

Graph-Reweight Behavior Cloning with Human Interventions for Precise Robotic Manipulation:

Appendix

Anonymous Author(s)

Affiliation

Address

email

A Implementation Details and Notation Clarifications

This appendix provides the implementation and evaluation details that are omitted from the main paper due to space. The goal is to make the real-robot experiments reproducible, clarify the data used by each component, and make explicit which information is and is not used when computing graph-retrieved progress (GRP). Unless otherwise stated, all policy-learning experiments use the cumulative offline dataset

$$\mathcal{D}^r = \mathcal{D}_{\text{demo}} \cup \mathcal{D}_{\text{rollout}}^{1:r}, \quad \mathcal{D}_{\text{rollout}}^{1:r} = \mathcal{D}_{\text{success-rollout}}^{1:r} \cup \mathcal{D}_{\text{fail-rollout}}^{1:r},$$

where rollouts may include human interventions. The graph is built over observed states, while the VLA objective remains a supervised action-chunk prediction loss.

A.1 Real-World Setup and Task Details

Our real-world manipulation platform consists of a 7-DoF Flexiv Rizon 4s arm equipped with a Robotiq 2F-85 parallel gripper, a head-mounted RealSense L515 camera, two wrist-mounted RealSense D405 cameras, one side RealSense D405 camera, and a SpaceMouse used for shared-control interventions. The workspace and camera arrangement are shown in Figure 1. Although the hardware rig contains four RGB-D cameras, each task uses a fixed subset of three RGB views during data collection and training. This design adapts the observation geometry to the visual requirements of each precise manipulation task without changing the policy interface. Across all tasks, the action space is 8-dimensional, and the robot control loop runs at 30 Hz during both data collection and inference. All real-robot policy deployment and online inference runs are executed on a single NVIDIA GeForce RTX 5090 GPU. We do not apply focal-length scaling in the reported task configurations.

We evaluate GR-BC on four long-horizon manipulation tasks. Each task is judged by terminal physical success rather than dense progress labels. A rollout may therefore be labeled as failed even when it contains recoverable local segments, and a successful demonstration may still contain hesitations, small contact corrections, or local regressions.

Submitted to the 10th Conference on Robot Learning (CoRL 2026). Do not distribute.

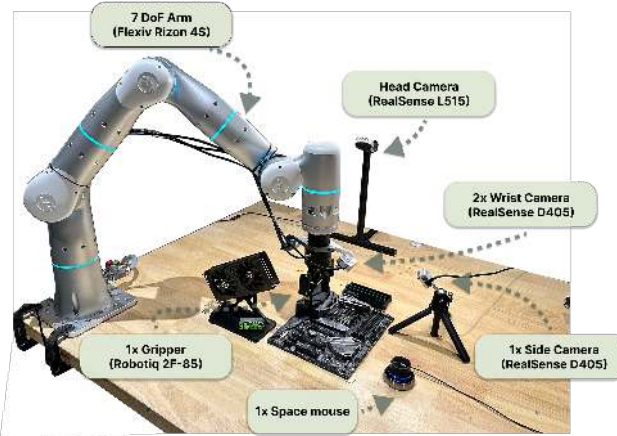


Figure 1: Real-world robot workspace used for GR-BC data collection and evaluation.

Assembly Model. The robot grasps a black four-hole assembly part and inserts it into a white four-corner socket model. The task has broad generalization because the part can start from a large tabletop region. It admits two valid success modes, where the robot grasps the part from either the left or right side. These modes form multiple successful terminal clusters in the reachability graph; Appendix D.3 provides the corresponding analysis. Success requires full insertion followed by gripper opening.

Graphics Card Assembly. The robot inserts a graphics card into the target PCIe slot. This task combines large-amplitude arm motion with precise connector alignment: the graphics card is relatively heavy, while the bottom latch and gold-finger connector must be inserted accurately. We use an RTX 4060 graphics-card mockup for this task. Success requires the card to be fully inserted and the gripper to open.

Table 1: Configuration for Assembly Model.

Field	Setting
Task instruction	Pick up the black connector and insert it into the white socket.
Camera configuration	Head, left wrist, right wrist; no focal-length scaling
Initial demonstrations	215
Mean episode length	376
Action chunk horizon N	30 frames
Graph stride	$k = 15$ frames

Table 2: Configuration for Graphics Card.

Field	Setting
Task instruction	Pick up the graphics card and insert it into the motherboard PCIe slot.
Camera configuration	Head, left wrist, side; no focal-length scaling
Initial demonstrations	125
Mean episode length	911
Action chunk horizon N	30 frames
Graph stride	$k = 15$ frames

Screw Tightening. The robot uses a screwdriver to tighten two screws sequentially. This is a long-horizon and contact-rich task: before rotation, the policy must localize the small screw head and align the tool tip accurately. The task uses M2 screw heads. Success requires each of the two screws to be tightened by 90 degrees.

Memory Stick Insertion. The robot picks up a thin memory stick from a box and inserts it into the target socket. The task requires generalization over randomized object poses and demands finer connector alignment than the graphics-card assembly task. We use a DDR4 memory stick, whose socket opening is approximately 1 mm wide. Success requires the memory stick to be fully inserted and the slot latch to lock.

Table 3: Configuration for Screw Tightening.

Field	Setting
Task instruction	Pick up the screwdriver, tighten the two screws one after the other.
Camera configuration	Head, right wrist, side; no focal-length scaling
Initial demonstrations	100
Mean episode length	1,517
Action chunk horizon N	30 frames
Graph stride	$k = 15$ frames

Table 4: Configuration for Memory Stick.

Field	Setting
Task instruction	Pick up the RAM and insert it into the motherboard RAM slot.
Camera configuration	Left wrist, right wrist, side; no focal-length scaling
Initial demonstrations	150
Mean episode length	1,008
Action chunk horizon N	30 frames
Graph stride	$k = 15$ frames

Rollouts are collected with SpaceMouse-based shared control. During policy execution, any nonzero SpaceMouse input triggers human takeover. Control is returned to the policy after a short fixed timeout with no SpaceMouse input. The resulting trajectories interleave autonomous execution, failure-prone behavior, and corrective segments, making them useful for offline data analysis, policy post-training, and reward-model evaluation pipelines with different input requirements.

A.2 Dataset Composition

The initial dataset contains expert demonstrations. In each offline iteration, the current policy is deployed to collect additional real-world rollouts, which are retained regardless of final outcome. Table 5 summarizes the number of demonstration trajectories, successful and failed rollouts, and the mean number of rollouts collected per offline round for each task.

A.3 Graph Construction Details

For each task, we construct a task-conditioned reachability graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ over the cumulative offline dataset. Nodes are subsampled trajectory states, with all episode terminal states and all

Table 5: Aggregate dataset statistics.

Statistic	Assembly Model	Graphics Card	Screw Tightening	Memory Stick
Demo trajectories	215	125	100	150
Successful rollouts	85	270	160	240
Failed rollouts	100	120	80	80
Mean rollouts per round	90	120	80	100

successful terminal states retained. Each node stores synchronized multi-view visual embeddings, proprioception, and the binary gripper state. The visual encoder is the manipulation-centric representation (MCR) model [1]; we compare this choice against R3M [2] and V-JEPA2.1 variants in Appendix D.2.

Temporal edges encode physically executed transitions within a trajectory. Similarity edges bridge states from different trajectories when they pass multi-view mutual-nearest-neighbor retrieval, gripper-state consistency, and adaptive edge-length thresholding. Proprioception is stored for diagnostics and policy input but is not used in similarity retrieval because the multi-view MCR features plus gripper-state filtering provided the most stable retrieval interface in our scans. Accepted similarity pairs are implemented as two directed edges with identical cost so that reverse shortest-path search can be run on a directed graph. Similarity edges are not interpreted as executable transitions; they only identify visually/proprioceptively compatible states for potential estimation. We connect all successful terminal nodes to the virtual sink g_* , then run reverse Dijkstra search [3] to compute $R_{\text{rec}}(v)$, the minimum graph cost from v to any successful terminal state. Unless otherwise specified, we use the 95th percentile of finite R_{rec} as R_{max} . When converting sparse node progress to dense frame-level progress, we interpolate only finite node pairs; otherwise progress is set to 0.

Figure 2 illustrates the graph construction process on the Assembly Model task using two representative trajectories. The left panel shows temporally connected trajectory nodes, cross-trajectory similarity links, the virtual sink g_* , and the retrieved remaining-step values $R_{\text{rec}}(v)$. Here R denotes the graph-retrieved remaining-step estimate from the current node to a successful terminal node, so smaller values indicate states closer to success and $R = 0$ corresponds to a successful terminal node or the virtual success sink. The right panel visualizes matched image states used to verify the local visual consistency of the accepted similarity edges; d denotes the MCR feature-space distance between the two matched camera-image states, where smaller values indicate stronger visual or representational similarity.

Figure 3 visualizes the resulting graphs built from the demonstration data and one rollout round across all four tasks.

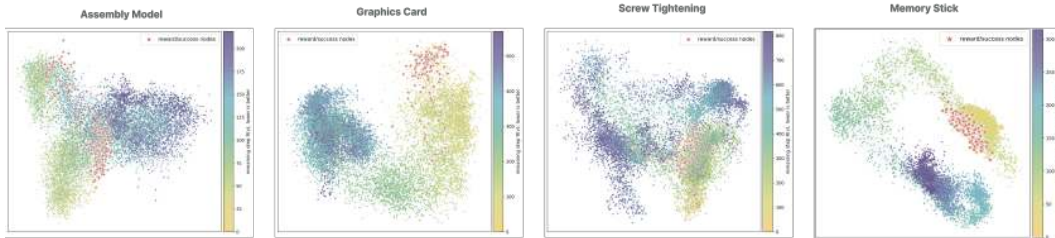


Figure 3: Graph overview constructed from the demonstration data and one rollout round for all four tasks. Darker node colors indicate larger remaining-step estimates.

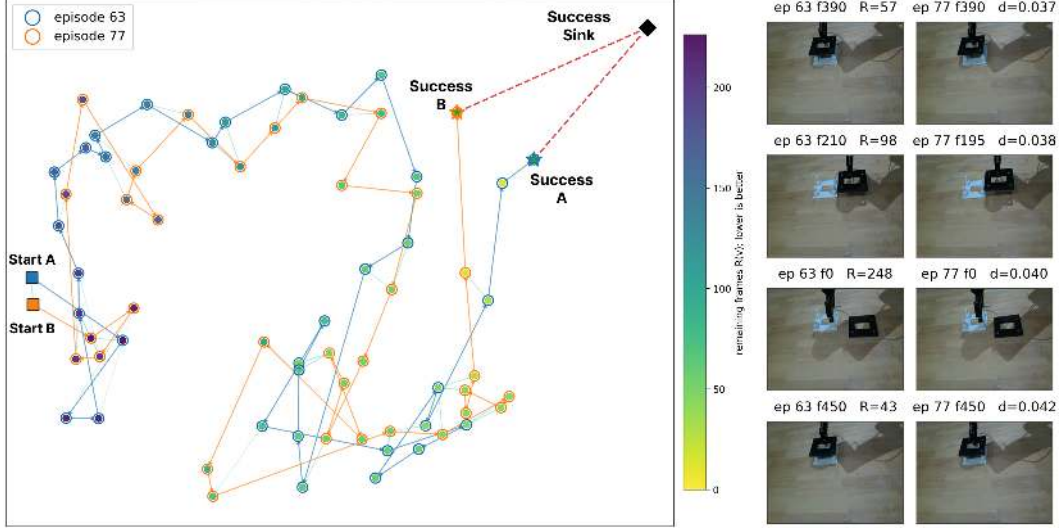


Figure 2: Two-trajectory graph construction example on the Assembly Model task. Blue and orange nodes denote two different trajectories, solid edges denote temporal transitions, and dashed red edges denote zero-cost edges from successful terminal nodes to g_* . Node colors indicate retrieved remaining steps $R_{\text{rec}}(v)$, where lower values correspond to states closer to task completion. The right image pairs show representative matched states; R is the graph-retrieved remaining-step estimate and d is the MCR feature-space distance between the matched states.

102 B Progress Evaluation Protocol and Additional Progress Results

103 This section provides the progress-evaluation protocol used to compare GRP with learned or
 104 foundation-model-based progress and reward baselines, including ReWiND [4], SARM [5], Robo-
 105 Dopamine [6], and RoboMeter [7]. These methods differ in their intended supervision and data
 106 interfaces; therefore, we report both interface assumptions and fixed evaluation metrics instead of
 107 forcing all methods into an artificial identical training protocol.

108 B.1 Baseline Implementation and Supervision

109 The progress baselines are evaluated under their intended supervision settings. ReWiND uses
 110 language-guided reward estimation [4]; SARM learns stage-aware rewards from stage or progress
 111 supervision [5]; Robo-Dopamine and RoboMeter provide general-purpose process or trajectory-
 112 comparison reward signals [6, 7]. These learned reward baselines are therefore not supervision-
 113 matched to GRP: they may use progress labels, segment comparisons, or failure-derived examples
 114 when required by their own training interfaces. GRP instead uses terminal success information and
 115 graph connectivity, without dense progress labels, subtask labels, or trajectory comparisons.

116 Across the four real-robot tasks, we use the same evaluation protocol and keep each learned progress
 117 baseline as close as possible to its intended training recipe. All progress pipelines in the baseline
 118 comparison are instantiated separately for each task, not only for Screw Tightening. For each task,
 119 progress metrics are computed on a fixed evaluation set with 30 successful rollout episodes and
 120 30 failed rollout episodes from that task, and the reported baseline comparisons summarize the
 121 four-task evaluation rather than a single Screw Tightening run. In the main-paper progress table,
 122 Class. SE and Class. FE are averaged over the four tasks and reported out of 30 episodes per task.
 123 For GRP, this primary progress table is a *transductive graph diagnostic*: the evaluation trajectories
 124 are scored within the task graph to test whether graph-retrieved progress separates known successful
 125 and failed trajectory structure. It should not be interpreted as held-out progress generalization. The
 126 held-out query setting is evaluated separately in Appendix B.3, where held-out rollouts are inserted
 127 only as query trajectories and their terminal outcomes are not used to define success sinks. When

a baseline requires task-specific training or adaptation, we train it for approximately 20 epochs whenever the method implementation and compute budget permit, and otherwise use the closest method-prescribed schedule. We avoid introducing a separate hand-tuned training protocol for each task; method-specific supervision, adaptation, and input-interface details are described below for each baseline.

ReWiND. ReWiND trains a language-guided Transformer reward model to predict task progress from trajectory prefixes. It uses DINOv2 visual embeddings and MiniLM language embeddings, with forward progress supervision, rewind segments, and mismatched-language negatives. For each task, we run the ReWiND pipeline with that task’s language instruction and trajectories, using the official training code while preserving its language-conditioned reward-estimation interface.

SARM. SARM learns a language-conditioned progress reward from robot trajectory videos, avoiding hand-designed dense rewards. For each task, we run SARM with that task’s language instruction and trajectories. We use its **single-stage mode**, where each successful trajectory is treated as one continuous task stage. We intentionally use the strongest setting that does not require extra stage annotation, rather than providing manually segmented stage labels for these long-horizon precise manipulation tasks. Frozen CLIP encoders extract visual and language embeddings, and a Transformer reward head predicts normalized task progress over temporal windows. For efficiency, we precompute CLIP embeddings and evaluate the model using dense per-frame progress curves.

Robo-Dopamine. Robo-Dopamine learns a general process reward model from language instructions and multi-view visual comparisons. Given reference start/end observations and a pair of BEFORE/AFTER observations, the model predicts relative task progress or regression. For each task, we adapt the released Robo-Dopamine GRM checkpoint with LoRA on converted LeRobot trajectories. Each training example contains the task instruction, reference start/end images, and sampled BEFORE/AFTER multi-view images. Successful trajectories provide temporal progress-change supervision. We do not use failed trajectories in the Robo-Dopamine adaptation; its *Failure data* entry is therefore marked as “–” in the baseline-interface summary. These progress-change labels are part of the Robo-Dopamine baseline interface and are counted as additional progress supervision, not as information used by GRP. We freeze the multimodal backbone and train only task-specific LoRA adapters, and evaluate the adapted GRM by converting its predicted progress changes into dense per-frame progress curves.

RoboMeter. RoboMeter is a general-purpose vision-language reward model that combines progress, success, and trajectory-preference prediction. It builds on a Qwen3-VL backbone with separate reward heads for progress, preference, and success. For each task, we fine-tune the RoboMeter-4B checkpoint on the corresponding processed task trajectories. Terminal rewards provide trajectory-level success labels, while sampled trajectory segments are used to construct preference examples that also supervise the progress and success heads. These preference and terminal-success signals are part of the RoboMeter baseline interface and are counted as additional reward-model supervision, not as information used by GRP. In our implementation, this baseline uses the wrist-view camera as its visual input. We evaluate the resulting reward model using its predicted progress and success scores as dense per-frame reward signals.

B.2 Metric Definitions

For an episode τ , let $p(s_t) \in [0, 1]$ denote the normalized progress estimate at time t . We report four metrics.

Δ_{SF} . The success–failure progress gap Δ_{SF} measures separation between successful and failed episodes:

$$\Delta_{\text{SF}} = \mathbb{E}_{\tau \in \text{SE}} [p(s_T^\tau)] - \mathbb{E}_{\tau \in \text{FE}} [p(s_T^\tau)],$$

where s_T^τ is the terminal state of trajectory τ . Larger values indicate better final-outcome separation.

174 ρ_{dev} . We manually localize a fixed set W_{dev} of deviation windows that contain visible local re-
 175 gressions, such as teleoperation mistakes, contact misalignment, or pre-intervention deviations. The
 176 deviation-detection rate is

$$\rho_{\text{dev}} = \frac{1}{|W_{\text{dev}}|} \sum_{(t_s, t_e) \in W_{\text{dev}}} \mathbb{I}[p(s_{t_s}) - p(s_{t_e}) > \delta_{\text{dev}}].$$

177 Higher ρ_{dev} indicates greater sensitivity to fine-grained local regressions.

178 **Deviation-window annotation protocol.** The deviation windows are annotated before computing
 179 any method’s progress scores. For each of the four tasks, an annotator inspects that task’s fixed
 180 evaluation rollout videos and marks short intervals where the robot visibly moves away from task
 181 completion, loses contact alignment, executes an incorrect teleoperation correction, or enters the
 182 frames immediately preceding an intervention. The annotator uses only the raw rollout video and
 183 task context, not GRP scores, learned-reward scores, model predictions, or any progress curve. Each
 184 annotation stores the task name, episode index, start frame, end frame, and the label `bad_interval`;
 185 the same frozen per-task annotation files are then used for all methods. We set δ_{dev} once on the
 186 normalized progress scale and do not tune it per method or task. This makes ρ_{dev} a blind, window-
 187 level test of whether a progress estimator decreases over human-identified local regressions, rather
 188 than a window set selected from a model’s output.

189 **Class. SE and Class. FE.** Following the automated judgment logic of SARM, we classify each
 190 evaluation episode τ using its predicted progress sequence $p(s_t^\tau)$. The predicted episode label is
 191 defined as

$$\text{Label} = \begin{cases} \text{SE}, & \text{if } p(s_T^\tau) > 0.8 \wedge \frac{3}{T} \sum_{t=\lceil 2T/3 \rceil}^T p(s_t^\tau) > 0.6, \\ \text{FE}, & \text{otherwise,} \end{cases}$$

192 Class. SE measures how many successful episodes are correctly recognized as successful, while
 193 Class. FE measures how many failed episodes are correctly rejected as failures.

194 B.3 Demo-Only Zero-Shot Progress Evaluation

195 As an additional single-task diagnostic,
 196 separate from the four-task baseline com-
 197 parison above, we evaluate a stricter zero-
 198 shot progress setting that approximates on-
 199 line rollout scoring before rollout aggrega-
 200 tion by using a Screw Tightening dataset.
 201 The base graph is built only from 100 suc-
 202 cessful expert demonstrations. We then re-
 203 serve 18 additional successful trajectories
 204 and 17 failed trajectories as held-out roll-
 205 outs. Each held-out rollout is temporarily
 206 inserted into the demonstration-only graph,
 207 scored by graph-retrieved progress (GRP), and removed before the next rollout is evaluated. Thus,
 208 held-out rollouts cannot exchange information with each other, and their outcomes are never used to
 209 define success states.



Figure 4: Demo-only zero-shot progress evaluation on held-out Screw Tightening rollouts.

210 This protocol tests whether GRP can provide a query-time progress signal for unseen online roll-
 211 outs before they are added to the training pool. Table 7 reports trajectory-level progress diagnostics.
 212 **Terminal Progress** is the progress value at the final sampled state of a rollout. **Trajectory Avg.**
 213 **Progress** averages progress over all sampled states in the rollout. **Low-Progress State Ratio** mea-
 214 sures the fraction of sampled states whose progress is below the low-progress threshold, indicating
 215 how often the rollout stays far from the demonstrated success manifold. **Reachable State Ratio**
 216 measures the fraction of sampled states that can reach a demonstration success node through the

graph. **False Classification** reports false negatives (FN) for held-out successful rollouts and false positives (FP) for failed rollouts.

In this diagnostic split, GRP yields a large terminal-progress gap between held-out successful and failed rollouts. All failed trajectories receive zero terminal progress, while most held-out successful trajectories reach high terminal progress. This suggests that graph-retrieved progress can provide a useful query-time rollout score in this setting. As verified online rollouts are appended to the graph, the reachable state support expands, so the progress signal can become increasingly accurate without retraining a dense neural reward model.

Table 7: **Demo-only zero-shot progress evaluation.** The graph is built from successful demonstrations only. Held-out rollouts are inserted one at a time, scored by graph-retrieved progress, and removed before the next rollout is evaluated. The diagnostic split yields a large terminal-progress gap between held-out successful and failed rollouts.

Split	# Traj.	Terminal Progress	Trajectory Avg. Progress	Low-Progress State Ratio	Reachable State Ratio	False Classification
Held-out success	18	0.879	0.440	0.328	0.998	2/18 FN
Held-out failure	17	0.000	0.019	0.952	0.127	0/17 FP
Success-failure gap	—	0.879	0.421	−0.624	0.871	—

Note: The demo-only base graph contains 8756 nodes, 8656 temporal edges, 9700 similarity edges, and 100 successful terminal nodes.

C Policy Improvement Evaluation Details and Single-task Backbone Compatibility Study

This section provides the raw policy-evaluation protocol, baseline training details, and an additional controlled single-task backbone compatibility study for using GR-BC as a post-training interface beyond the X-VLA setting evaluated in the main four-task study.

C.1 VLA Training Details

The main iterative policy-learning study uses X-VLA [8] because it enables controlled multi-round real-robot evaluation under the same data and compute budget. The single-task backbone compatibility study in Appendix C.2 additionally evaluates SmolVLA [9] and $\pi_{0.5}$ [10]. Across backbones, GR-BC preserves the original supervised VLA loss and only changes the sample-level multiplier w_t .

C.2 Single-task Backbone Compatibility Study on Graphics Card

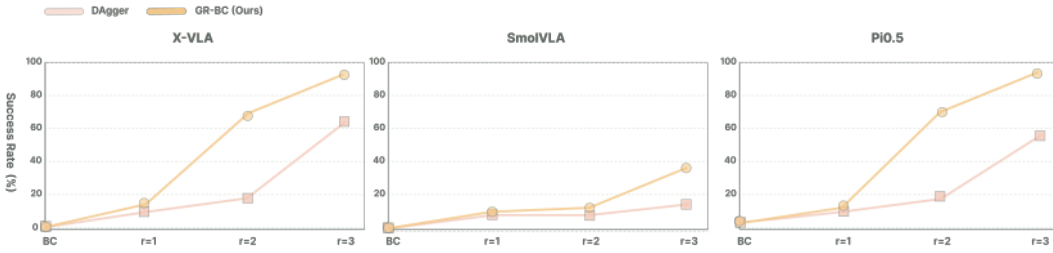
The full iterative policy-learning comparison in Sec. 5.2 of the main paper uses X-VLA [8] for all baselines, since X-VLA allows controlled multi-round evaluation across four real-world tasks under the same compute and data budget. To test backbone compatibility in a controlled single-task study, we additionally evaluate the Graphics Card task with three different VLA backbones: SmolVLA [9], $\pi_{0.5}$ [10], and X-VLA [8].

In this study, we compare GR-BC with the DAgger-style uniform aggregation baseline under each backbone. The graph construction, GRP computation, transition weights, data split, and evaluation poses are kept fixed, while only the VLA backbone and its corresponding supervised training objective are changed. For SmolVLA, which predicts a 50-frame action chunk, we use the same 30-frame GRP consequence horizon for weighting and apply the resulting sample-level weight to the full supervised chunk. This preserves a fixed graph-scoring interface while leaving the backbone-specific action objective unchanged. This setting supports backbone compatibility in a controlled single-task study, while leaving broader architecture and task coverage to future evaluation.

As shown in Table 9 and Fig. 5, GR-BC improves over the DAgger-style baseline across all three VLA backbones on the Graphics Card task. The improvement is especially clear after the second and third post-training rounds, where graph-based consequence reweighting better exploits useful recovery and progress-making chunks from the aggregated rollout data. These results support backbone

Table 8: VLA post-training hyperparameters.

Parameter	X-VLA	SmolVLA	$\pi_{0.5}$
Input image resolution	Raw videos: 480×640 ; model inputs: 256×256 , 256×256 , 224×224	480×640 raw RGB videos; padded to 512×512	480×640 raw RGB videos; resized to 224×224
Number of camera views	3	3	3
Action representation	8-D joint/gripper command	8-D joint/gripper command	8-D joint/gripper command
Action chunk horizon N	30 frames	50 frames	30 frames
Hardware	4 NVIDIA H100 GPUs	4 NVIDIA H100 GPUs	4 NVIDIA H100 GPUs
Batch size	32 per GPU; 128 effective	32 per GPU; 128 effective	16 per GPU; 64 effective
Learning rate	1.0×10^{-4}	1.0×10^{-4}	2.5×10^{-5}
Optimizer	$\beta = (0.9, 0.95)$; wd = 1.0×10^{-4}	$\beta = (0.9, 0.95)$; wd = 1.0×10^{-10}	$\beta = (0.9, 0.95)$; wd = 1.0×10^{-2}
Training steps	20k	20k	40k
Training time	8h	6h	20h

Figure 5: Single-task backbone compatibility on Graphics Card. GR-BC improves over the DAGger-style baseline across X-VLA, SmolVLA, and $\pi_{0.5}$ in this controlled study.

254 compatibility in a controlled single-task study. Because GR-BC only multiplies the original super-
255 vised VLA loss by the precomputed chunk weight w_t , it can be applied to several supervised VLA
256 objectives in this setting, including standard behavior cloning, diffusion-style action prediction [11],
257 and flow-style action prediction.

258 C.3 Policy Baseline Training Details

259 All policy baselines start from the same BC checkpoint trained on successful demonstrations. The
260 DAGger-style baseline aggregates shared-control rollout data and applies unweighted supervised
261 training [12]; AWR uses action-value-derived weights [13]; RA-BC follows reward-weighted be-
262 havior cloning using a learned progress or reward model such as SARM [5]; GR-BC uses graph-
263 retrieved progress and does not train an extra neural reward, value, critic, or progress model. The
264 AWR baseline is conceptually related to advantage-weighted and action-level off-policy VLA post-
265 training methods [13, 14].

266 **DAGger-style.** We implement a DAGger-
267 style uniform aggregation baseline inspired
268 by Dataset Aggregation [12], adapted to our
269 shared-control action-chunk setting. This is not
270 a pure oracle-relabeling DAGger setup: we do
271 not retroactively query an expert action for ev-
272 ery learner-visited autonomous state. In each
273 offline post-training round, rollout trajectories
274 collected with SpaceMouse shared control are
275 added to the dataset. Autonomous chunks use
276 the action chunk actually executed by the pol-
277 icy, while intervention chunks use the human

Table 9: Single-task backbone compatibility study on the Graphics Card task. We report success rate (%) across behavior cloning initialization and three iterative post-training rounds.

Backbone	Method	BC	$r = 1$	$r = 2$	$r = 3$
X-VLA	DAGger-style	0	8	17	63
	GR-BC	0	14	67	92
SmolVLA	DAGger-style	0	8	8	14
	GR-BC	0	10	12	35
$\pi_{0.5}$	DAGger-style	3	12	18	55
	GR-BC	3	14	69	93

teleoperation or corrective action chunk. The new data is uniformly aggregated with the historical dataset $\mathcal{D}^r$, and the policy is updated via the standard unweighted maximum-likelihood objective:

$$\mathcal{L}_{\text{DAgger-style}} = \mathbb{E}_{(s_t, g, a_{t:t+N-1}) \sim \mathcal{D}^r} [\mathcal{L}_{\text{VLA}}(\pi_\phi, s_t, g, a_{t:t+N-1})].$$

Unlike GR-BC, the DAgger-style baseline treats all aggregated transitions as equally positive supervision ($w_t = 1$), meaning it does not evaluate the consequence of individual action chunks or mathematically distinguish between autonomous error segments and expert recovery segments.

RA-BC. We implement Reward-Aligned Behavior Cloning (RA-BC) following the pipeline of SARM [5], which leverages a learned neural progress model to reweight imitation data. Since none of the compared progress pipelines uses multi-stage task segmentation, we use the same single-stage/single-mode setting for RA-BC: each successful trajectory is treated as one continuous stage, and its progress target is the normalized temporal position within that successful trajectory. We train $P_\theta(s) \in [0, 1]$ on successful demonstrations only, rather than adding rollout, failure, or intervention labels to the reward model. This avoids giving the learned reward baseline extra supervision beyond the single-stage progress interface; rollout and intervention data are still used in the downstream policy dataset, but not as reward-model labels. For each action chunk $a_{t:t+N-1}$ executed at state s_t , the trajectory impact is evaluated via the endpoint difference

$$\Delta \hat{p}_t = P_\theta(s_{t+N}) - P_\theta(s_t).$$

In the LeRobot RA-BC loader, the default endpoint mode uses N frames ahead within the same episode, falls back to the last frame near episode end, and maps the progress delta $r_i = \Delta \hat{p}_t$ into a sample weight using $\kappa = 0.01$ and $\epsilon = 10^{-6}$:

$$\tilde{w}_i = \text{clip}\left(\frac{r_i - (\mu - 2\sigma)}{4\sigma + \epsilon}, 0, 1\right), \quad w_i = \mathbb{I}[r_i > \kappa] + \mathbb{I}[0 \leq r_i \leq \kappa] \tilde{w}_i,$$

where μ and σ are computed over all valid progress deltas in the training split. Negative-progress chunks receive zero weight, missing deltas use the fallback weight 1.0, and each minibatch is renormalized so the weights sum to the batch size. Because RA-BC does not retrieve dataset-supported remaining steps from a structural reachability graph, its continuous reward signal lacks topological granularity, making it prone to missing local execution errors such as slight misalignments or transient intra-chunk regressions.

AWR. We implement AWR following the ALOE-style policy in our LeRobot codebase. The baseline has two stages. First, the actor is initialized from the same supervised VLA checkpoint and frozen while training an offline action-chunk critic. The critic is a Transformer over state, image tokens, and the full action chunk, with hidden size 256, 6 layers, 8 attention heads, and two Q readout tokens; we take the minimum over the two heads as the pessimistic value Q_{pess} . For a chunk with valid horizon h , the training reward is -1 per executed frame, with terminal reward 0 for successful episodes and $-L_{\text{fail}}$ for failures, where L_{fail} defaults to the mean episode length for the task. The TD target is

$$y_t = \sum_{j=0}^{h-1} \gamma^j r_{t+j} + \gamma^h \min_k Q_{\bar{\theta}, k}(s_{t+h}, \hat{a}_{t+h:t+h+N-1}, g),$$

with $\gamma = 0.99$; the bootstrap term is removed when the chunk reaches a terminal state. The next action chunk $\hat{a}$ is sampled from the frozen current policy and cached during replay construction. We optimize the critic for 50k steps with batch size 128, Adam lr 3×10^{-4} , cosine scheduling with 100 warmup steps, gradient clipping at 1.0, and target-critic Polyak update weight 0.005. Second, we freeze the critic, score the aggregated policy-training chunks, and run a weighted behavior-cloning update from the same BC initialization. The critic defines the chunk advantage

$$A^\pi = Q_{\text{pess}}(s_t, a_{t:t+N-1}, g) - V^\pi(s_t, g),$$

Table 10: Hyperparameters and recommended ranges for GR-BC.

Parameter	Meaning	Recommended range	Effect
k	Temporal subsampling interval	10–20 frames	Controls graph density. Smaller values preserve fine contact dynamics but increase graph size; larger values reduce cost but may miss short regressions.
K	Mutual KNN size per camera view	10–30	Controls similarity-edge recall. Larger values improve connectivity but may introduce false visual neighbors.
α_{edge}	Adaptive similarity-edge threshold	0.8–1.2	Controls cross-trajectory stitching. Larger values improve failure reachability; smaller values give more conservative edges.
ϵ_c	Stability constant for adaptive length	10^{-5} – 10^{-3}	Prevents unstable normalization when local visual scale is very small.
R_{max}	Normalization scale for remaining steps	95th percentile; 95–99th percentile recommended range	Maps remaining steps to normalized progress. A small value saturates progress early; a large value compresses progress differences.
N	Action chunk horizon	20–40 frames	Defines the lookahead for chunk consequence. Smaller values give local credit; larger values capture longer effects but may mix behaviors.
δ_{min}	Minimum net progress threshold	0.001–0.005	Filters stagnant chunks caused by interpolation or visual noise.
η	Maximum allowed progress regression	0.02–0.05	Controls tolerance to transient regressions. Smaller values suppress unstable chunks; larger values keep more recovery behavior.
β_w	Progress bonus amplitude	0.3–0.8	Controls the strength of reweighting among accepted chunks.
γ	Progress sensitivity	10–30	Controls how sharply weights respond to small progress changes.
w_{min}	Lower accepted-branch weight	0.3–0.7	Prevents accepted slow-progress chunks from being ignored.
w_{max}	Upper accepted-branch weight	1.3–2.0	Prevents a few high-progress chunks from dominating training.

where V^π is estimated by evaluating action chunks sampled from the current policy with the same critic. We convert the advantage into a clipped exponential weight,

$$w_t^{\text{AWR}} = \exp(\text{clip}(A^\pi / \beta_{\text{AWR}}, -\epsilon_{\text{clip}}, \epsilon_{\text{clip}})),$$

batch-normalize the weights to mean one, and use them to reweight the standard behavior-cloning objective. The temperature β_{AWR} and clipping bound ϵ_{clip} are fixed globally for all tasks and rounds rather than tuned per task.

D Graph Diagnostics, Representation Ablations, and Hyperparameter Analysis

This section explains why the graph works, why MCR is used for node representation, how robust the method is to graph and weighting hyperparameters, and how shortest-path visualizations support the consequence-based weighting claim.

D.1 Hyperparameter Analysis

GR-BC contains three groups of hyperparameters: graph construction, progress normalization, and chunk-level weighting. This appendix reports diagnostic sensitivity checks on graph connectivity and chunk weights, followed by the default ranges used in our experiments. These checks are graph-level and weight-level diagnostics on the Assembly Model rollout set; they are not an exhaustive policy-retraining sweep over every hyperparameter.

Table 11 reports two compact diagnostics supporting the ranges in Table 10. The left panel varies the temporal subsampling interval k under the same gripper-filtered graph rule. Smaller k increases graph size and edge count; $k = 10$ – 15 keeps high failure-rollout reachability, and $k = 15$ is our default cost–coverage tradeoff. The right panel varies the chunk-weight rule, where $w = 0$ if $\Delta p_{30} \leq \delta_{\text{min}}$, otherwise $w = \text{clip}(1 + \beta_w \tanh(\gamma \Delta p_{30}), 0.5, 1.5)$. Increasing δ_{min} makes the accepted set more selective and raises the zero-weight ratio, while larger β_w and γ sharpen weight separation but remain bounded by $w_{\text{max}} = 1.5$.

Table 11: Hyperparameter sensitivity diagnostics.

Temporal subsampling						Chunk weighting							
k	Sim. edges	Fail reach.	$R_{\max}$	Zero wt.	High wt.	$\delta_{\min}$	β_w	γ	Accepted	Zero wt.	Mean wt.	> 1 wt.	p90 / max
5	62,349	95.8%	125.0	26.8%	1.9%	0.003	0.3	10	72.7%	27.3%	0.811	72.7%	1.195 / 1.228
10	30,221	97.5%	194.0	26.4%	21.7%	0.005	0.5	20	69.6%	30.4%	0.915	69.6%	1.457 / 1.482
15	20,229	98.3%	255.0	24.6%	38.3%	0.010	0.5	20	63.2%	36.8%	0.845	63.2%	1.457 / 1.482

D.2 Encoder Comparison: R3M, MCR, and V-JEPA2.1

Graph construction depends on whether the representation groups physically compatible manipulation states together. We compare MCR [1] with R3M [2] and two V-JEPA2.1 visual encoders [15]. We focus on graph-level diagnostics, since visually plausible retrieval can still introduce harmful shortcuts when the retrieved states are dynamically incompatible. Figure 6 provides a qualitative overview of the resulting graph structure and progress behavior.

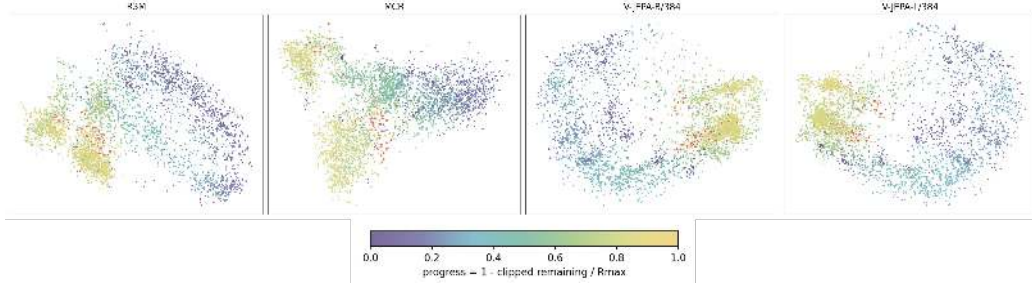


Figure 6: Encoder comparison overview on the Assembly Model rollout dataset. The visualization compares graph structure and progress behavior across R3M, MCR, and V-JEPA2.1 encoders under the same graph-construction rule.

Table 12: Encoder comparison on the Assembly Model rollout dataset. All methods use stride 15, history length 0, and the same multi-view consensus and gripper-consistency check. Arrows indicate the preferred direction.

Encoder	Sim. edges ↓	$R_{\max}$	Reachable ↑	Mean regression ↓	Total variation ↓	End progress ↑	Encode time ↓
R3M	12913	145.1	96.3%	0.388	1.420	0.745	60.2s
MCR	9973	223.5	94.4%	0.437	1.289	0.713	64.3s
V-JEPA2.1-B/384	19310	129.4	96.9%	0.656	1.688	0.567	123.7s
V-JEPA2.1-L/384	18980	131.8	95.1%	0.642	1.655	0.562	176.9s

Table 12 summarizes the graph diagnostics. “Sim. edges” counts accepted non-temporal similarity edges after multi-view consensus and gripper-consistency checks. Unless otherwise specified, $R_{\max}$ is the 95th percentile of finite shortest-path distances to success and is used for progress normalization; we recommend 95–99% as a robust range. “Reachable” measures the fraction of nodes connected to a success node. “Mean regression” and “Total variation” quantify backward progress jumps and overall progress smoothness. “End progress” is the average final episode progress, and “Encode time” is the feature extraction time for all sampled nodes and three camera views.

R3M is competitive and even better on mean regression and encoding speed, while MCR yields a more conservative graph with lower total variation and fewer accepted similarity shortcuts. Both V-JEPA2.1 encoders are substantially slower and produce denser graphs with larger regression and variation. We choose MCR because it gives the most conservative graph among the compared encoders, with the lowest total variation and acceptable encoding cost.

D.3 Multiple Successful Terminal Clusters

We use Assembly Model as a diagnostic for multi-terminal success. The task admits multiple correct terminal configurations, and our analysis recovers this structure directly: successful terminal nodes form two visually and physically distinct groups, as shown by the feature-space visualization and representative camera frames in Figure 7. Our graph connects all successful terminals to the virtual sink g_* , so $R_{\text{rec}}(s)$ is the shortest cost to the nearest reachable success mode rather than to a

single hand-specified goal. This supports the robustness of our pipeline under multi-modal success definitions.

The value $p \in [0, 1]$ indicates the normalized trajectory progress. Labels m0 and m1 correspond to two distinct successful execution modes, while fail denotes states from failed trajectories.

D.4 Shortest-Path Visualization

Figure 8 visualizes one recovered path from a failed state to a successful terminal state. In the visualization, E denotes the episode index and F denotes the frame index within that episode. The edge label T denotes a temporal transition within an episode, and S denotes a similarity-based bridge across episodes. Start and Goal mark the queried state and the recovered successful terminal state, respectively.

This example shows that our pipeline does not discard failed data. Instead, failed states can be connected through similarity bridges to useful recovery segments and then routed toward successful terminals. Moreover, the path crosses different episodes and success modes, indicating that the method can exploit shared sub-trajectories where different successful strategies intersect. As a result, the graph-based weighting captures reusable structure from both successful and failed demonstrations, enabling effective consequence-based credit assignment.

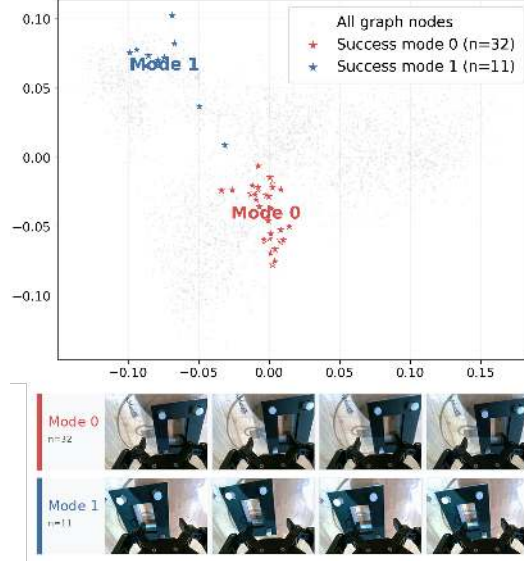


Figure 7: Multiple Success Modes on Assembly Model.

Episode-coded graph shortcut

Episode route: E83 -> E69 -> E11 -> E84 -> E54 | S: similarity jump, T: temporal step



Figure 8: Shortest-path visualization from a failed state to a successful terminal state. The path alternates between temporal transitions and similarity bridges, showing how failed data can be routed through reusable recovery segments.

391 E Additional Progress Visualization

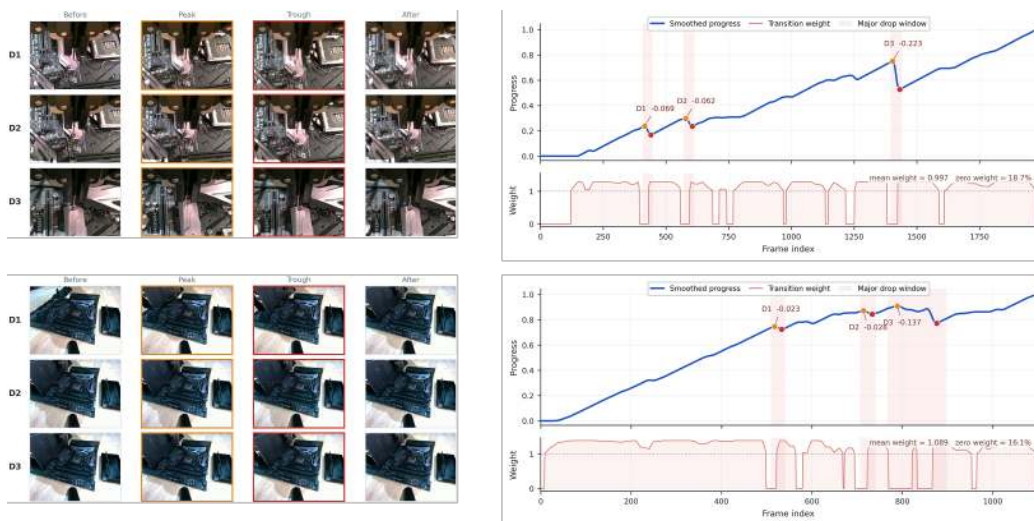


Figure 9: Examples of Progress prediction on demonstration data.

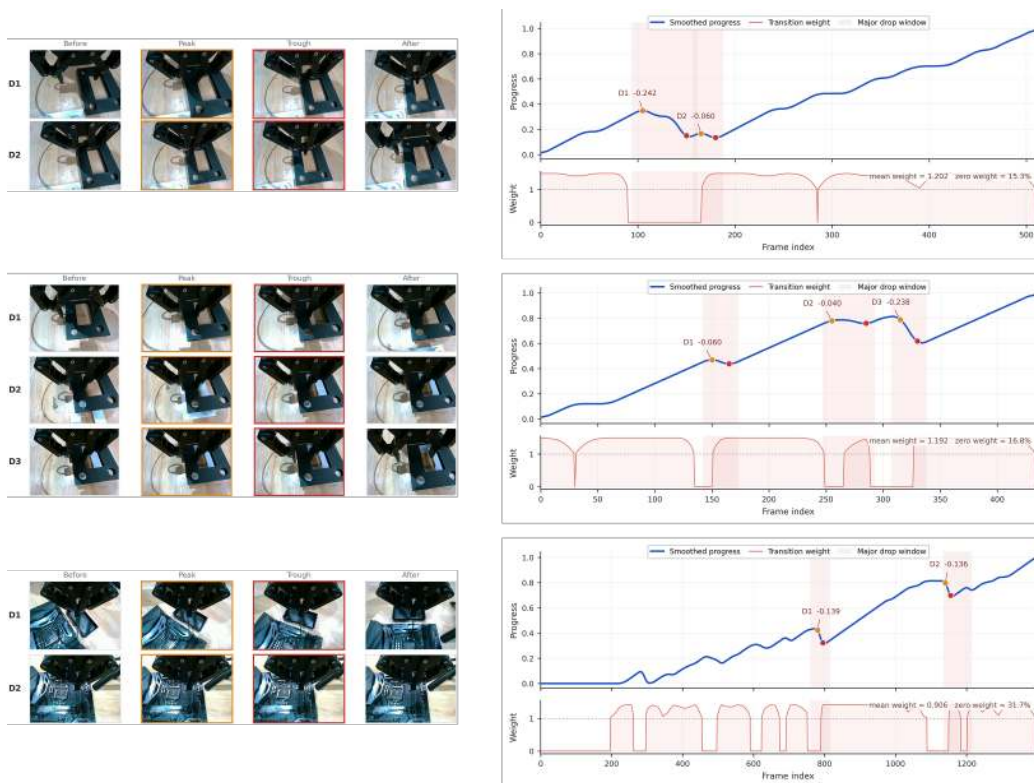


Figure 10: Examples of Progress prediction on policy rollouts.

References

- [1] G. Jiang, Y. Sun, T. Huang, H. Li, Y. Liang, and H. Xu. Robots pre-train robots: Manipulation-centric robotic representation from large-scale robot datasets. In *International Conference on Learning Representations*, volume 2025, pages 81885–81905, 2025.
- [2] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.
- [3] E. W. Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: his life, work, and legacy*, pages 287–290. 2022.
- [4] J. Zhang, Y. Luo, A. Anwar, S. A. Sontakke, J. J. Lim, J. Thomason, E. Biyik, and J. Zhang. Rewind: Language-guided rewards teach robot policies without new demonstrations. *arXiv preprint arXiv:2505.10911*, 2025.
- [5] Q. Chen, J. Yu, M. Schwager, P. Abbeel, Y. Shentu, and P. Wu. Sarm: Stage-aware reward modeling for long horizon robot manipulation. *arXiv preprint arXiv:2509.25358*, 2025.
- [6] H. Tan, S. Chen, Y. Xu, Z. Wang, Y. Ji, C. Chi, Y. Lyu, Z. Zhao, X. Chen, P. Co, et al. Robo-dopamine: General process reward modeling for high-precision robotic manipulation. *arXiv preprint arXiv:2512.23703*, 2025.
- [7] A. Liang, Y. Korkmaz, J. Zhang, M. Hwang, A. Anwar, S. Kaushik, A. Shah, A. S. Huang, L. Zettlemoyer, D. Fox, et al. RoboMeter: Scaling general-purpose robotic reward models via trajectory comparisons. *arXiv preprint arXiv:2603.02115*, 2026.
- [8] J. Zheng, J. Li, Z. Wang, D. Liu, X. Kang, Y. Feng, Y. Zheng, J. Zou, Y. Chen, J. Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.
- [9] M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi, C. Pascal, M. Russi, A. Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.
- [10] K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. R. Equi, C. Finn, N. Fusai, M. Y. Galliker, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *9th Annual Conference on Robot Learning*, 2025.
- [11] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.
- [12] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.
- [13] X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.
- [14] R. Yang, H. Wang, C. Liu, X. Yan, Y. Wang, X. Du, S. Yue, Y. Liu, C. Zhang, L. Qi, et al. Aloe: Action-level off-policy evaluation for vision-language-action model post-training. *arXiv preprint arXiv:2602.12691*, 2026.
- [15] L. Mur-Labadia, M. Muckley, A. Bar, M. Assran, K. Sinha, M. Rabbat, Y. LeCun, N. Ballas, and A. Bardes. V-jepa 2.1: Unlocking dense features in video self-supervised learning. *arXiv preprint arXiv:2603.14482*, 2026.