tasks on an SO-101 robot arm. ActAtlas achieves the best result on every benchmark, improving the average success rate over the strongest baseline by 12.4% in simulation, with the largest LIBERO gain of 16.2% on the long-horizon suite, and by 25.0% in the real world. Ablations further show that adaptive granularity, graph relations, and FiLM conditioning all contribute to the gain. Our contributions are:

- We introduce **ActAtlas**, an execution-atlas representation for long-horizon manipulation. It addresses the confusion between similar-looking states at different task stages by describing each state with both a shared prototype (anchor) and the directed transitions between anchors. The number of anchors is learned from the demonstrations through growth, merging, and splitting, so the granularity fits the task instead of being fixed in advance.
- We propose a **graph-conditioned diffusion policy** that brings this task-stage information into closed-loop action generation. Graph attention over the atlas produces a fixed-size context that conditions a single shared diffusion policy. The graph weights are initialized from demonstration transitions and then trained end-to-end with the action denoising loss, so the policy learns which stage relations help it choose the correct continuation.
- Outperformance demonstrates that ActAtlas has stronger long-horizon capability than baselines in multiple tasks both in simulation and the real world. Ablations further isolate the contribution of each component.

2 Related Work

Task-aware Representation in Imitation Learning. Several methods find recurring behavior in demonstrations without labels. They segment trajectories in time (Kipf et al., 2019), learn from cross-view correspondence (Sermanet et al., 2018), or discover concepts through prediction and reconstruction (Liu et al., 2024; Zhou et al., 2025). Other methods learn skills from video or multimodal data (Xu et al., 2023; Liu et al., 2026), or grow a skill library as new tasks arrive (Wan et al.,

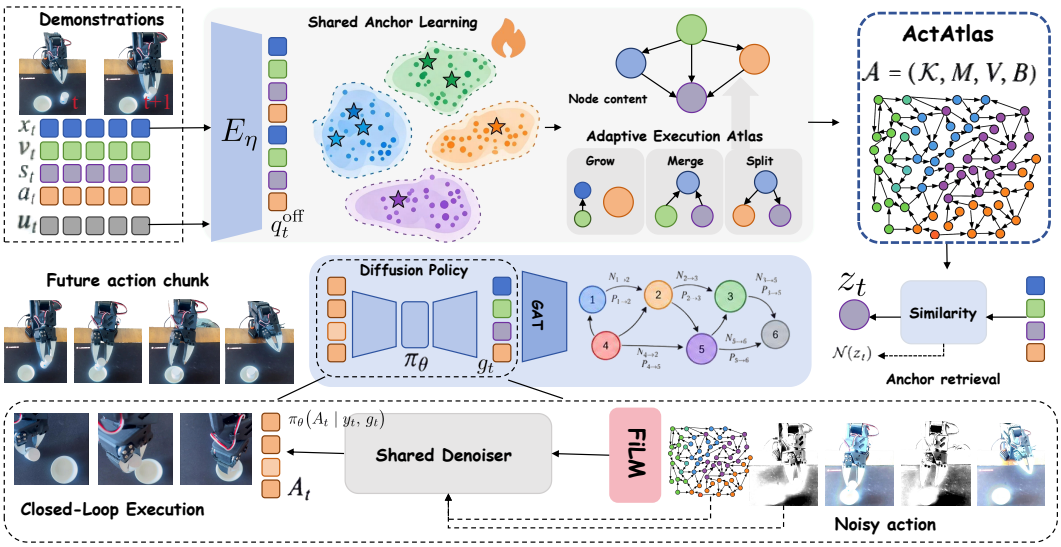


Figure 1: **Overview of ActAtlas. Top:** Demonstrations are encoded into shared execution anchors. State and observation–action supervision learn anchor content, while grow, merge, and split adapt the active set; observed transitions initialize directed relation scores in the execution atlas $\mathcal{A} = (\mathcal{K}, M, V, B)$. **Middle:** Anchor retrieval produces z_t, after which graph attention aggregates related-anchor features and a fusion module forms the fixed-width context g_t for the diffusion policy, which predicts a future action chunk A_t. **Bottom:** The expanded denoising path shows g_t and sensory context conditioning the shared denoiser through FiLM. After a short action prefix is executed, updated observations and preceding execution support the next retrieval.

2024). Like these methods, ActAtlas learns shared representations without labels. Unlike them, it adapts the number of anchors to the data and uses each anchor as a node in a graph of transitions.

Structured conditioning for long-horizon manipulation. Several methods add structure to long-horizon policies. Bottom-up skill discovery builds structure from unsegmented demonstrations (Zhu et al., 2022), Neural Task Graphs infer a task graph from a single video (Huang et al., 2019), and trace-based policies encode past motion as visual traces (Zheng et al., 2025; Hu et al., 2026). For Diffusion Policy (Chi et al., 2025), hierarchical variants guide denoising with target poses (Ma et al., 2024), Crossway Diffusion adds a self-supervised state reconstruction loss (Li et al., 2024), and CoRDE routes inputs to diffusion experts with concept priors (Huang et al., 2026). In contrast, ActAtlas builds a graph over learned anchors and turns it into a fixed-size condition with local anchor content and graph context for one shared diffusion policy trained with the action loss.

3 Method

ActAtlas learns adaptive shared anchors for long-horizon manipulation, then trains one graph-conditioned diffusion policy as an action generator (Figure 1; details in Appendix 6).

3.1 Problem Statement

Demonstrations $\mathcal{D} = \{\xi^{(n)}\}_{n=1}^{N}$ contain trajectories $\xi^{(n)} = \{(x_t, v_t, s_t, a_t)\}_{t=1}^{T_n}$, where $x_t \in \mathbb{R}^d$ supports anchor retrieval, v_t is an observation feature, s_t is proprioception, and $a_t \in \mathbb{R}^{d_a}$ is the action. Given $y_t = [v_t; \mathrm{vec}(s_{t-L+1:t})]$, the policy predicts $A_t = a_{t:t+H-1} \in \mathbb{R}^{H \times d_a}$:

$$z_t = Q_\eta(x_t, z_{t^-}; \mathcal{A}, P^{\mathrm{tr}}), \qquad g_t = \mathcal{R}_\phi(\mathcal{A}, z_t), \qquad A_t \sim \pi_\theta(\cdot \mid y_t, g_t). \tag{1}$$

The atlas $\mathcal{A} = (\mathcal{K}, M, V, B)$ stores active indices, retrieval keys $M = \{\mu_k\}$, shared values $V = \{\nu_k\}$, and directed scores B. Each anchor is a cross-demonstration reference and graph node. The active count $K = |\mathcal{K}|$ adapts during Stage 1, while $g_t \in \mathbb{R}^{d_g}$ has fixed width. Keys support matching

and values supply shared content. The previous query t^- supplies execution history through P^{tr}; y_t retains instance-specific information.

3.2 Stage 1: Anchor-graph (atlas) Learning

Shared content and self-supervision. Offline queries $q_t^{\mathrm{off}} = \mathrm{Norm}(E_\eta(x_t) + \alpha_u E_u(u_t))$ combine state features and normalized demonstration time $u_t \in [0, 1]$ to organize shared anchors across demonstrations. We distinguish them from the time-free online queries $q_t^{\mathrm{on}} = \mathrm{Norm}(E_\eta(x_t))$ used in Stage 2 and deployment; the transition prior is constructed with the same online retriever (Appendix 6.1; Appendix 6.9). Unit-norm queries and prototypes give

$$r_t(k) = \frac{\exp((q_t^{\mathrm{off}})^\top \mu_k/\tau_q)}{\sum_{j\in\mathcal{K}} \exp((q_t^{\mathrm{off}})^\top \mu_j/\tau_q)}, \qquad \widetilde{z}_t = \arg\max_{k\in\mathcal{K}} (q_t^{\mathrm{off}})^\top \mu_k. \tag{2}$$

Within each window, set $z_{T_w} = \widetilde{z}_{T_w}$ and apply $z_t = \min\{\widetilde{z}_t, z_{t+1}\}$ backwards in stored slot order; constant runs define segments (Sec. 6.2). The soft assignments remain unchanged. A straight-through value (Van Den Oord et al., 2017) selects shared content while passing encoder gradients through soft weights:

$$\zeta_t = \zeta_t^{\mathrm{soft}} + \mathrm{sg}\big(\nu_{z_t} - \zeta_t^{\mathrm{soft}}\big), \qquad \zeta_t^{\mathrm{soft}} = \sum_{k\in\mathcal{K}} r_t(k)\nu_k. \tag{3}$$

Here sg stops gradients: the forward value is ν_{z_t}, and the backward path differentiates the soft combination. Shared content is learned from state preservation and synchronized sensory–motor associations. For m samples,

$$\mathcal{L}_{\mathrm{state}} = \frac{1}{md}\sum_{i=1}^{m}\Big[\|\zeta_i - \mathrm{sg}(x_i)\|_2^2 + \lambda_q \|\,\mathrm{sg}(\zeta_i) - x_i\|_2^2\Big]. \tag{4}$$

With fixed input features, the first term trains the encoder; values follow aggregation. Shared error separates into within-group variation and prototype mismatch (Sec. 6.4). To associate content with synchronized observations and actions, define $e_i^c = \mathrm{Norm}(f_c(\zeta_i))$, $e_i^v = \mathrm{Norm}(f_v(v_i))$, $e_i^a = \mathrm{Norm}(f_a(a_i))$, and logits $S_{ij}^b = \gamma_b (e_i^c)^\top e_j^b$ with learned $\gamma_b > 0$. Symmetric matching gives

$$\begin{aligned} \mathcal{L}_b &= -\frac{1}{2m}\sum_{i=1}^{m}\left[\log\frac{\exp(S_{ii}^b)}{\sum_j \exp(S_{ij}^b)} + \log\frac{\exp(S_{ii}^b)}{\sum_j \exp(S_{ji}^b)}\right], \qquad b \in \{v, a\}, \\ \mathcal{L}_{\mathrm{anchor}} &= \mathcal{L}_{\mathrm{state}} + \lambda_{\mathrm{align}}(\mathcal{L}_v + \mathcal{L}_a). \end{aligned} \tag{5}$$

Same-timestep pairs provide positive correspondences; both matching losses train the projections and, through soft assignments, the state encoder. For each occupied anchor, $\bar{q}_k$ is the mean of its assigned offline queries. Keys and values consolidate these queries without gradients, using $0 < \rho < 1$:

$$\mu_k \leftarrow \mathrm{Norm}\left(\rho\mu_k + (1-\rho)\bar{q}_k\right), \qquad \nu_k \leftarrow \mathrm{Norm}\left(\rho\nu_k + (1-\rho)\bar{q}_k\right). \tag{6}$$

For fixed queries and assignments with nonzero update directions, this improves or preserves query-space fit (Proposition 6.2); proofs and surrogate gradients are in Sec. 6.4 and Sec. 6.3.

Adapting the number of anchors. Starting from K_0 active entries within $K_{\max}$ slots, three non-gradient operations adjust sharing granularity:

- **Grow.** If $\max_k \chi_{\mathcal{W}}^\top \mu_k < \tau_g$, initialize its key and value in a free slot at $\chi_{\mathcal{W}} = \mathrm{Norm}(|\mathcal{W}|^{-1}\sum_{t\in\mathcal{W}} x_t)$.
- **Merge.** When the most similar active pair satisfies $\mu_i^\top \mu_j > \tau_m$, use $\mathrm{Norm}(w_i\mu_i + w_j\mu_j)$ with positive usage weights; values follow the same rule.
- **Split.** If the largest running per-coordinate query dispersion D_k exceeds τ_s and a slot is free, initialize both keys and values at $\mathrm{Norm}(\mu_k \pm \epsilon_s r)$, with a random unit vector r.

These operations extend coverage, consolidate overlap, and refine dispersed groups; bounds and schedules are in Sec. 6.5 and Sec. 7.4.

3.3 EXECUTION-FLOW GRAPH

Assignments jointly organize shared content and relations. Counts ΔN_{ij} record adjacent assignments within demonstration windows, initializing directed scores:

$$B_{ij} \leftarrow B_{ij} + \eta_B \, \Delta N_{ij} \quad (i \neq j), \qquad W_{ij} = \frac{\exp(B_{ij})}{\sum_{\ell \in \mathcal{K}} \exp(B_{i\ell})}. \tag{7}$$

Stage 2 further trains B through the action objective, turning the transition-initialized W into action-supervised relation weights for graph aggregation. Equation (7) uses inter-anchor switches to initialize these relations; current-anchor information is retained by the selected local value and the self term in Eq. (10). After training, a frozen time-free retriever re-encodes the training demonstrations at intervals of H_e executed actions. Counts $C_{ij}^{(H_e)}$ include both stays and switches and define:

$$P_{ij}^{\mathrm{tr}} = \frac{C_{ij}^{(H_e)} + \alpha_{\mathrm{tr}}}{\sum_{\ell \in \mathcal{K}} C_{i\ell}^{(H_e)} + K\alpha_{\mathrm{tr}}}, \qquad \alpha_{\mathrm{tr}} > 0. \tag{8}$$

Thus W supplies action-supervised relational context to graph aggregation, while P^{tr} supplies preceding-execution context to closed-loop retrieval over the same anchor space (Sec. 6.9).

3.4 STAGE 2: ATLAS-CONDITIONED POLICY LEARNING

Retrieval and relational context. Current-state queries are $q_t^{\mathrm{on}} = \mathrm{Norm}(E_\eta(x_t))$. Let $\bar{p}_t(k) = P^{\mathrm{tr}}_{z_{t^-},k}$ after an executed prefix and $\bar{p}_t(k) = 1/K$ at reset. Selection precedes graph encoding:

$$z_t = \arg\max_{k \in \mathcal{K}} \left[(q_t^{\mathrm{on}})^\top \mu_k + \lambda_{\mathrm{tr}} \log \bar{p}_t(k) \right], \qquad \lambda_{\mathrm{tr}} \geq 0. \tag{9}$$

Here λ_{tr} weights preceding-execution evidence relative to current-state matching: $\lambda_{\mathrm{tr}} = 0$ gives state-matched retrieval, while $\lambda_{\mathrm{tr}} > 0$ combines $(q_t^{\mathrm{on}})^\top \mu_k$ with the transition distribution induced by the previously selected anchor. Positive smoothing keeps all candidates available, and the uniform prior at episode reset leaves the first selection state-based. Stage 2 training uses $\lambda_{\mathrm{tr}} = 0$. Active values initialize multi-head graph attention (Veličković et al., 2017). One head computes

$$\alpha_{ij} = \mathrm{softmax}_j \left[\left(W_{ij} + \epsilon_{\mathrm{self}} \mathbf{1}_{\{i=j\}} \right) \frac{(U_Q h_i)^\top (U_K h_j)}{\sqrt{d_h}} \right], \qquad m_i = \sum_j \alpha_{ij} U_V h_j, \tag{10}$$

Shared projections and nodewise updates use the same W across layers. The selected local value and related-anchor features form

$$\widetilde{V} = \mathrm{GraphEncoder}_\phi(V, W), \qquad g_t = F_\phi\big([\nu_{z_t}; \widetilde{\nu}_{z_t}]\big) \in \mathbb{R}^{d_g}. \tag{11}$$

This fixed-width condition supports one shared generator. For a fixed atlas, g_t depends on z_t, while y_t carries sensory history. Detailed layer details and permutation equivariance are in Sec. 6.6.

Action-supervised conditioning and execution. One denoiser uses $c_t = [y_t; g_t]$ and FiLM (Perez et al., 2018), $F' = (1 + \gamma(d_{\tau,t})) \odot F + \beta(d_{\tau,t})$, with $d_{\tau,t} = [e_\tau; y_t; g_t]$. Noise prediction (Ho et al., 2020) minimizes

$$X_\tau = \sqrt{\bar{\alpha}_\tau} A_t + \sqrt{1 - \bar{\alpha}_\tau}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \quad \mathcal{L}_{\mathrm{diff}} = \mathbb{E}\left[\frac{1}{H d_a} \left\| \epsilon_\theta(X_\tau, \tau, c_t) - \epsilon \right\|_F^2 \right]. \tag{12}$$

Here $\bar{\alpha}_\tau$ is the cumulative signal coefficient; the expectation covers demonstration chunks, diffusion steps, and Gaussian noise. Since the relation scores B parameterize the graph aggregation weights, $\mathcal{L}_{\mathrm{diff}}$ backpropagates through graph attention to train the execution-flow relations for action prediction; the full gradients are derived in Sec. 6.7. An auxiliary classifier predicts the retrieved index:

$$\mathcal{L}_{\mathrm{policy}} = \mathcal{L}_{\mathrm{diff}} + \lambda_{\mathrm{aux}} \mathcal{L}_{\mathrm{aux}}, \qquad \mathcal{L}_{\mathrm{aux}} = -\mathbb{E} \log R_\psi(z_t \mid s_{t-L+1:t}, g_t). \tag{13}$$

Stage 2 fixes E_η and $\mathcal{K}$, continues prototype aggregation, and trains B, ϕ, θ, ψ (Sec. 6.8). Evaluation freezes parameters, prototypes, and P^{tr}. Each query executes the first H_e predicted actions, then updates its preceding index; episode reset clears it, and denoising holds it fixed. Prior strength is fixed before testing; sampling and both retrieval settings are detailed in Sec. 6.9.

Table 1: **Success Rates (SR) in simulation.** Δ_{DP}: gain over DP. Macro-Avg.: equal weight on the four benchmarks. The last row is ours minus the best baseline. **Bold**: best; underline: second best.

Method	LIBERO		RoboTwin 2.0		RoboMimic		D3IL		Macro-Avg.	
	SR(%)	Δ_{DP}	SR(%)	Δ_{DP}	SR(%)	Δ_{DP}	SR(%)	Δ_{DP}	SR(%)	Δ_{DP}
DP	76.1	–	29.6	–	59.8	–	63.3	–	57.2	–
HiMaCon+DP	83.9	+7.8	46.4	+16.8	61.4	+1.6	69.8	+6.5	65.4	+8.2
AutoCGP+DP	86.0	+9.9	47.8	+18.2	66.1	+6.3	74.4	+11.1	68.6	+11.4
TF-DP	83.1	+7.0	39.8	+10.2	72.1	+12.3	79.5	+16.2	68.6	+11.4
ActAtlas (ours)	**91.7**	**+15.6**	**62.8**	**+33.2**	**83.7**	**+23.9**	**86.0**	**+22.7**	**81.0**	**+23.8**
Δ *over best*	+5.7		+15.0		+11.6		+6.5		+12.4	

4 EXPERIMENTS

We answer four questions. **Q1:** Does ActAtlas improve task success, especially on long-horizon tasks (Section 4.1)? **Q2:** Which components matter, and is the gain explained by a longer history (Section 4.2)? **Q3:** What do the learned anchors capture (Section 4.3)? **Q4:** Does ActAtlas remain effective on a real robot (Section 4.4)?

Benchmarks. We use four simulation benchmarks: LIBERO (Liu et al., 2023), RoboTwin 2.0 (Chen et al., 2025), RoboMimic (Mandlekar et al., 2021), and D3IL (Jia et al., 2024). They differ in task composition, bimanual interaction, demonstration quality, and behavior diversity. For LIBERO, we report all four suites: L-Spatial, L-Object, L-Goal, and L-Long (LIBERO-10).

Baselines. We compare with Diffusion Policy (DP) (Chi et al., 2025), two concept-based methods built on DP, HiMaCon+DP (Liu et al., 2026) and AutoCGP+DP (Zhou et al., 2025), and the trace-conditioned TF-DP (Hu et al., 2026). On LIBERO and the real robot, we also include ACT (Zhao et al., 2023). We additionally compare with the vision-language-action models π_0 (Black et al., 2024) and $\pi_{0.5}$ (Intelligence et al., 2025).

Metrics. We report full-task success rates and equal-weight benchmark averages. The LIBERO score equally weights its four suites, while the Macro-Avg. in Table 1 equally weights the four benchmark-level means. All reported gains and gaps are expressed in %. Detailed aggregation rules are in Appendix 8.1.

Evaluation protocol. We use a two-level evaluation protocol. Table 1 provides the cross-benchmark comparison: DP-based methods use three independent training seeds and report benchmark-level means. Table 2 provides a dedicated LIBERO repeated-run evaluation: ACT and DP-based methods use four seeds with 50 initial states per task and seed, while π_0 and $\pi_{0.5}$ are single-seed reference estimates; repeated-seed entries report mean and standard error (SE). All headline ActAtlas results in Tables 1–2 and Table 3(a) use state-matched retrieval ($\lambda_{\mathrm{tr}} = 0$); Table 3(b) separately studies transition-aware retrieval. Exact aggregation and SE computation are in Appendix 8.1 and Appendix 8.5.

4.1 MAIN RESULTS IN SIMULATION

Overall performance. Table 1 shows that ActAtlas achieves the best success rate on all four benchmarks. Its macro-average is 81.0%, compared with 68.6% for the strongest baselines (AutoCGP+DP and TF-DP), a gain of 12.4%. The largest gains are on RoboTwin 2.0 (+15.0%) and RoboMimic (+11.6%). ActAtlas also outperforms both concept-based methods on every benchmark, which suggests that adaptive anchors with graph context are a stronger condition than a single fixed-size concept inventory. The consistent gains over all baselines support learning adaptive execution together with their relations for diffusion-based imitation control. The VLA entries in Table 2 provide broader performance across different model scales and pretraining settings.

Table 2: **Success rates (%) on the four LIBERO suites.** ACT and DP-based methods report mean ± SE across four training seeds, with 50 initial states per task and seed. Mean equally weights the four suites and reports a point estimate. Bold, underline, and Δ_{pol} compare these methods only. †: single-seed VLA reference point estimates. L-Long is LIBERO-10.

Suite	ACT	DP	$\pi_{0.5}$ (4.1B)†	π_0 (3.2B)†	HiMaCon +DP	AutoCGP +DP	ActAtlas	Δ_{pol}
L-Spatial	76.3 ±0.4	78.5 ±0.7	97.0	95.7	82.6 ±0.8	<u>88.7</u> ±0.4	**89.2** ±1.2	+0.5
L-Object	78.8 ±0.5	87.6 ±1.1	98.6	98.8	89.7 ±0.4	<u>90.8</u> ±0.4	**94.1** ±0.8	+3.3
L-Goal	67.2 ±1.1	73.5 ±0.2	98.0	94.2	91.1 ±0.4	<u>94.5</u> ±0.4	**94.8** ±0.3	+0.3
L-Long	52.0 ±0.9	64.8 ±0.4	95.2	84.9	<u>72.4</u> ±0.7	69.8 ±0.8	**88.6** ±0.3	+16.2
Mean	68.6	76.1	97.2	93.4	84.0	<u>86.0</u>	**91.7**	+5.7

Table 3: **Ablation study.** Full-task success rates (%) with SEs over different seeds; **(a)** Results on three benchmarks where Gap (%) indicates full ActAtlas minus each variant. **(b)** Results on LIBERO-Long. Bold denotes the best performance in each setting.

(a) Component ablations across benchmarks

Factor	Variant	RoboMimic	RoboTwin 2.0	D3IL	Avg.	Gap
–	**Full ActAtlas**	**83.7** ±0.5	**62.8** ±0.7	**86.0** ±0.7	**77.5**	–
Granularity	FixedAtlas-K	77.0 ±0.4	52.1 ±0.1	73.5 ±0.5	67.5	10.0
Relations	Anchor-only	62.1 ±0.4	37.2 ±0.8	61.4 ±0.4	53.6	23.9
	Shuffled Graph	64.5 ±0.5	42.1 ±0.9	69.5 ±0.2	58.7	18.8
Conditioning	Concat-g	74.5 ±0.4	44.2 ±0.3	72.5 ±1.1	63.7	13.8
History	LongHist-DP	63.3 ±0.7	33.5 ±0.4	66.3 ±0.9	54.4	23.1
	LongHist-AutoCGP	74.8 ±1.2	51.2 ±0.3	77.3 ±0.6	67.8	9.7

(b) Conditioning and retrieval on LIBERO-Long

Conditioning	State-matched ($\lambda_{\text{tr}} = 0$)	Transition-aware ($\lambda_{\text{tr}} > 0$)
Per-anchor embedding	76.5 ±0.7	82.3 ±0.4
Graph context	**88.6** ±0.3	**91.3** ±0.2

Long-horizon tasks. Table 2 breaks down the LIBERO results. Especially on long-horizon tasks (L-long), which require several connected operations to complete, ActAtlas reaches 88.6%, 16.2% above HiMaCon+DP, 18.8% above AutoCGP+DP, and 23.8% above DP. Details in Appendix 8.4.

4.2 Ablation Study

We ablate adaptive granularity, graph relations, conditioning, and observation history on RoboMimic, RoboTwin 2.0, and D3IL (Table 3(a)); full variant definitions are in Appendix 8.2. Table 3(b) separately controls conditioning and retrieval on LIBERO-Long, with $\lambda_{\text{tr}} = 0$ for state-matched retrieval and $\lambda_{\text{tr}} > 0$ for transition-aware retrieval.

Graph relations matter most. Removing graph aggregation (Anchor-only) lowers the average success rate from 77.5% to 53.6%, showing the value of related-anchor context beyond the selected anchor. Shuffled Graph retains the graph encoder and relation weights but breaks anchor–relation correspondence, reducing the average to 58.7%. Together, these results show that the gain comes from combining anchor-local content with structured relational context.

Adaptive granularity and FiLM help. Fixing the number of anchors lowers the average by 10.0%, which supports that learning the granularity from data helps the improvement. Replacing FiLM with concatenation lowers it by 13.8%, supporting modulation of intermediate denoising features rather than input concatenation of the atlas context.

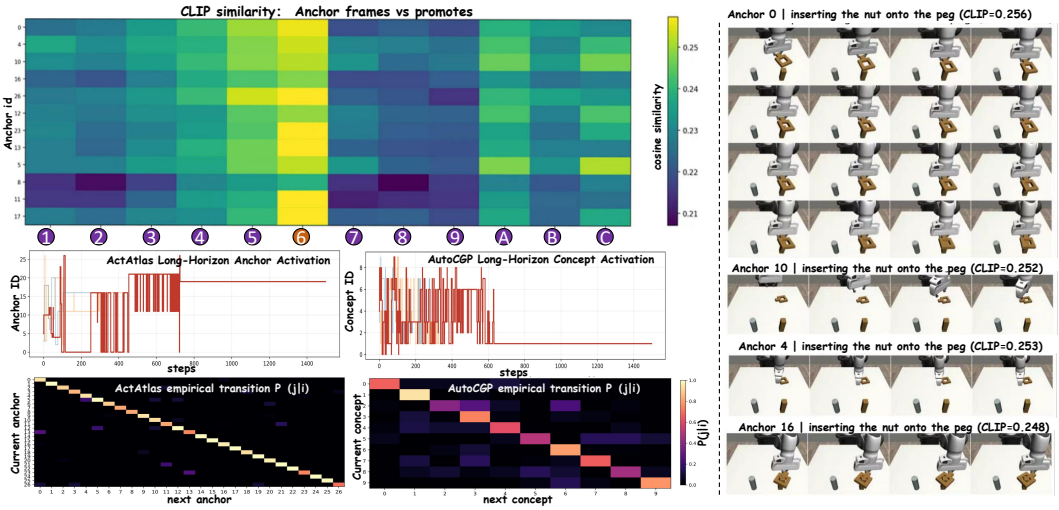


Figure 2: **Anchor usage and content on NutAssemblySquare. Top left:** CLIP similarity between anchor images and 12 text prompts. **Right:** example image groups for several anchors from a successful rollout. **Middle left:** anchor index over time in successful (faint) and failed (dark red) rollouts. **Bottom left:** row-normalized anchor transitions observed in closed-loop rollouts, showing temporal persistence and switching structure. AutoCGP denotes an AutoCGP-style model with a fixed codebook. Indices denote categories and imply no order.

A longer history is not enough. A longer history improves DP from 50.9% to 54.4% and AutoCGP+DP from 62.8% to 67.8%, but ActAtlas remains 23.1% and 9.7% higher, respectively. Thus, longer history alone does not explain the execution-atlas gains.

Controlled conditioning and retrieval. Table 3(b) isolates conditioning and retrieval while keeping the same Stage 1 atlas and matched policy and training settings; the full protocol is in Appendix 8.3. With state-matched retrieval, graph context improves success from $76.5 \pm 0.7\%$ to $88.6 \pm 0.3\%$, isolating the benefit of relational conditioning. On the same graph-conditioned policy, transition-aware retrieval further raises performance to $91.3 \pm 0.2\%$. Thus graph context strengthens action conditioning, while the execution prior provides an additional closed-loop retrieval gain.

4.3 What Do the Anchors Learn?

We analyze the anchors on RoboMimic NutAssemblySquare (Figure 2), complementing the control ablations with a direct view of anchor organization during execution.

Temporal structure. We record the selected anchor during rollouts and compute persistence, transition density, and row entropy (definitions in Appendix 9.2). Compared with an AutoCGP-style fixed codebook, ActAtlas stays in the same anchor more often (0.934 vs. 0.738), uses fewer observed transitions (0.069 vs. 0.640), and has lower row entropy (0.404 vs. 1.082). Together, the activation trajectories and transition statistics show that adaptive anchors organize closed-loop behavior into more persistent local execution patterns with more concentrated switching structure.

Semantic content. We compare the average CLIP (Radford et al., 2021) image embedding of each anchor with 12 text descriptions of the task. The prompts are used only for this analysis, never for training. Anchors 0, 4, 10, and 16 all match best with "inserting the nut onto the peg" (cosine scores 0.248–0.256), while providing different states and graph contexts to the policy; Anchor 0 is active for 607 of 747 steps (81.3%) in the successful rollout. This shows that the atlas can preserve finer-grained execution variation within a shared coarse task semantic.

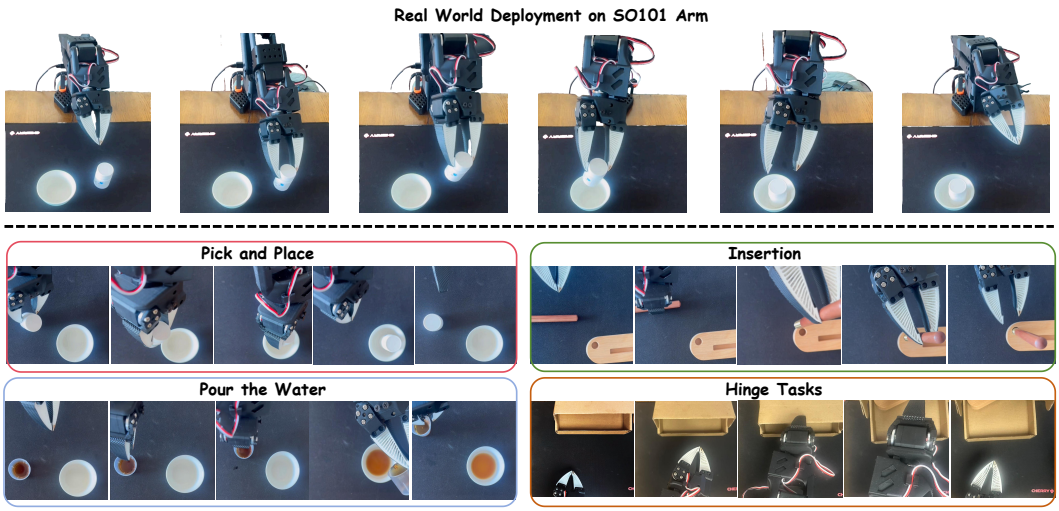


Figure 3: **Real-world tasks on the SO-101 arm.** The top row shows a Pick and Place rollout. The other panels show Pick and Place, Insertion, Pour the Water, and Hinge Tasks.

Table 4: **Real-world success rates (%) in real world.** 50 trials per method and task. Brackets: 95% Wilson confidence intervals. Δ: ActAtlas minus the best baseline (AutoCGP+DP).

Method	Pick and Place	Pour the Water	Insertion	Hinge Tasks	Mean
ACT	16.0 [8.3, 28.5]	12.0 [5.6, 23.8]	28.0 [17.5, 41.7]	14.0 [7.0, 26.2]	17.5
DP	30.0 [19.1, 43.8]	18.0 [9.8, 30.8]	22.0 [12.8, 35.2]	26.0 [15.9, 39.6]	24.0
AutoCGP+DP	42.0 [29.4, 55.8]	32.0 [20.8, 45.8]	34.0 [22.4, 47.8]	42.0 [29.4, 55.8]	37.5
ActAtlas (ours)	**54.0** [40.4, 67.0]	**66.0** [52.2, 77.6]	**58.0** [44.2, 70.6]	**72.0** [58.3, 82.5]	**62.5**
Δ	+12.0	+34.0	+24.0	+30.0	+25.0

4.4 Real-World Experiments

Setup. We evaluate four SO-101 tasks shown in Figure 3: Pick and Place, Pour the Water, Insertion, and Hinge Tasks. Each method uses 50 trials per task, with success requiring full-task completion within the time limit; detailed objectives and criteria are in Appendix 11.1.

Results. ActAtlas reaches 62.5% mean success, compared with 37.5% for AutoCGP+DP, 24.0% for DP, and 17.5% for ACT (Table 4). The largest gains are on Pour the Water (+34.0%) and Hinge Tasks (+30.0%), supporting transfer of the execution atlas from simulation to physical manipulation.

5 Conclusion

We presented ActAtlas, which addresses two challenges in long-horizon manipulation: learning an appropriate representation granularity and using relations between task stages during action generation. ActAtlas learns data-adaptive execution anchors through growth, merging, and splitting, connects them with an execution-flow graph, and uses graph context to condition a shared diffusion policy. Experiments in simulation and on a real robot show consistent improvements, especially on tasks with connected operations. These results show that representing both the current execution pattern and its relations to other patterns improves long-horizon generative robot policies.

Limitations. The transition-aware retriever summarizes execution history through only the previously selected anchor and a fixed empirical transition prior, while the graph is learned from successful demonstrations. Future work includes richer history-aware inference, learning from failures, and larger real-world evaluations.

REPRODUCIBILITY STATEMENT

Section 3 defines the model and training objectives. Appendix 6 gives the full algorithms and proofs, Appendix 7 lists the architecture, hyperparameters, and training schedules, and Appendices 8–11 describe the evaluation protocols, aggregation rules, and real-world trial counts.

AI USE STATEMENT

Generative AI tools were used only for limited language editing to improve the clarity and readability of the manuscript. They were not used to generate the research methodology, mathematical derivations, experimental results, or references. The authors reviewed all AI-assisted edits, verified the final technical content, and take full responsibility for the manuscript.

REFERENCES

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A Vision-Language-Action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

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. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Yuxuan Hu, Xiangyu Chen, Chuhao Zhou, Yuxi Liu, Gen Li, Jindou Jia, and Jianfei Yang. Trace-focused diffusion policy for multi-modal action disambiguation in long-horizon robotic manipulation. *arXiv preprint arXiv:2602.07388*, 2026.

De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8557–8566. IEEE, 2019.

Haidong Huang, Xixin Zhao, Yaohua Zhou, Jiayu Song, Jiayi Zhang, Jun Ma, Haiyue Zhu, and Xiaocong Li. Corde: Concept-prior routed diffusion experts for structural generalization in robot manipulation. *arXiv preprint arXiv:2606.21935*, 2026.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: A Vision-Language-Action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Xiaogang Jia, Denis Blessing, Xinkai Jiang, Moritz Reuss, Atalay Donat, Rudolf Lioutikov, and Gerhard Neumann. Towards diverse behaviors: A benchmark for imitation learning with human demonstrations. In *International Conference on Learning Representations*, volume 2024, pp. 41666–41686, 2024.

Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning*, pp. 3418–3428. PMLR, 2019.

Oliver Kroemer, Scott Niekum, and George Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *Journal of machine learning research*, 22(30):1–82, 2021.

Xiang Li, Varun Belagali, Jinghuan Shang, and Michael S Ryoo. Crossway diffusion: Improving diffusion-based visuomotor policy via self-supervised learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16841–16849. IEEE, 2024.

Zhixuan Liang, Yao Mu, Hengbo Ma, Masayoshi Tomizuka, Mingyu Ding, and Ping Luo. Skilldiffuser: Interpretable hierarchical planning via skill abstractions in diffusion-based task execution. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16467–16476. IEEE, 2024.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Ruizhe Liu, Qian Luo, and Yanchao Yang. Infocon: Concept discovery with generative and discriminative informativeness. In *International Conference on Learning Representations*, volume 2024, pp. 37726–37752, 2024.

Ruizhe Liu, Pei Zhou, Qian Luo, Li Sun, Jun Cen, Yibing Song, and Yanchao Yang. HiMaCon: Discovering hierarchical manipulation concepts from unlabeled multi-modal data. *Advances in Neural Information Processing Systems*, 38:105366–105403, 2026.

Xiao Ma, Sumit Patidar, Iain Haughton, and Stephen James. Hierarchical diffusion policy for kinematics-aware multi-task robotic manipulation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18081–18090. IEEE, 2024.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 1134–1141. IEEE, 2018.

Saif Sidhik, Mohan Sridharan, and Dirk Ruiken. Towards an online framework for changing-contact robot manipulation tasks. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5203–5210. IEEE, 2021.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Weikang Wan, Yifeng Zhu, Rutav Shah, and Yuke Zhu. Lotus: Continual imitation learning for robot manipulation through unsupervised skill discovery. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 537–544. IEEE, 2024.

Mengda Xu, Zhenjia Xu, Cheng Chi, Manuela Veloso, and Shuran Song. Xskill: Cross embodiment skill discovery. In *Conference on robot learning*, pp. 3536–3555. PMLR, 2023.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Ruijie Zheng, Yongyuan Liang, Shuaiyi Huang, Jianfeng Gao, Hal Daumé III, Andrey Kolobov, Furong Huang, and Jianwei Yang. Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. In *International Conference on Learning Representations*, volume 2025, pp. 54277–54296, 2025.

Pei Zhou, Ruizhe Liu, Qian Luo, Fan Wang, Yibing Song, and Yanchao Yang. Autocgp: Closed-loop concept-guided policies from unlabeled demonstrations. In *International Conference on Learning Representations*, volume 2025, pp. 44815–44842, 2025.

Yifeng Zhu, Peter Stone, and Yuke Zhu. Bottom-up skill discovery from unsegmented demonstrations for long-horizon robot manipulation. *IEEE Robotics and Automation Letters*, 7(2):4126–4133, 2022.

APPENDIX CONTENTS

6 Method Details and Proofs 14

6.1 Anchor Roles and Data Interfaces 14

6.2 Offline Assignment Rule 14

6.3 Gradients of the Stage 1 Losses 15

6.4 Error Decomposition and Prototype Updates 16

6.5 Adaptive Anchor Set and Relation Scores 17

6.6 Graph Encoder and Permutation Equivariance 20

6.7 Gradients of the Denoising Loss with Respect to Relations 21

6.8 Policy Network and Parameter Updates 22

6.9 Diffusion Sampling and Closed-Loop Execution 23

7 Implementation and Training Details 24

7.1 Architecture, Hyperparameters, and Cost 25

7.2 Data Windows and Inputs 26

7.3 Optimization 26

7.4 Schedule of Structural Operations 27

7.5 Runtime Execution 28

7.6 Checkpoints and Reproducibility 28

8 Experimental Protocol 28

8.1 Benchmarks and Averaging 28

8.2 Baselines and Ablation Variants 29

8.3 Controlled Conditioning and Retrieval on LIBERO-Long 29

8.4 Selected LIBERO Tasks 30

8.5 Standard Errors 30

9 Anchor Analysis Details 30

9.1 Rollout Collections and Index Conventions 31

9.2 Transition Statistics 31

9.3 CLIP Probe 32

9.4 Prompt Margin Bound 33

10 Computational Overhead 33

11 Real-World Details 34

11.1 Task Objectives and Success Criterion 34

11.2 Success Counts 34

11.3 Wilson Confidence Intervals 34

6 METHOD DETAILS AND PROOFS

This appendix expands Section 3. Table 5 links each part of the method to its details. Two general notes apply throughout. First, statements about prototype updates assume fixed queries and fixed assignments within one batch; they describe one update step, not the whole training process. Second, the analysis does not assume human labels for anchors or a model of robot dynamics.

Table 5: **Guide to Appendix 6.**

Component	Main text	Appendix
Anchor assignment and losses	Eqs. (2)–(5)	6.1, 6.2, 6.3
Prototype updates	Eq. (6)	6.4
Adaptive set and relations	Section 3.2, Eq. (7)	6.5
Graph context	Eqs. (10)–(11)	6.6
Relation gradients	Sec. 3.4	6.7
Policy and training	Eqs. (12), (13)	6.8
Transition prior and execution	Eqs. (8)–(9)	6.9

6.1 ANCHOR ROLES AND DATA INTERFACES

Two roles of an anchor. Each anchor k has a key μ_k and a value ν_k. The key is used to match states, and the value is the content passed to the graph encoder. The active set $\mathcal{K}$ sets the granularity, and B stores directed relations over the same slots. The policy is conditioned on the fused context g_t, not on the integer index z_t alone. This is the main difference from concept-based policies: AutoCGP predicts a distribution over a fixed set of learned concepts (Zhou et al., 2025), and HiMaCon uses continuous concepts with an auxiliary prediction loss (Liu et al., 2026). ActAtlas instead adapts the number of anchors and explicitly aggregates the values of related anchors. The difference lies in how the representation is organized and used, not in the words "anchor" and "concept."

Notation. Let d_v and d_s be the dimensions of the observation feature and the proprioception. Keys, values, and queries have width d, the graph encoder has width d, and $d_g = d$. A Stage 1 batch contains n_w windows of length T_w, so it has $m = n_w T_w$ time steps. A Stage 2 sample is $(x_t, v_t, s_{t-L+1:t}, A_t)$ with $A_t \in \mathbb{R}^{H \times d_a}$. Three lengths have different meanings: T_w is the Stage 1 window length, L is the proprioceptive history length, and H is the action horizon. We write $q_t^{\mathrm{off}} = \mathrm{Norm}(E_\eta(x_t) + \alpha_u E_u(u_t))$ for Stage 1 atlas organization and $q_t^{\mathrm{on}} = \mathrm{Norm}(E_\eta(x_t))$ for Stage 2 and deployment retrieval. Stage 2 prototype aggregation and the construction of P^{tr} use the same time-free online retriever; Appendix 6.9 gives the closed-loop construction.

Feature preparation. If the state feature already has width d, it is passed directly to E_η. Otherwise, adaptive average pooling along the feature axis converts it to width d; this is a fixed reshaping, not a learned layer. If no separate state feature is given, the latest proprioceptive vector is used. The state feature must be computed in the same way during training and deployment. For a nonzero vector b, $\mathrm{Norm}(b) = b/\|b\|_2$. In code we use $b/\max(\|b\|_2, \epsilon_{\mathrm{norm}})$ for numerical stability; all statements about unit vectors refer to the exact case. Anchors with no assigned states are not updated, and inactive slots are excluded from matching and from the graph.

Window boundaries. All sequence operations stay inside one window. Transition counts never connect the end of one window to the start of another. Window extraction, padding, action scaling, and alignment of samples belong to the data loader and are not learned parts of the atlas. Appendix 7.2 describes the loader.

6.2 OFFLINE ASSIGNMENT RULE

Let $I = (i_1, \ldots, i_K)$ list the active slots in increasing order. Cosine matching gives a local index $\widetilde{k}_t \in \{1, \ldots, K\}$ and the soft probabilities in Eq. (2). The hard indices are processed backward

within each window:

$$\begin{aligned} k_{T_w} &= \widetilde{k}_{T_w}, \\ k_t &= \min\{\widetilde{k}_t, k_{t+1}\}, \qquad t = T_w - 1, \dots, 1, \\ z_t &= i_{k_t}. \end{aligned} \tag{14}$$

Each maximal run of equal indices forms one segment. Let $b_1 = 1 < b_2 < \cdots < b_{R+1} = T_w + 1$, where the interior boundaries are the steps t with $z_t \neq z_{t-1}$; the segments are $[b_r, b_{r+1})$ for $r = 1, \dots, R$.

Property. The recurrence gives $k_t = \min_{u \geq t} \widetilde{k}_u$, so $k_t \leq k_{t+1}$. Moreover, k is the largest non-decreasing sequence that lies below $\widetilde{k}$. If j is any non-decreasing sequence with $j \leq \widetilde{k}$, then $j_t \leq j_u \leq \widetilde{k}_u$ for all $u \geq t$, so $j_t \leq k_t$. This describes the index operation exactly; it is not an optimizer of the cosine scores. The rule depends on the stored slot order and is used only for offline windows. At test time the query has length one and the rule has no effect.

Offline and online queries. The soft probabilities r_t are computed before the backward rule and are not changed by it. The rule therefore changes the forward value in Eq. (3) but not the probabilities used for its gradient. Algorithm 1 gives the full procedure.

Algorithm 1 Offline window assignment

Input: Window $x_{1:T_w}$, times $u_{1:T_w}$, active memory $(\mathcal{K}, M, V)$, encoders E_η, E_u, scale α_u, temperature τ_q

Output: Queries q^{off}, indices z, probabilities r, values ζ, segments

$I \leftarrow$ ordered active slots; take a snapshot of the active keys and values
for $t = 1, \dots, T_w$ **do**
  $q_t^{\text{off}} \leftarrow \text{Norm}(E_\eta(x_t) + \alpha_u E_u(u_t))$
  Compute r_t and $\widetilde{k}_t$ by Eq. (2)
end for
$k_{T_w} \leftarrow \widetilde{k}_{T_w}$
for $t = T_w - 1, \dots, 1$ **do**
  $k_t \leftarrow \min\{\widetilde{k}_t, k_{t+1}\}$
end for
for $t = 1, \dots, T_w$ **do**
  $z_t \leftarrow i_{k_t}$; $\zeta_t^{\text{soft}} \leftarrow \sum_{j=1}^{K} r_t(i_j)\, \nu_{i_j}$
  $\zeta_t \leftarrow \zeta_t^{\text{soft}} + \text{sg}(\nu_{z_t} - \zeta_t^{\text{soft}})$
end for
Form segments from the runs of equal z_t
return $(q^{\text{off}}, z, r, \zeta, \text{segments})$

6.3 Gradients of the Stage 1 Losses

Straight-through gradient. Hold the active keys and values fixed, and write $\ell_{ik} = (q_i^{\text{off}})^\top \mu_k / \tau_q$. Eq. (3) returns ν_{z_i} in the forward pass and differentiates ζ_i^{soft} in the backward pass. As in VQ-VAE (Van Den Oord et al., 2017), this is a chosen surrogate gradient, not the true derivative of the hard argmax. From the softmax Jacobian,

$$\begin{aligned} \frac{\partial r_i(j)}{\partial \ell_{ik}} &= r_i(j)\big(\mathbf{1}_{\{j=k\}} - r_i(k)\big), \\ \left.\frac{\partial \zeta_i}{\partial \ell_{ik}}\right|_{\text{ST}} &= r_i(k)\big(\nu_k - \zeta_i^{\text{soft}}\big), \\ J_{\zeta,q}^{\text{ST}} &= \frac{1}{\tau_q} \sum_k r_i(k)\big(\nu_k - \zeta_i^{\text{soft}}\big)\mu_k^\top. \end{aligned} \tag{15}$$

For an upstream gradient $b_i = \partial\mathcal{L}/\partial\zeta_i$, the query receives $\widetilde{\nabla}_{q_i}\mathcal{L} = (J^{\mathrm{ST}}_{\zeta,q})^\top b_i$. If $q_i = h_i/\|h_i\|_2$, the gradient before normalization and the upstream gradient from the first term of Eq. (4) are

$$\begin{aligned}\widetilde{\nabla}_{h_i}\mathcal{L} &= \frac{I - q_i q_i^\top}{\|h_i\|_2}\,\widetilde{\nabla}_{q_i}\mathcal{L},\\ b_i^{\mathrm{state}} &= \frac{2}{md}\big(\nu_{z_i} - x_i\big).\end{aligned} \tag{16}$$

The second reconstruction term. The second term of Eq. (4) stops the gradient through ζ_i. Its gradient $2\lambda_q(x_i - \mathrm{sg}(\zeta_i))/(md)$ flows only to the input feature. When x_i is a fixed feature, this term has no trainable parameter to update. The encoder is trained by the first term and by the contrastive losses. The values are buffers updated only by Eq. (6). If an upstream feature extractor is made trainable, its extra gradient paths must be specified separately.

Contrastive loss. The action branch receives the current action a_i, not the future chunk A_i. In our implementation it is zero-padded to width Hd_a: $[a_i; 0_{(H-1)d_a}]$; in general, longer vectors are truncated and shorter ones are padded. Each projection head is Linear, GELU, and Linear with output width d_e, followed by normalization. The scales are $\gamma_b = \min\{\exp(\kappa_b), 100\}$, with $\exp(\kappa_b)$ initialized to $1/0.07$. For $b \in \{v, a\}$ and a batch of size $m \geq 2$, the symmetric InfoNCE loss is

$$\begin{aligned}S^b_{ij} &= \gamma_b (e^c_i)^\top e^b_j,\\ \mathcal{L}_b &= -\frac{1}{2m}\sum_{i=1}^{m}\left[\log\frac{\exp(S^b_{ii})}{\sum_j \exp(S^b_{ij})} + \log\frac{\exp(S^b_{ii})}{\sum_j \exp(S^b_{ji})}\right].\end{aligned} \tag{17}$$

Let $P^b_{ij} = \exp(S^b_{ij})/\sum_r \exp(S^b_{ir})$ and $C^b_{ij} = \exp(S^b_{ij})/\sum_r \exp(S^b_{rj})$ be the row and column softmaxes. Then

$$\begin{aligned}\frac{\partial\mathcal{L}_b}{\partial S^b_{ij}} &= \frac{P^b_{ij} + C^b_{ij} - 2\cdot\mathbf{1}_{\{i=j\}}}{2m},\\ \frac{\partial S^b_{ij}}{\partial e^c_i} &= \gamma_b\, e^b_j.\end{aligned} \tag{18}$$

These gradients train both projection heads and, through ζ^{soft}_i, the encoder. A batch with one sample has zero contrastive loss. The contrastive loss uses vision and action only; the language prompts in Section 4.3 are never used for training.

6.4 Error Decomposition and Prototype Updates

This section expands the sharing and consolidation interpretation in Sec. 3.2. Proposition 6.1 decomposes the forward state-preservation error in Eq. (4) into within-group variation and prototype mismatch. Proposition 6.2 characterizes Eq. (6) as constrained fitting in query space. The assignments connect these two analyses: they specify which inputs share a reference and which encoded queries update it. The objectives remain distinct, so the query-space bound is interpreted for fixed queries and assignments.

Proposition 6.1 (Error decomposition)**.** *Let $\mathcal{I}_k = \{i : z_i = k\}$ be a nonempty group of size n_k with mean $\bar{x}_k = n_k^{-1}\sum_{i\in\mathcal{I}_k} x_i$. For any shared vector ν,*

$$\sum_{i\in\mathcal{I}_k}\|x_i - \nu\|_2^2 = \sum_{i\in\mathcal{I}_k}\|x_i - \bar{x}_k\|_2^2 + n_k\|\bar{x}_k - \nu\|_2^2. \tag{19}$$

Without a norm constraint, the best shared vector is $\bar{x}_k$.

Proof. Write $x_i - \nu = (x_i - \bar{x}_k) + (\bar{x}_k - \nu)$ and expand:

$$\sum_{i\in\mathcal{I}_k}\|x_i - \nu\|_2^2 = \sum_{i\in\mathcal{I}_k}\|x_i - \bar{x}_k\|_2^2 + n_k\|\bar{x}_k - \nu\|_2^2 + 2\Big\langle\sum_{i\in\mathcal{I}_k}(x_i - \bar{x}_k),\, \bar{x}_k - \nu\Big\rangle. \tag{20}$$

The centered sum is zero, so the cross term vanishes. Only the last term depends on ν, and it is zero at $\nu = \bar{x}_k$. □

Taking $\nu = \nu_k$ in Eq. (19) gives the shared-value error discussed in Sec. 3.2. The first term measures variation among states that share an anchor, while the second measures their common value's mismatch to the group mean. This connects the assignment representation to the content retained by a shared prototype. The implemented consolidation uses normalized encoded queries, rather than directly minimizing this input-feature error; its separate fitting interpretation follows.

Proposition 6.2 (Prototype update)**.** *Let $q_1, \dots, q_n$ be fixed unit vectors with mean $\bar{q}$, let u_0 be the previous unit prototype, and let $0 < \rho < 1$. Define $\mathcal{F}(u) = n^{-1} \sum_i \|q_i - u\|_2^2$. If $b = \rho u_0 + (1-\rho)\bar{q} \neq 0$, then $u^+ = \mathrm{Norm}(b)$ is the unique minimizer of $J(u) = \rho\|u - u_0\|_2^2 + (1-\rho)\mathcal{F}(u)$ over unit vectors, and*

$$\mathcal{F}(u^+) + \frac{\rho}{1-\rho}\|u^+ - u_0\|_2^2 \leq \mathcal{F}(u_0). \tag{21}$$

Proof. For unit vectors, $\|q - u\|_2^2 = 2 - 2q^\top u$. Hence

$$\begin{aligned} \mathcal{F}(u) &= 2 - 2\bar{q}^\top u, \\ J(u) &= 2 - 2b^\top u, \\ \min_{\|u\|_2=1} J(u) &= 2 - 2\|b\|_2. \end{aligned} \tag{22}$$

By Cauchy–Schwarz, the unique minimizer is $u^+ = b/\|b\|_2$. Since u_0 is also a unit vector, $J(u^+) \leq J(u_0) = (1-\rho)\mathcal{F}(u_0)$, that is, $\rho\|u^+ - u_0\|_2^2 + (1-\rho)\mathcal{F}(u^+) \leq (1-\rho)\mathcal{F}(u_0)$. Dividing by $1-\rho$ gives Eq. (21). Setting $u_0 = \mu_k$ gives the key update, and setting $u_0 = \nu_k$ gives the value update. □

Remarks. If $\rho > 1/2$, then $\|\bar{q}\|_2 \leq 1$ implies $\|b\|_2 \geq 2\rho - 1 > 0$, so the condition $b \neq 0$ always holds; this includes our setting $\rho = 0.95$. The result is local to one batch and does not require later encoder or structural updates to minimize the same objective. Keys and values are stored separately because they play different roles. In our implementation they start equal and receive the same updates (EMA, growth, merge, and split), so $\mu_k = \nu_k$ throughout. The extra information used by the policy comes from the graph-enhanced value $\widetilde{\nu}_k$, not from a difference between keys and values.

6.5 Adaptive Anchor Set and Relation Scores

The grow/merge/split rules in Sec. 3.2 organize sharing granularity. Below, Eq. (24) relates the growth test to nearest-prototype distance, Proposition 6.3 characterizes the normalized usage-weighted merge and bounds its local cost, and Eq. (27) describes the size of the split proposal. These statements concern local coverage and fitting under the stated conditions. The batch order and operation schedule are specified in Algorithm 3 and Sec. 7.4.

Initialization and running statistics. The memory has $K_{\max}$ slots, and the first K_0 are active at the start. Keys are initialized with row-normalized orthogonal initialization, values copy the keys, usage weights start at one, and dispersions start at zero. Local indices used for matching or classification are mapped back to the global slots through I. For each occupied anchor, we update the usage weight and the per-dimension dispersion:

$$\begin{aligned} w_k &\leftarrow \rho w_k + (1-\rho) n_k \qquad (n_k > 0), \\ \widehat{D}_k &= \frac{1}{n_k d} \sum_{i:z_i=k} \|q_i - \mu_k^+\|_2^2, \\ D_k &\leftarrow \rho_D D_k + (1-\rho_D)\widehat{D}_k \quad (n_k \geq 2). \end{aligned} \tag{23}$$

The dispersion is measured around the key after the current EMA step, with $\rho_D = 0.9$. A separate counter records how many samples have contributed but is not used to normalize D_k. Usage weights are moving averages, not total counts. New anchors from growth or splitting use the first inactive slot.

Growth. The growth descriptor is the normalized mean of the input states in a window, $\chi_{\mathcal{W}} = \mathrm{Norm}(|\mathcal{W}|^{-1} \sum_{t \in \mathcal{W}} x_t)$. It is not an action chunk and not an average of queries. For unit vectors, the growth test can be written as a distance test:

$$\max_{k \in \mathcal{K}} \chi_{\mathcal{W}}^{\top} \mu_k < \tau_g \quad \Longleftrightarrow \quad \min_{k \in \mathcal{K}} \|\chi_{\mathcal{W}} - \mu_k\|_2^2 > 2(1 - \tau_g). \tag{24}$$

A new anchor is initialized with key and value $\chi_{\mathcal{W}}$, usage one, and zero dispersion. Let $C_M(x) = \min_{\mu \in M} \|x - \mu\|_2^2$ be the distance from x to its nearest prototype. Adding $\chi_{\mathcal{W}}$ gives $C_{M \cup \{\chi_{\mathcal{W}}\}}(x) \leq C_M(x)$ for every x, with zero error at $x = \chi_{\mathcal{W}}$, because the set of candidates only grows. This statement concerns nearest-prototype coverage with the old prototypes fixed; it does not claim that the full training loss decreases.

Merging. The most similar pair of distinct active keys (i, j) is merged when $\mu_i^{\top} \mu_j > \tau_m$. Slot i keeps the weighted average of keys and of values, and slot j is deactivated.

Proposition 6.3 (Merge cost)**.** *Let $w_i, w_j > 0$, $S = w_i + w_j$, $c = w_i \mu_i + w_j \mu_j \neq 0$, and $\delta^2 = \|\mu_i - \mu_j\|_2^2$. Over unit vectors, $J(u) = w_i \|u - \mu_i\|_2^2 + w_j \|u - \mu_j\|_2^2$ is minimized at $u^+ = c/\|c\|_2$, and*

$$\min_{\|u\|_2 = 1} J(u) = 2S - 2\|c\|_2 = \frac{2 w_i w_j \delta^2}{S + \|c\|_2} \leq \min(w_i, w_j)\, \delta^2. \tag{25}$$

Proof. For unit vectors, $J(u) = 2S - 2c^{\top} u$, so Cauchy–Schwarz gives the unique minimizer $c/\|c\|_2$ and the minimum $2(S - \|c\|_2)$. Next,

$$\begin{aligned} \|c\|_2^2 &= w_i^2 + w_j^2 + 2 w_i w_j \mu_i^{\top} \mu_j = S^2 - w_i w_j \delta^2, \\ 2(S - \|c\|_2) &= \frac{2(S^2 - \|c\|_2^2)}{S + \|c\|_2} = \frac{2 w_i w_j \delta^2}{S + \|c\|_2}. \end{aligned} \tag{26}$$

Evaluating J at the feasible points $u = \mu_i$ and $u = \mu_j$ gives $w_j \delta^2$ and $w_i \delta^2$, which proves the upper bound. The same argument applies to the values. □

After a merge, the kept slot receives the sum of the two usage weights (with a small constant in the denominator of the weighted average for stability) and the larger of the two dispersions. The removed slot has its usage, dispersion, and sample counter set to zero, and its prototype contents are ignored. Past transition counts are not recomputed. The bound concerns only the merged pair; it does not claim that the full loss decreases after reassignment.

Splitting. The occupied anchor with the largest dispersion is split if $D_k > \tau_s$ and a free slot exists. With a random unit vector r, we form $c_+ = \mathrm{Norm}(\mu_k + \epsilon_s r)$ and $c_- = \mathrm{Norm}(\mu_k - \epsilon_s r)$. The old slot receives c_+, and the first free slot receives c_-, for both key and value. The usage is divided equally, and both dispersions and sample counters are reset. We use $\epsilon_s = 0.1$. For $0 < \epsilon_s < 1$,

$$\|\mu_k \pm \epsilon_s r\|_2 \geq 1 - \epsilon_s > 0, \qquad \|c_{\pm} - \mu_k\|_2 \leq 2\epsilon_s. \tag{27}$$

The second inequality follows by inserting $\mu_k \pm \epsilon_s r$ between the two unit vectors and using $\big|\|\mu_k \pm \epsilon_s r\|_2 - 1\big| \leq \epsilon_s$. A split is a small local proposal, not a principal-component estimate; its usefulness is decided by later assignments and updates.

Range of the split threshold. Since $\|q - \mu_k\|_2^2 \leq 4$ for unit vectors, and a convex moving average that starts at zero stays in the same range,

$$0 \leq \widehat{D}_k \leq \frac{4}{d}, \qquad 0 \leq D_k \leq \frac{4}{d}. \tag{28}$$

A threshold τ_s that allows splitting must therefore be below $4/d$. This is necessary but not sufficient for a split. The threshold and the feature dimension must be reported together.

Persistent slots. Structural operations run without gradients. They change the active mask and the memory statistics, but they do not merge, move, or reset rows and columns of B. The full score matrix is kept for all $K_{\max}$ slots, and its active submatrix is used to build the graph. A reactivated slot therefore keeps its stored scores. This storage rule must be kept when reproducing the model. Algorithm 2 summarizes the operations in one batch, and Appendix 7.4 gives the schedule.

Algorithm 2 Structural operations in one Stage 1 batch

Input: Windows $\{\mathcal{W}_b\}_{b=1}^{n_w}$; active memory and statistics; $K_{\max}, \tau_g, \tau_m, \tau_s, \epsilon_s, p_m, p_s$
Output: Updated memory and active set; B unchanged

for $b = 1, \ldots, \min(n_w, 4)$ **do**
  $\chi_b \leftarrow \mathrm{Norm}(|\mathcal{W}_b|^{-1} \sum_{t \in \mathcal{W}_b} x_t)$
  if a free slot exists **and** $\max_{k \in \mathcal{K}} \chi_b^\top \mu_k < \tau_g$ **then**
    Activate the first free slot j with $\mu_j = \nu_j = \chi_b$, $w_j = 1$, $D_j = 0$; reset its sample counter
  end if
end for
Draw $b_m \sim \mathrm{Bernoulli}(p_m)$
if $b_m = 1$ **and** $|\mathcal{K}| \geq 2$ **then**
  $(i, j) \leftarrow \arg\max_{i \neq j \in \mathcal{K}} \mu_i^\top \mu_j$
  if $\mu_i^\top \mu_j > \tau_m$ **then**
    Merge keys and values by Proposition 6.3; keep slot i with combined usage and $D_i \leftarrow \max(D_i, D_j)$
    Deactivate j and clear its statistics
  end if
end if
Draw $b_s \sim \mathrm{Bernoulli}(p_s)$
if $b_s = 1$ **and** a free slot exists **then**
  $k \leftarrow \arg\max_{i \in \mathcal{K}} D_i$
  if $D_k > \tau_s$ **then**
    $j \leftarrow$ first free slot; $r \leftarrow \mathrm{Norm}(\varepsilon)$ with $\varepsilon \sim \mathcal{N}(0, I_d)$
    $(\mu_k, \nu_k) \leftarrow (c_+, c_+)$ and $(\mu_j, \nu_j) \leftarrow (c_-, c_-)$, both computed from the old μ_k
    Activate j, divide the usage equally, and reset both dispersions
  end if
end if
return the updated memory

Transition evidence and score normalization. Within each window, consecutive assignments give the counts ΔN_{ij}, and Eq. (7) updates B only for $i \neq j$. Starting from zero, a window seen many times adds its evidence many times. The scores are accumulated logits, not probability estimates: $W_{ij} \propto \exp(B_{ij})$. For example, a row of equal scores gives uniform weights, whether or not that row has observed transitions. Every finite B gives positive weight to every active pair, so unobserved transitions are not hard-masked. Restricting the update to $i \neq j$ does not remove self-attention, which is handled by ϵ_{self} in Eq. (10). Our code also contains an optional recovery array that counts transitions in failed rollouts, but it is not used in the policy described here.

Algorithm 3 Stage 1: learning the execution atlas

Input: Demonstration windows with (x, v, a, u); initialized memory; $B = 0$; encoders and projection heads; optimizer $\mathcal{O}_1$
Output: Encoder, active set, prototypes, and relation scores

for each training batch **do**
  Run Algorithm 1 on each window to get $(q^{\mathrm{off}}, z, r, \zeta)$
  Keep these forward results for the loss of this batch
  Without gradients, update occupied keys and values by Eq. (6)
  Update usage and dispersion by Eq. (23)
  Count within-window transitions and update B by Eq. (7)
  Compute $\mathcal{L}_{\mathrm{anchor}}$ by Eqs. (4) and (5)
  Run Algorithm 2 without gradients
  Backpropagate through the stored soft assignments, clip the gradient norm to 1, and step $\mathcal{O}_1$
  Step the learning-rate scheduler and run the periodic checks (Table 10)
end for
return $(E_\eta, E_u, \mathcal{K}, M, V, B)$ and the projection heads

Algorithm 3 follows the order of our implementation. The batch uses the memory snapshot taken before any update. The EMA update and the score update happen during the forward pass, structural operations come after the losses are computed, and the optimizer then updates the network parameters. New anchors are available from the next batch on; they do not relabel the current batch.

6.6 Graph Encoder and Permutation Equivariance

This section gives the layer operations underlying Eqs. (10)–(11). Shared projections, residual paths, normalization, and nodewise feed-forward transformations combine related-anchor messages into a condition of width d_g. On a fixed evaluation atlas, deterministic graph encoding assigns a condition to each retrieved index; current observations and proprioceptive history enter separately through y_t. Proposition 6.4 shows that jointly relabeling nodes, relations, and the selected index preserves the corresponding context.

Multi-head graph attention. The active values, after row normalization, form the initial node features $\mathsf{H}^{(0)} \in \mathbb{R}^{K\times d}$. The encoder has two layers and four heads, with no dropout. For layer ℓ and head h, let $d_h = d/n_h$ and compute row-wise projections $\mathsf{Q}^{\ell,h}, \mathsf{K}^{\ell,h}, \mathsf{V}^{\ell,h}$. The multi-head form of Eq. (10) is

$$\begin{aligned}
\mathsf{S}^{\ell,h} &= \mathsf{Q}^{\ell,h}(\mathsf{K}^{\ell,h})^\top/\sqrt{d_h},\\
\mathsf{A}^{\ell,h} &= \mathrm{softmax}_{\mathrm{row}}\big((W+\epsilon_{\mathrm{self}}I_K)\odot \mathsf{S}^{\ell,h}\big),\\
\mathsf{M}^{\ell,h} &= \mathsf{A}^{\ell,h}\mathsf{V}^{\ell,h},\\
\mathsf{Z}^{\ell} &= \mathrm{LN}_1\left(\mathsf{H}^{(\ell)} + \mathrm{Out}_\ell([\mathsf{M}^{\ell,1};\cdots;\mathsf{M}^{\ell,n_h}])\right),\\
\mathsf{H}^{(\ell+1)} &= \mathrm{LN}_2\left(\mathsf{Z}^{\ell} + \mathrm{FF}_\ell(\mathsf{Z}^{\ell})\right).
\end{aligned} \tag{29}$$

Heads are concatenated along the feature axis. The query, key, and value projections have no bias, and the output projection has a bias. The feed-forward layer is $d \to 2d \to d$ with GELU, and layer normalization is applied to each node separately. The same W is used in every layer. We set $\epsilon_{\mathrm{self}} = 0.1$; it scales the self score before the softmax. Note that a zero multiplier gives a zero logit, not a removed edge.

Fusion. After the graph encoder, the selected global slot is mapped to its local row. The fusion layer applies Linear ($2d \to d$), LayerNorm, and GELU to the concatenation of the original and graph-enhanced values, which gives Eq. (11) with $d_g = d$. The width of g_t does not depend on K, but its value depends on the other active anchors and the learned relations. A fixed width does not mean a fixed computation cost (Appendix 7.1).

Proposition 6.4 (Permutation equivariance)**.** *Fix an atlas and a deterministic graph encoder with shared node-wise transformations. Let P be a permutation matrix, and replace V with PV and the active scores B with $PBP^\top$. Then*

$$\mathrm{GraphEncoder}_\phi(PV, PWP^\top) = P\ \mathrm{GraphEncoder}_\phi(V, W). \tag{30}$$

If the selected index is permuted in the same way, the fusion layer returns the same context g_t. The same holds for anchor selection when the maximizer is unique, or when ties are broken by a rule that is permuted in the same way.

Proof. Row softmax commutes with a joint permutation of rows and columns, so $W' = \mathrm{softmax}_{\mathrm{row}}(PBP^\top) = PWP^\top$. Row normalization and shared linear projections commute with P, so

$$\begin{gathered}
\mathsf{Q}' = P\mathsf{Q},\quad \mathsf{K}' = P\mathsf{K},\quad \mathsf{V}' = P\mathsf{V},\quad \mathsf{S}' = P\mathsf{S}P^\top,\\
W' + \epsilon_{\mathrm{self}}I = P(W+\epsilon_{\mathrm{self}}I)P^\top,\quad \mathsf{A}' = P\mathsf{A}P^\top,\quad \mathsf{M}' = P\mathsf{M}.
\end{gathered} \tag{31}$$

Concatenation, the output projection, residual connections, and node-wise feed-forward and normalization layers keep the same row permutation. Induction over layers gives Eq. (30). If the selected row is e_k, the new selector is Pe_k, and

$$(Pe_k)^\top PV = e_k^\top V, \qquad (Pe_k)^\top P\widetilde{V} = e_k^\top \widetilde{V}. \tag{32}$$

So the fusion layer receives the same two vectors and returns the same g_t. For selection, the cosine scores are permuted together with the keys, and the unique maximizer is relabeled accordingly. □

This property concerns the graph computation on a given atlas. It does not cover the offline backward rule in Eq. (14), which uses the numeric slot order. Random dropout masks would also break a sample-wise comparison, but our encoder has no dropout. The property does not imply that two different atlases give the same policy condition.

6.7 Gradients of the Denoising Loss with Respect to Relations

This section derives how the denoising loss in Eq. (12) backpropagates to the relation scores B, as summarized in Sec. 3.4. We separate the context-to-denoiser chain rule, a single attention head with fixed inputs, and the total gradient of the shared relation matrix. The local message derivative explains how changing one relation weight changes an aggregate; the row-centered score gradient links the accumulated effect to action error. Because the same W is used in every layer and head, its total gradient sums all such uses. The sampled-loss expressions below give the expected-loss gradient when differentiation and expectation can be exchanged.

Proposition 6.5 (Gradient through the graph context)**.** *Fix the active set, the prototypes, the selected anchor, the sensory input, the noisy action, and the diffusion step. Assume the fusion layer, the graph encoder, and the denoiser are differentiable at the current parameters. Let $\omega = (\mathrm{vec}(B), \phi)$, $\widehat{\epsilon}_t = \epsilon_\theta(X_\tau, \tau, c_t)$, $J_{g,\omega} = \partial g_t/\partial\omega$, and $J_{\widehat{\epsilon},g} = \partial\,\mathrm{vec}(\widehat{\epsilon}_t)/\partial g_t$. Then the gradient of the sampled loss is the integrand of*

$$\nabla_\omega \mathcal{L}_{\mathrm{diff}} = \frac{2}{H d_a}\mathbb{E}\Big[J_{g,\omega}^\top J_{\widehat{\epsilon},g}^\top \mathrm{vec}(\widehat{\epsilon}_t - \epsilon)\Big], \tag{33}$$

and Eq. (33) holds exactly whenever differentiation and expectation can be exchanged.

Proof. Let $r = \mathrm{vec}(\widehat{\epsilon} - \epsilon)$ and $n_a = H d_a$. With the noise and the target fixed,

$$\begin{aligned} \ell_{\mathrm{diff}} &= n_a^{-1} r^\top r, \\ d\ell_{\mathrm{diff}} &= 2 n_a^{-1} r^\top d\,\mathrm{vec}(\widehat{\epsilon}), \\ d\,\mathrm{vec}(\widehat{\epsilon}) &= J_{\widehat{\epsilon},g}\, dg, \\ dg &= J_{g,\omega}\, d\omega. \end{aligned} \tag{34}$$

Therefore $\nabla_\omega \ell_{\mathrm{diff}} = (2/n_a) J_{g,\omega}^\top J_{\widehat{\epsilon},g}^\top r$. Taking the expectation gives the result when the exchange is justified, for example by an integrable local bound on the sampled gradients. For a finite batch, the loss is a finite sum and no exchange is needed. The denoiser parameters θ receive their usual direct gradient and are not part of ω. □

One attention head. Fix the inputs of one head. Let $s_{ij} = (U_Q h_i)^\top (U_K h_j)/\sqrt{d_h}$, $a_{ij} = (W_{ij} + \epsilon_{\mathrm{self}} \mathbf{1}_{\{i=j\}}) s_{ij}$, $\alpha_{ij} = \exp(a_{ij})/\sum_k \exp(a_{ik})$, $v_j = U_V h_j$, and $m_i = \sum_k \alpha_{ik} v_k$. Changing W_{ij} changes only a_{ij} in row i, so

$$\begin{aligned} \frac{\partial \alpha_{ik}}{\partial a_{ij}} &= \alpha_{ik}\big(\mathbf{1}_{\{k=j\}} - \alpha_{ij}\big), \\ \frac{\partial m_i}{\partial a_{ij}} &= \alpha_{ij}(v_j - m_i), \\ \left.\frac{\partial m_i}{\partial W_{ij}}\right|_{h\ \text{fixed}} &= s_{ij}\,\alpha_{ij}\,(v_j - m_i). \end{aligned} \tag{35}$$

With the upstream gradient $b_i = \partial \ell_{\mathrm{diff}}/\partial m_i$, this head contributes $s_{ij}\alpha_{ij}\langle b_i, v_j - m_i\rangle$ to the gradient of W_{ij}. The difference $v_j - m_i$ measures how the message would change, and the inner product with b_i measures how this change affects the action error. The sign also depends on s_{ij}, so a frequently observed edge is not always strengthened.

Summing over layers and heads. To separate local and total derivatives, treat each use of W in layer ℓ and head h as a separate copy $W^{\ell,h}$, and then set all copies equal to W. If $b_i^{\ell,h}$ is the gradient with respect to the message of that head before concatenation, the chain rule gives

$$G_{ij} = \frac{\partial \ell_{\mathrm{diff}}}{\partial W_{ij}} = \sum_{\ell,h} s_{ij}^{\ell,h}\,\alpha_{ij}^{\ell,h}\,\big\langle b_i^{\ell,h},\, v_j^{\ell,h} - m_i^{\ell,h}\big\rangle. \tag{36}$$

Each term holds its own inputs fixed. The effect of earlier layers on later node features is included in the upstream gradients $b_i^{\ell,h}$, which are computed by backpropagation through the output projections, residual connections, normalization, and feed-forward layers. Eq. (36) is therefore the exact total derivative for the deterministic encoder in Appendix 6.6.

From weights to scores. For active rows, $W_{ik} = \exp(B_{ik})/\sum_j \exp(B_{ij})$. Applying the row-softmax Jacobian gives the relation-score gradient:

$$\begin{aligned} \frac{\partial W_{ik}}{\partial B_{ij}} &= W_{ik}\big(\mathbf{1}_{\{k=j\}} - W_{ij}\big), \\ \frac{\partial \ell_{\text{diff}}}{\partial B_{ij}} &= W_{ij}\Big(G_{ij} - \sum_k W_{ik} G_{ik}\Big), \\ \sum_j \frac{\partial \ell_{\text{diff}}}{\partial B_{ij}} &= 0. \end{aligned} \tag{37}$$

The last identity reflects that adding a constant to a row of B does not change W. Training therefore changes only the relative strength of edges from the same anchor. Inactive rows and columns receive no gradient in this forward pass. The same chain rule applies to any other differentiable loss that uses g_t, including the auxiliary loss.

6.8 Policy Network and Parameter Updates

Shared denoiser. The denoiser receives $c_t = [y_t; g_t]$ with $y_t = [v_t; \mathrm{vec}(s_{t-L+1:t})]$ and a sinusoidal embedding of the diffusion step. It is a 1D U-Net with channel widths $(256, 512, 1024)$. Each encoder level has two conditioned residual blocks, the bottleneck has two blocks, and each decoder level has two blocks. Encoder levels save skip features and then downsample with stride-two convolutions; decoder levels upsample with stride-two transposed convolutions and concatenate the skip features. If a decoded length differs from its skip length, nearest-neighbor interpolation matches them. A final conditioned block and a kernel-one convolution output d_a action channels.

FiLM blocks. Each conditioned block applies a convolution, group normalization, and Mish, then FiLM, then a second convolution, group normalization, and Mish, followed by a residual connection. Convolutions use kernel size five, and a kernel-one convolution is used on the residual path when the channel number changes. If F_ℓ is the activation before modulation,

$$\begin{aligned} d_{\tau,t} &= [e_\tau; v_t; \mathrm{vec}(s_{t-L+1:t}); g_t], \\ [\gamma_\ell; \beta_\ell] &= \mathrm{Linear}_\ell(d_{\tau,t}), \\ F'_\ell &= (1+\gamma_\ell) \odot F_\ell + \beta_\ell. \end{aligned} \tag{38}$$

The scale and shift are shared across the action time axis. The step embedding has sinusoidal width 256, followed by Linear $(256 \to 1024)$, Mish, and Linear $(1024 \to 1024)$. All anchors use the same denoiser parameters.

Auxiliary classifier. The classifier maps each proprioceptive vector and the graph context to 256-dimensional tokens, places the graph token after the L proprioceptive tokens, and processes them with a Transformer encoder (four layers, four heads, feed-forward width 512, no dropout). A linear head reads the output at the graph token and produces $K_{\max}$ logits. Inactive entries are masked, and the target is the local index of the selected anchor. This implements R_ψ in Eq. (13). Its output is used only for training; it is never used to select the anchor.

Which parameters are updated. The graph context is not detached at the classifier, so the auxiliary loss trains both the classifier and the graph parameters. Stage 2 losses do not reach the encoder E_η, because the selected index is an integer, and the straight-through value from the query is not used as a graph input. Table 6 summarizes the actual gradient paths. A parameter that appears in an optimizer does not necessarily receive a gradient in every stage.

In Stage 2, querying the memory in training mode still applies the EMA update to occupied prototypes, and the graph context is computed from the updated values. The active set does not change,

Table 6: **Parameter and buffer updates in each stage.** At test time, all parameters and buffers are fixed.

Object	Stage 1	Stage 2
State and time encoders	$\mathcal{L}_{\text{anchor}}$ through soft assignments	No gradient (hard selection); no time input
Projection heads and scales	$\mathcal{L}_v + \mathcal{L}_a$	Kept; not used
Keys, values, statistics	EMA with detached queries; structural operations	EMA with detached queries; fixed active set
Active set	Grow, merge, split	Fixed
Relation scores B	Transition counts	$\mathcal{L}_{\text{diff}} + \lambda_{\text{aux}}\mathcal{L}_{\text{aux}}$
Graph and fusion ϕ	Not used	$\mathcal{L}_{\text{diff}} + \lambda_{\text{aux}}\mathcal{L}_{\text{aux}}$
Denoiser θ	Not used	$\mathcal{L}_{\text{diff}}$
Classifier ψ	Not used	$\lambda_{\text{aux}}\mathcal{L}_{\text{aux}}$

and no new transition counts are added. So "fixed active set" does not mean that all atlas buffers are frozen during training. In evaluation mode, these EMA updates are turned off before any selection or sampling. Each stage uses its own AdamW optimizer, and the gradient norm is clipped to 1.0 (Appendix 7.3).

Algorithm 4 Stage 2: training the graph-conditioned policy

Input: Stage 1 checkpoint; samples $(x_t, v_t, s_{t-L+1:t}, A_t)$; denoiser; classifier; noise schedule; optimizer $\mathcal{O}_2$
Output: Trained policy and final atlas buffers

```
Keep the active set; enable prototype EMA in training mode
for each batch do
    q_t^on ← Norm(E_η(x_t)); z_t ← arg max_{k∈K} (q_t^on)^⊤ μ_k
    Update occupied keys and values with the detached queries
    W ← softmax_row(B_{I,I}); compute g_t by Eqs. (10)–(11)
    c_t ← [v_t; vec(s_{t−L+1:t}); g_t]
    Sample a diffusion step and noise; build X_τ and compute L_diff by Eq. (12)
    Predict anchor logits from (s_{t−L+1:t}, g_t) and compute L_aux with the local target of z_t
    Compute L_policy by Eq. (13); backpropagate, clip the gradient norm, and step O_2
end for
Switch to evaluation mode
return (E_η, K, M, V, B, φ, θ, ψ)
```

6.9 Diffusion Sampling and Closed-Loop Execution

Noise schedule. Let T_d be the number of diffusion steps. In this section we index steps by $k = 1, \ldots, T_d$ and set $\bar{\alpha}_0 = 1$; the code uses indices $k - 1$. We use the cosine schedule

$$
\begin{aligned}
f(r) &= \cos^2\Big(\frac{r + s_0}{1 + s_0} \cdot \frac{\pi}{2}\Big), \qquad s_0 = 0.008, \\
\beta_k &= \mathrm{clip}\Big(1 - \frac{f(k/T_d)}{f((k-1)/T_d)},\, 10^{-4},\, 0.9999\Big), \\
\bar{\alpha}_k &= \prod_{j=1}^{k} (1 - \beta_j).
\end{aligned}
\tag{39}
$$

Training samples k uniformly and uses Eq. (12) (Ho et al., 2020).

Reverse process. For one policy query, z_t, g_t, and c_t stay fixed during sampling. Starting from $X^{(T_d)} \sim \mathcal{N}(0, I_{H\times d_a})$, each step computes

$$\begin{aligned}
\widehat{\epsilon}_k &= \epsilon_\theta\big(X^{(k)}, k-1, c_t\big),\\
\widehat{A}_0^{(k)} &= \mathrm{clip}_{[-1,1]}\left(\frac{X^{(k)} - \sqrt{1-\bar\alpha_k}\,\widehat{\epsilon}_k}{\sqrt{\bar\alpha_k}}\right),\\
\sigma_k^2 &= \frac{1-\bar\alpha_{k-1}}{1-\bar\alpha_k}\beta_k,\\
X^{(k-1)} &= \sqrt{\bar\alpha_{k-1}}\,\widehat{A}_0^{(k)} + \sqrt{\big[1-\bar\alpha_{k-1}-\sigma_k^2\big]_+}\,\widehat{\epsilon}_k + \sigma_k Z_k, \qquad Z_k \sim \mathcal{N}(0, I).
\end{aligned} \tag{40}$$

The time argument $k-1$ matches the index convention of the network. At $k=1$ we have $\sigma_1 = 0$ and $\bar\alpha_0 = 1$, so the output is $X^{(0)} = \widehat{A}_0^{(1)}$ with no final noise. Clipping acts in the normalized action space; the task interface converts actions to robot commands.

Shorter sampling loops. Our sampler also accepts a shorter loop length $S < T_d$. In that case it runs over the stored indices $S-1, \dots, 0$; it does not build a uniformly respaced subset of the full schedule. These two procedures are different, so a shortened run must be reported with its actual step sequence. All reported results use the full schedule unless stated otherwise.

Transition prior and query state. The inference settings in Sec. 3.4 share one trained atlas and denoiser. Stage 2 uses state-matched selection ($\lambda_{\mathrm{tr}} = 0$), and transition-aware evaluation adds preceding-execution context at retrieval time. To construct P^{tr}, freeze the final encoder and prototypes, then re-encode training demonstrations with the same online query $q_t^{\mathrm{on}} = \mathrm{Norm}(E_\eta(x_t))$ used at deployment. If $\mathcal{Q}_n$ contains valid query times in demonstration n, spaced according to the execution prefix H_e and with $t + H_e \le T_n$, the counts in Eq. (8) are

$$\widehat{z}_t^{(n)} = \arg\max_{k\in\mathcal{K}} (q_t^{\mathrm{on},(n)})^\top \mu_k, \qquad q_t^{\mathrm{on},(n)} = \mathrm{Norm}(E_\eta(x_t^{(n)})), \qquad C_{ij}^{(H_e)} = \sum_{n=1}^{N}\sum_{t\in\mathcal{Q}_n} \mathbf{1}_{\{\widehat{z}_t^{(n)}=i,\ \widehat{z}_{t+H_e}^{(n)}=j\}}.$$

Counts include $i=j$ and never cross demonstration boundaries. They use the identities of the final active slots, so the prior and the policy share the same anchor ordering. The smoothed rows in Eq. (8) sum to one; an unobserved row is uniform, and every active candidate has a finite log-prior score. This construction gives P^{tr} the retrieval-side role described in Sec. 3.3, while the action-supervised W supplies relation weights to graph aggregation.

For the first query, a uniform $\bar p_t$ adds the same value to every score and therefore leaves state matching unchanged. Later queries use the previously selected index, after its action prefix has been executed. For candidates i, j, their score difference in Eq. (9) is $(q_t^{\mathrm{on}})^\top(\mu_i - \mu_j) + \lambda_{\mathrm{tr}} \log(\bar p_t(i)/\bar p_t(j))$. Thus, preceding execution can change the ranking when local scores are close. The prior is a score adjustment rather than a hard transition mask. Its strength and smoothing are fixed before testing; setting $\lambda_{\mathrm{tr}} = 0$ recovers the original selection at every query. No future observation, demonstration time, or classifier prediction enters the online choice. Each environment maintains its own preceding index and clears it at reset.

Receding-horizon control. At each query, Eq. (9) selects the anchor before graph context is constructed. The selected index, g_t, and c_t remain fixed throughout the reverse process. The sampler produces H actions, and the first $H_e \le H$ are executed before the index and observation history are passed to the next query. Parameters, the active set, prototypes, and P^{tr} remain fixed during evaluation. The classifier and optional recovery array do not control online selection. Algorithm 5 describes both retrieval settings with one loop.

7 Implementation and Training Details

This section describes our reference implementation, using the state-based RoboMimic NutAssemblySquare setting as the example. The SO-101 setting uses RGB inputs and is described in Appendix 11. Values given here are reference defaults; the settings of each reported run are part of its configuration record.

Algorithm 5 Closed-loop execution with optional transition-aware retrieval

Input: Trained $(E_\eta, \mathcal{K}, M, V, B, \phi, \theta)$; fixed P^{tr} and $\lambda_{\mathrm{tr}} \geq 0$; schedule $\{\beta_k, \bar{\alpha}_k\}_{k=1}^{T_d}$; H, L, H_e; task interface

Set evaluation mode; disable parameter and buffer updates
Initialize observation history and the preceding index $z_{\mathrm{prev}} \leftarrow \perp$
while the episode is not finished **do**
  Read x_t, v_t, and $s_{t-L+1:t}$; $q_t^{\mathrm{on}} \leftarrow \mathrm{Norm}(E_\eta(x_t))$
  if $z_{\mathrm{prev}} = \perp$ **then**
    $\bar{p}_t(k) \leftarrow 1/|\mathcal{K}|$ for all active k
  else
    $\bar{p}_t(k) \leftarrow P^{\mathrm{tr}}_{z_{\mathrm{prev}},k}$ for all active k
  end if
  Select z_t by Eq. (9)
  Compute W and g_t by Eqs. (7), (10), and (11); $c_t \leftarrow [v_t; \mathrm{vec}(s_{t-L+1:t}); g_t]$
  Sample $X^{(T_d)} \sim \mathcal{N}(0, I_{H \times d_a})$
  for $k = T_d, \ldots, 1$ **do**
    Compute $X^{(k-1)}$ by Eq. (40) with z_t, g_t, c_t fixed
  end for
  Execute the first H_e actions of $X^{(0)}$ as commands; stop early if the episode ends
  $z_{\mathrm{prev}} \leftarrow z_t$; update the history with new observations
end while
Clear z_{prev} at the next episode reset

7.1 Architecture, Hyperparameters, and Cost

Table 7: **Network architecture** (reference values, $d = 128$).

Module	Structure
State encoder E_η	Linear $128 \to 256$, LayerNorm, GELU, Linear $256 \to 128$
Time encoder E_u	Linear $1 \to 32$, Tanh, Linear $32 \to 128$; scale $\alpha_u = 0.1$
Projection heads	Linear–GELU–Linear, hidden width $2d_e = 256$, output width $d_e = 128$, then normalization
Graph encoder	2 layers, 4 heads, width 128; shared row-wise transformations; no dropout
Graph feed-forward	Linear $128 \to 256$, GELU, Linear $256 \to 128$; LayerNorm after each residual
Fusion F_ϕ	Linear $256 \to 128$, LayerNorm, GELU; $d_g = 128$
Denoiser ϵ_θ	1D U-Net, channels $(256, 512, 1024)$; two blocks per level and in the bottleneck; kernel size 5
Down/upsampling	Kernel-3 stride-2 convolution; kernel-4 stride-2 transposed convolution; skip concatenation
Step embedding	Sinusoidal (256); Linear $256 \to 1024$, Mish, Linear $1024 \to 1024$
Classifier R_ψ	Width 256; Transformer with 4 layers, 4 heads, feed-forward 512; masked inactive outputs

Hyperparameters. The memory starts with $K_0 = 5$ active anchors in $K_{\max} = 50$ slots. The EMA decay is $\rho = 0.95$, and the dispersion decay is $\rho_D = 0.9$. The assignment temperature is $\tau_q = 1.0$ in training and 0.1 in evaluation; the temperature changes the soft probabilities but not the argmax. The growth and merge thresholds are $\tau_g = 0.65$ and $\tau_m = 0.93$, the split perturbation is $\epsilon_s = 0.1$, and the transition increment is $\eta_B = 0.05$. The loss weights are $\lambda_q = 0.25$, $\lambda_{\mathrm{align}} = 0.1$, and $\lambda_{\mathrm{aux}} = 0.1$. The self-edge weight is $\epsilon_{\mathrm{self}} = 0.1$. The action horizon is $H = 16$, the history length is $L = 4$, and the diffusion model has $T_d = 60$ steps. The split threshold τ_s must satisfy Eq. (28) and is reported per run.

Computational cost. Let K be the number of active anchors, d the width, L_g the number of graph layers, and $C_\theta(H)$ the cost of one denoiser call at horizon H. After the state encoder, anchor selection costs $O(Kd)$, and each graph layer costs $O(Kd^2 + K^2 d)$ for the projections, dense attention,

and node-wise layers. The graph context is computed once per query and reused in all denoising steps. Excluding feature extraction,

$$\begin{aligned} \mathrm{Cost}_{\mathrm{query}} &= O\big(d^2 + Kd + L_g(Kd^2 + K^2 d) + T_d\, C_\theta(H)\big), \\ \mathrm{Memory}_{\mathrm{atlas}} &= O(K_{\max} d + K_{\max}^2), \end{aligned} \tag{41}$$

and each attention layer stores $O(n_h K^2)$ weights. The classifier is not used at test time. During Stage 1, each contrastive loss needs an $m \times m$ similarity matrix, costing $O(m^2 d_e)$ time and $O(m^2)$ memory. A full merge search costs $O(K^2 d)$, and a growth test costs $O(Kd)$ per window. These are algorithmic costs; measured training time and latency depend on the hardware and configuration.

7.2 Data Windows and Inputs

Window construction. Three indices play different roles: the demonstration time u_t used in Stage 1, the diffusion step k, and the optimizer step j. The normalized time u_t is computed over the full demonstration before windows are extracted. With $L = 4$ and $H = 16$, each window has $T_w = L + H = 20$ steps. A demonstration of length T_n gives $\max(T_n - T_w + 1, 0)$ windows. With zero-based index and start s, the loader extracts

$$\begin{aligned} \mathcal{W}_s &= \{s, \ldots, s + L + H - 1\}, \\ t_s &= s + L - 1, \\ S_s^{\mathrm{hist}} &= s_{s:s+L-1}, \\ (x_s^{\mathrm{cond}}, v_s^{\mathrm{cond}}) &= (x_{t_s}, v_{t_s}), \\ A_s &= a_{s+L:s+L+H-1}, \\ u_s^{\mathrm{off}} &= u_{s:s+L+H-1}. \end{aligned} \tag{42}$$

Stage 1 uses all steps in $\mathcal{W}_s$. Stage 2 uses the history, the state and observation at the last history step, and the action slice. Demonstrations are sorted by their IDs, and an optional limit selects the first ones before windows are built. Each epoch shuffles the windows, so the sampling unit is a window and longer demonstrations contribute more samples. The meaning of a stored action index follows the data-collection convention and must be kept when mapping actions to robot commands.

Inputs. Table 8 lists the inputs in the state-based setting. The field called `visual_feat` in this adapter is built from numerical environment states and serves as the observation feature v_t. RGB frames are rendered separately, only for the CLIP probe in Appendix 9.3. All inputs are converted to float32 and padded or truncated to a fixed width; this is separate from the unit-norm projection used in anchor matching.

Table 8: **Inputs in the state-based setting** (n_w is the batch size; the object-state width depends on the dataset).

Input	Content	Shape
s_t	End-effector position and quaternion, two gripper positions	$[n_w, 4, 9]$
v_t	Proprioception, object state, relative end-effector–object position	$[n_w, 256]$
x_t	Joint positions and velocities, object state	$[n_w, 128]$
A_t	Actions in the prediction window	$[n_w, 16, 7]$
u^{off}	Normalized demonstration time in the window	$[n_w, 20]$

7.3 Optimization

Each stage uses AdamW. The gradients of all parameters are clipped jointly to a global norm of at most one before each step. For a parameter ϑ with clipped gradient g_j, learning rate η_j, and weight

decay λ_w,

$$\begin{aligned} m_j &= \beta_1 m_{j-1} + (1-\beta_1) g_j, \\ v_j &= \beta_2 v_{j-1} + (1-\beta_2) g_j^{\odot 2}, \\ \vartheta_j &= (1-\eta_j \lambda_w)\vartheta_{j-1} - \eta_j \frac{m_j/(1-\beta_1^j)}{\sqrt{v_j/(1-\beta_2^j)} + \epsilon_{\text{opt}}}, \\ (\beta_1, \beta_2, \epsilon_{\text{opt}}) &= (0.9,\, 0.999,\, 10^{-8}). \end{aligned} \tag{43}$$

Prototypes are updated by Eq. (6), not by AdamW. B is stored as one parameter tensor for all slots, so its weight decay applies to the whole tensor whenever it receives a gradient. Table 9 lists the settings.

Table 9: **Optimizer hyperparameter settings.** M: batches per epoch; E: number of epochs.

Setting	Stage 1	Stage 2
Learning rate	5×10^{-4}	10^{-4}
Weight decay	10^{-5}	10^{-6}
Gradient clipping (global norm)	1.0	1.0
Batch size	64	64
Epochs	50	50
Cosine schedule period	EM	EM
Scheduler step	every batch	every epoch

With N_w windows and batch size n_w (keeping the last incomplete batch), there are $M = \lceil N_w/n_w \rceil$ batches per epoch and EM updates in total. The learning rate follows

$$\eta(j_s) = \frac{\eta_0}{2}\Big[1 + \cos\Big(\frac{\pi j_s}{EM}\Big)\Big], \tag{44}$$

where j_s counts scheduler calls. In Stage 1 the scheduler is called every batch, and in Stage 2 it is called every epoch, so the two stages use different clocks. Optimizer states are reset between the two stages, while the atlas and its buffers are carried over.

7.4 Schedule of Structural Operations

Structural checks happen at two places (Table 10). Inside the forward pass, after the losses are computed and before backpropagation, growth is tested on up to four windows of the batch, followed by a merge check with probability 0.02 and a split check with probability 0.01. Several growths can be accepted in one batch, but at most one merge and one split. After each optimizer step, the training loop also runs periodic checks: growth on the first window every 50 steps, a merge check every 200 steps, and a split check every 500 steps. The step counter starts at zero, so the first batch is included. A check changes the memory only if its threshold and capacity conditions hold.

Table 10: **Schedule of structural checks.**

Operation	Inside the forward pass	After optimizer steps
Grow	first $\min(n_w, 4)$ windows	first window, every 50 steps
Merge	probability 0.02	every 200 steps
Split	probability 0.01	every 500 steps

Order within one batch. A batch first computes queries, soft probabilities, hard indices, and straight-through values. Then the EMA updates the memory statistics, transitions update B, the losses are computed from the stored forward results, the internal structural checks run, and the optimizer updates the network. Finally, the loop steps the scheduler and runs the periodic checks. The next batch uses the updated memory. In Stage 2 the active set is fixed, but the EMA may still update occupied prototypes before their values enter the graph.

7.5 Runtime Execution

At each query, the selected anchor, graph context, current observation, and proprioceptive history form one fixed denoising condition. The retrieval setting is chosen before evaluation: $\lambda_{\mathrm{tr}} = 0$ uses current-state matching, while $\lambda_{\mathrm{tr}} > 0$ also uses the preceding index and the fixed prior. The index is maintained separately for each environment, updated after executing an action prefix, and cleared at episode reset (Sec. 6.9). During training, one diffusion step is sampled per action chunk and shared across its action dimensions. Inference starts from Gaussian noise and runs the reverse process over $T_d = 60$ steps.

In the robosuite driver, the length-four history is initialized by repeating the first proprioceptive observation, and later observations enter a rolling buffer. The features x_t and v_t use the latest state. The policy predicts 16 actions and, by default, executes one ($H_e = 1$). Optional scaling and smoothing are applied to the whole action vector, including the gripper:

$$\begin{aligned} \widetilde{a}_t &= c_a \, a_t^{\mathrm{gen}}, \\ a_t^{\mathrm{exec}} &= \lambda_e \, a_{t-1}^{\mathrm{exec}} + (1 - \lambda_e) \, \widetilde{a}_t, \\ (c_a, \lambda_e) &= (1, 0) \quad \text{by default.} \end{aligned} \tag{45}$$

The simulator runs at 20 Hz, which sets the simulated time per action; network latency is a separate quantity. The driver switches to evaluation mode and disables gradients before querying, so parameters and buffers stay fixed during a rollout. The environment adapter converts actions to robot commands.

7.6 Checkpoints and Reproducibility

A saved model contains the network parameters and the persistent buffers: the active mask, keys and values, usage and dispersion statistics, relation scores, and the diffusion schedule. Together they determine the selected anchors and the policy condition. Stage 2 loads this state before training. A full training checkpoint also stores the optimizer and scheduler states and the current epoch; evaluation uses only the model state. The active set is part of the checkpoint, while the counts of grow, merge, and split events are recorded in the training log.

A transition-aware evaluation additionally stores P^{tr} with the final active-slot order, H_e, α_{tr}, and λ_{tr}. These settings identify the retrieval configuration without changing the trained denoiser. The preceding index is episode-local execution state rather than a trained parameter; it is initialized at each reset (Sec. 6.9).

Randomness comes from window order, parameter initialization, structural checks and splits, training noise, sampling noise, and environment initialization. To reproduce a rollout, one needs the checkpoint, the input and action interfaces, the environment seed, the diffusion step sequence, the execution length, and the action postprocessing.

8 Experimental Protocol

This section explains how the results in Section 4 are computed. Benchmark results, ablations, anchor analyses, and real-robot trials are separate experiments with their own task sets and averaging rules.

8.1 Benchmarks and Averaging

We report LIBERO (Liu et al., 2023), RoboTwin 2.0 (Chen et al., 2025), RoboMimic (Mandlekar et al., 2021), and D3IL (Jia et al., 2024). Table 1 is the three-seed cross-benchmark summary for DP-based methods, while Table 2 is the dedicated four-seed LIBERO repeated-run evaluation. The LIBERO score is the equal-weight mean of its four suites, and the Macro-Avg. in Table 1 is the equal-weight mean of the four benchmark-level scores. TF-DP is reported only at the benchmark level. The five selected LIBERO tasks (Appendix 8.4) form a separate task set, and the ablation uses three benchmarks. Table 11 lists the averaging domains.

Table 11: **Averaging domains.** Each average gives equal weight to the items in its domain.

Summary	Items with equal weight	Table
Macro-Avg.	Four simulation benchmarks	1
LIBERO Mean	L-Spatial, L-Object, L-Goal, L-Long	2
Ablation Avg.	RoboMimic, RoboTwin 2.0, D3IL	3
Selected-task Avg.	Mug, Bowl, Crm, Moka, MW	13
SO-101 Mean	Four real-world tasks	4

For method m on task d with n_d trials and binary outcomes Y_{mdr},

$$\begin{aligned} \widehat{p}_{md} &= \frac{100}{n_d} \sum_{r=1}^{n_d} Y_{mdr}, \\ \widehat{p}_{m,\mathcal{T}} &= \frac{1}{|\mathcal{T}|} \sum_{d \in \mathcal{T}} \widehat{p}_{md}, \\ \Delta_{m,b,\mathcal{T}} &= \widehat{p}_{m,\mathcal{T}} - \widehat{p}_{b,\mathcal{T}}. \end{aligned} \tag{46}$$

Differences are in percentage points. "Best baseline" means the baseline with the highest average, not a different baseline for each task. All values are rounded to one decimal place after averaging, so two equal rounded values may come from slightly different unrounded ones.

8.2 Baselines and Ablation Variants

DP (Chi et al., 2025) is the base diffusion policy. AutoCGP+DP (Zhou et al., 2025) adds concept conditioning, and HiMaCon+DP (Liu et al., 2026) adds concept-prediction supervision. TF-DP (Hu et al., 2026) conditions on execution traces, and ACT (Zhao et al., 2023) is a Transformer-based action-chunking policy. These are full-policy comparisons. The ablation variants instead change one part of ActAtlas (Table 12).

Table 12: **Definitions of the ablation variants.**

Variant	Change
FixedAtlas-K	Uses a fixed number of anchors; no growth, merging, or splitting. Prototype learning and graph conditioning are kept.
Anchor-only	Uses only the value of the selected anchor; no graph aggregation.
Shuffled Graph	Keeps the graph encoder but breaks the match between each anchor and its successors in the relation weights.
Concat-g	Concatenates g_t to the denoiser input instead of using FiLM.
LongHist-DP	DP with a longer observation history.
LongHist-AutoCGP	AutoCGP+DP with a longer observation history.

For FixedAtlas-K, the result depends on the chosen K together with the rest of the training setup; this K is different from $K_{\max}$ and from the number of indices shown in Figure 2. For Shuffled Graph, the intervention is defined by how the indices are permuted and at which stage the permutation is applied. For the LongHist variants, a longer input history is different from a longer action horizon or a longer execution length.

8.3 Controlled Conditioning and Retrieval on LIBERO-Long

Table 3(b) isolates the effects of relational conditioning and transition-aware retrieval while retaining the same Stage 1 atlas. The graph-context branch uses

$$g_t^{\text{graph}} = F_\phi([\nu_{z_t}; \widetilde{\nu}_{z_t}]) \,,$$

where $\widetilde{\nu}_{z_t}$ is the graph-enhanced value defined in Eq. (11). The per-anchor embedding branch replaces the graph-enhanced value with an independently learned anchor embedding:

$$g_t^{\text{emb}} = F_\xi([\nu_{z_t}; M_\xi(e_{z_t})]) \,,$$

where $e_k \in \mathbb{R}^{d_e}$ is trainable and M_ξ is a shared multilayer perceptron. Both conditioning vectors have width d_g.

Within each training seed, the two branches share the same Stage 1 atlas, encoder, active anchor set, data order, and prototype updates. Their denoisers are trained separately with the same architecture, FiLM interface, action horizon, execution prefix, and training losses, with matched conditioning-module parameter budgets.

Both checkpoints are trained using state-matched retrieval, $\lambda_{\mathrm{tr}} = 0$. Each frozen checkpoint is then evaluated under both state-matched and transition-aware retrieval on the same initial states for all ten LIBERO-Long tasks. This protocol isolates graph conditioning under state-matched retrieval and then measures the additional retrieval gain from preceding-execution context on the same frozen policy. The transition-aware evaluations use the same P^{tr}, smoothing configuration, and fixed transition-prior strength.

8.4 Selected LIBERO Tasks

Table 13 compares DP and ActAtlas on five LIBERO tasks: put the white mug on the plate (Mug), put the black bowl on the plate (Bowl), put the cream cheese box in the basket (Crm), put the moka pot on the stove (Moka), and open the microwave (MW). ActAtlas improves four tasks and is worse on Crm. The average gain is small (+1.4), in contrast with the large gain on L-Long (+16.2). This suggests that ActAtlas helps most when a task contains several connected operations. Because the two task sets also differ in objects, goals, and interactions, this comparison does not isolate task length alone; the ablations provide more direct evidence about the components.

Table 13: **Selected long-horizon LIBERO tasks.** Success rates (%) with standard errors. Δ: ActAtlas minus DP.

Task	DP	ActAtlas	Δ
Mug	88.6 ± 0.3	$\mathbf{92.3} \pm 0.2$	$+3.7$
Bowl	87.3 ± 0.5	$\mathbf{88.2} \pm 0.4$	$+0.9$
Crm	$\mathbf{76.4} \pm 1.1$	70.2 ± 0.7	-6.2
Moka	82.9 ± 0.7	$\mathbf{88.4} \pm 0.3$	$+5.5$
MW	90.2 ± 0.6	$\mathbf{93.4} \pm 0.6$	$+3.2$
Avg.	85.1	**86.5**	$+1.4$

8.5 Standard Errors

For tables reporting mean $\pm$ SE across training seeds, one independently trained seed is treated as the statistical unit. Within each seed, evaluation trials are first averaged within each task; when an entry contains multiple tasks, the task-level success rates are then averaged equally to obtain one seed-level score. Let q_{ms} denote the resulting score of method m for training seed s, and let S be the number of independent training seeds. We compute

$$\begin{aligned} \bar{q}_m &= \frac{1}{S}\sum_{s=1}^{S} q_{ms}, \\ \mathrm{SE}(\bar{q}_m) &= \sqrt{\frac{\sum_{s=1}^{S}(q_{ms}-\bar{q}_m)^2}{S(S-1)}}. \end{aligned} \tag{47}$$

Tables using three training seeds therefore use $S = 3$, while the repeated-seed LIBERO evaluations use $S = 4$. Averages reported without uncertainty are point estimates obtained from the corresponding task or benchmark averages. The real-robot results use Wilson intervals instead (Appendix 11.3).

9 Anchor Analysis Details

Section 4.3 uses three data: the anchor selected during execution, the transitions between selected anchors, and images associated with each anchor in a successful rollout. This section describes how

each is collected and computed. The control value of the anchors is tested separately by the ablations in Section 4.2.

9.1 Rollout Collections and Index Conventions

We record the selected anchor index during successful and failed rollouts on NutAssemblySquare. The collection script searches over episode initializations and keeps traces from both outcome groups, so the figure shows both completed and failed executions. Success rates are computed from the evaluation trials, while the temporal panels only summarize the kept traces. In Figure 2, faint lines are successful rollouts and dark red lines are failed ones. Each trace ends at its own episode boundary.

We distinguish four quantities: the number of slots $K_{\max}$, the number of active anchors $|\mathcal{K}|$, the indices visited in a collection, and the size K_{diag} of the displayed matrix. The plotted index ranges are 0–26 for ActAtlas and 0–9 for the AutoCGP-style model. Indices are category labels: they do not imply an order in time or a distance between anchors. The AutoCGP-style model is our local implementation with a fixed codebook, and it differs from the full AutoCGP+DP baseline in the performance tables.

If several actions are executed per query, the same index is recorded for each executed step. The recording interval and the execution length therefore affect how often the index stays the same. These settings are kept with each collection. The CLIP probe uses its own successful rollout and does not provide synchronized stage labels for the temporal panels.

9.2 Transition Statistics

Let $z_t^{(n)}$ be the selected anchor at step t of rollout n. We count transitions within each rollout, including self-transitions:

$$\begin{aligned} N_{ij} &= \sum_n \sum_{t=1}^{T_n-1} \mathbf{1}\big[z_t^{(n)} = i,\ z_{t+1}^{(n)} = j\big], \\ n_i &= \sum_j N_{ij}, \\ M_{\mathrm{tr}} &= \sum_i n_i. \end{aligned} \tag{48}$$

Observed rows are normalized as $\widehat{P}_{ij} = N_{ij}/n_i$, and unobserved rows are set to zero. With occupancy weights $w_i = n_i/M_{\mathrm{tr}}$ and $0 \log 0 = 0$, we report

$$\begin{aligned} D_{\mathrm{diag}} &= \sum_i w_i \widehat{P}_{ii}, \\ R_{\mathrm{nz}} &= \frac{\#\{(i,j) : \widehat{P}_{ij} > 0\}}{K_{\mathrm{diag}}^2}, \\ H_{\mathrm{row}} &= -\sum_i w_i \sum_j \widehat{P}_{ij} \log \widehat{P}_{ij}. \end{aligned} \tag{49}$$

Substituting the row normalization gives simple forms:

$$\begin{aligned} D_{\mathrm{diag}} &= \frac{\sum_i N_{ii}}{M_{\mathrm{tr}}}, \\ 1 - D_{\mathrm{diag}} &= \frac{1}{M_{\mathrm{tr}}} \sum_n \sum_{t=1}^{T_n-1} \mathbf{1}\big[z_t^{(n)} \neq z_{t+1}^{(n)}\big], \\ 0 \le H_{\mathrm{row}} &\le \log K_{\mathrm{diag}}. \end{aligned} \tag{50}$$

So D_{diag} is the fraction of consecutive steps that keep the same anchor, and $1 - D_{\mathrm{diag}}$ is the fraction that switch. The entropy bound follows from the bound for each categorical row and $\sum_i w_i = 1$; we use natural logarithms (nats). Adding unused rows changes the denominator of R_{nz}, and a different number of anchors changes all three statistics. We therefore use them only to describe each collection. Table 14 gives the values reported in Section 4.3.

Table 14: **Transition statistics on NutAssemblySquare**, computed on the selected rollouts at each model's own granularity.

Metric	ActAtlas	AutoCGP-style
D_{diag} (↑ more persistent)	0.934	0.738
R_{nz} (↓ sparser)	0.069	0.640
H_{row} (↓ more concentrated)	0.404	1.082

Learned relations versus observed transitions. The relation weights $W = \mathrm{softmax}_{\mathrm{row}}(B)$ in Eq. (7) start from Stage 1 inter-anchor transition evidence and are refined by Stage 2 action-supervised gradients. The rollout matrix $\widehat{P}$ records the executed anchor dynamics, including self-transitions, while P^{tr} in Eq. (8) provides the retrieval prior constructed from the final online retriever. Together, W supplies policy-conditioning relations, P^{tr} supplies preceding-execution retrieval context, and $\widehat{P}$ characterizes the temporal organization observed during execution. Table 14 summarizes this behavioral organization, and the Anchor-only and Shuffled Graph ablations test the control contribution of relational context and anchor–relation correspondence.

9.3 CLIP PROBE

Setup. We use frozen OpenCLIP ViT-B-32 image and text encoders with OpenAI weights (Radford et al., 2021), together with their image preprocessing and tokenizer, in evaluation mode. For each anchor, the script goes through its execution steps in time order, keeps frames at least six steps apart, and selects up to 24 frames. Up to 16 frequently used anchors are kept. These are the default extraction settings; the figure shows a subset of the groups.

Scores. For the image set $\mathcal{J}_k$ of anchor k, let $u_I = \mathrm{Norm}(E_{\mathrm{img}}(I))$ and $v_r = \mathrm{Norm}(E_{\mathrm{text}}(p_r))$ for the $R = 12$ prompts in Table 15. For a nonzero mean feature,

$$\begin{aligned} \bar{u}_k &= \mathrm{Norm}\Big(\frac{1}{|\mathcal{J}_k|}\sum_{I\in\mathcal{J}_k} u_I\Big), \\ S_{kr} &= \bar{u}_k^{\top} v_r, \\ r_k^{\star} &= \arg\max_r S_{kr}. \end{aligned} \tag{51}$$

In general, S_{kr} is not the mean of the individual image–text scores, because the mean feature is normalized again. The rows compare mean image features with text, not prototype vectors with text. The prompts are used only after execution, and the heatmap shows raw cosine scores.

Table 15: **Text prompts used in the CLIP probe**, in the order used by the script. IDs match Figure 2.

ID	Prompt	ID	Prompt
1	reaching towards the nut	7	hovering above the peg
2	grasping the nut	8	regrasping the nut
3	lifting the nut	9	recovering from failure
4	moving the nut towards the peg	A	opening the gripper
5	aligning the nut with the peg	B	closing the gripper
6	inserting the nut onto the peg	C	fine adjustment / small corrections

Findings. Anchors 0, 10, 4, and 16 share the top prompt "inserting the nut onto the peg," with scores 0.256, 0.252, 0.253, and 0.248. In the successful rollout used for the probe, Anchor 0 is selected in 607 of 747 recorded steps (81.3%). Their shared coarse semantic description coexists with different anchor states and graph contexts, showing how the atlas retains finer-grained execution variation within a common task semantic. We read this probe together with the temporal and control ablations.

9.4 Prompt Margin Bound

The following bound describes how clearly the top prompt stands out among the fixed prompts in Table 15. Let $r_k^\star$ be a top prompt and $\tau_p > 0$ a temperature. Define the margin and the normalized scores

$$\Delta_k = S_{k,r_k^\star} - \max_{r \neq r_k^\star} S_{kr}, \qquad p_k(r) = \frac{\exp(S_{kr}/\tau_p)}{\sum_{j=1}^{R} \exp(S_{kj}/\tau_p)}. \tag{52}$$

Proposition 9.1 (Prompt margin). *For finite scores, $R \geq 2$, and $\tau_p > 0$,*

$$p_k(r_k^\star) \geq \frac{1}{1 + (R-1)\exp(-\Delta_k/\tau_p)}. \tag{53}$$

Proof. Divide the numerator and the denominator by $\exp(S_{k,r_k^\star}/\tau_p)$. Every other prompt is at least Δ_k below the top score, so

$$p_k(r_k^\star) = \Big[1 + \sum_{r \neq r_k^\star} \exp\Big(\frac{S_{kr} - S_{k,r_k^\star}}{\tau_p}\Big)\Big]^{-1} \geq \big[1 + (R-1)\exp(-\Delta_k/\tau_p)\big]^{-1}. \tag{54}$$

Equality holds when every other prompt is exactly Δ_k below the top score. Ties give $\Delta_k = 0$. □

This is a statement about relative scores within a fixed prompt set. It does not measure whether a description is semantically correct, and it does not guarantee that task stages are recovered. The heatmap shows raw scores, and its maximum alone does not determine Δ_k, because the other scores also matter.

10 Computational Overhead

This section reports the extra computation that ActAtlas adds on top of a standard diffusion policy (DP). The denoiser, the diffusion schedule, and the sampling procedure are shared with DP, so the overhead comes only from the atlas modules: the state encoder, the graph encoder, the fusion layer, and the relation scores.

Extra parameters. Table 16 lists the parameters of the atlas modules, computed from the architecture in Table 7 ($d = 128$, $K_{\max} = 50$). The modules used at test time add about 0.37M parameters. The projection heads and the auxiliary classifier are used only during training. The FiLM layers of the denoiser also receive the $d_g = 128$ extra condition dimensions of g_t.

Table 16: **Parameters of the atlas modules** ($d = 128$, $K_{\max} = 50$).

Module	Used at	Parameters
State encoder E_η	training and test	66,432
Graph encoder (2 layers)	training and test	264,192
Fusion layer F_ϕ	training and test	33,152
Relation scores B	training and test	2,500
Total at test time		**366,276**
Time encoder E_u	Stage 1 only	4,288
Projection heads f_c, f_v, f_a	Stage 1 only	226,432
Auxiliary classifier R_ψ	Stage 2 only	$\approx$ 2.16M

Cost per policy query. Let $K \leq K_{\max}$ be the number of active anchors. Selecting the anchor costs $O(Kd)$ after the state encoder, and each graph layer costs $O(Kd^2 + K^2 d)$. With $K = 50$ and $d = 128$, the two graph layers need about 1.4×10^7 multiply-accumulate operations in total. This cost is paid *once per query*: the context g_t is then reused in all $T_d = 60$ denoising steps, while the U-Net denoiser, which DP also uses, is evaluated T_d times. The auxiliary classifier is not used at test time. The atlas needs $O(K_{\max} d + K_{\max}^2)$ memory for the keys, values, and relation scores, which is $12{,}800 + 2{,}500$ numbers in our setting.

Training cost. Stage 1 adds the contrastive losses, which compare every pair of time steps in a batch. With $n_w = 64$ windows of length $T_w = 20$, a batch has $m = 1{,}280$ steps, so each contrastive loss builds a $1{,}280 \times 1{,}280$ similarity matrix. The structural operations (growth, merging, and splitting) run without gradients and cost at most $O(K^2 d)$ for a merge search. Stage 2 has the same training loop as DP, plus one graph forward pass and the auxiliary classifier per batch.

11 Real-World Details

The real-robot platform is a single-arm SO-101. Policies use RGB images and proprioception and output joint and gripper position commands. Camera processing, task layout, and command conversion belong to this hardware interface and differ from the state-based simulation adapter in Appendix 7.2.

11.1 Task Objectives and Success Criterion

Figure 3 shows the four tabletop tasks, and Table 17 states the goal of each. ACT, DP, AutoCGP+DP, and ActAtlas use the same demonstrations, data splits, inputs, and test conditions. The DP-based methods share the denoiser and the prediction, sampling, and execution settings, and keep their own representation modules; ACT keeps its own architecture.

Table 17: **Goals of the real-world tasks.** All tasks also require holding the final state for two seconds.

Task	Goal
Pick and Place	Grasp the cup, place it inside the target container, and release it.
Pour the Water	Pour water from the source container into the target container and complete the pouring motion.
Insertion	Align the rod and insert it into the socket to the target depth.
Hinge Tasks	Move the hinged lid from its initial position to the target position.

Success criterion. Let $\mathcal{G}_d$ be the set of goal configurations for task d. A trial succeeds if the robot reaches $\mathcal{G}_d$ within the time limit and keeps the goal condition for two seconds. Intermediate events, such as grasping the cup or touching the socket, do not count as success. The image sequences in Figure 3 only illustrate execution; all rates are computed over the full set of evaluation trials.

11.2 Success Counts

Each method is run for 50 trials per task. Table 18 lists the raw counts behind Table 4. One trial changes a task success rate by 2 points. Because all tasks have the same number of trials, the four-task mean equals the total number of successes divided by 200.

Table 18: **Success counts on SO-101** (out of 50 per task).

Method	Pick	Pour	Insert	Hinge	Total	Mean (%)
ACT	8	6	14	7	35/200	17.5
DP	15	9	11	13	48/200	24.0
AutoCGP+DP	21	16	17	21	75/200	37.5
ActAtlas	27	33	29	36	125/200	62.5

The best baseline mean is 37.5% (AutoCGP+DP), so ActAtlas is 25.0 points higher. The per-task differences from AutoCGP+DP are 12.0, 34.0, 24.0, and 30.0 points, in the order of the table.

11.3 Wilson Confidence Intervals

For a task with s successes in $n = 50$ trials, let $\widehat{p} = s/n$. We treat the trials of one task as independent Bernoulli draws. The 95% Wilson interval contains all p that satisfy the score inequality, with

$z = 1.9599639845$:

$$n(\widehat{p} - p)^2 \le z^2 p(1-p) \quad \Longleftrightarrow \quad (n + z^2)p^2 - (2n\widehat{p} + z^2)p + n\widehat{p}^2 \le 0. \tag{55}$$

The two roots of this quadratic give the interval $[c - h,\ c + h]$, with

$$\begin{aligned} c &= \frac{\widehat{p} + z^2/(2n)}{1 + z^2/n}, \\ h &= \frac{z\sqrt{\widehat{p}(1-\widehat{p})/n + z^2/(4n^2)}}{1 + z^2/n}. \end{aligned} \tag{56}$$

Multiplying by 100 gives percentages. For example, 33/50 successes on Pour the Water give 66.0% with interval $[52.2, 77.6]\%$, and 36/50 on Hinge Tasks give 72.0% with interval $[58.3, 82.5]\%$. All intervals in Table 4 use this formula, and only the final endpoints are rounded.

These intervals describe the uncertainty from a finite number of trials for one trained policy, not the variation across independently trained models. Because the four tasks have different success probabilities, we report the mean as a point estimate and do not average the intervals. Overlap between two intervals is not a test of the difference between methods; such a test would need to account for the actual comparison design, such as matched initial layouts.