PEARLVLA: PROGRESSIVE EMBODIED ACTION-PLAN REFINEMENT IN LATENT SPACE

Anonymous authors
Paper under double-blind review

ABSTRACT

Current Vision-Language-Action (VLA) models face a trade-off between efficient action generation and explicit deliberation. Directly decoding actions from vision-language backbone representations enables low-latency control, whereas textual reasoning, pixel-level subgoals, or world-model evaluation of decoded actions can improve planning but incur substantial latency and computational cost. We propose PearlVLA, a VLA framework that progressively refines a VLM-derived latent plan using feedback from the predicted consequence of each intermediate plan. PearlVLA uses a frozen latent world model (LaWM) pretrained on action-free video. At each refinement round, the current plan produces a continuous latent action code, and the LaWM predicts the corresponding latent visual subgoal. A future-guided plan refiner uses this subgoal to update the plan, so each revision reshapes the next LaWM query. After K rounds, the refined plan is passed once to the host policy's action interface to produce an action chunk. We further introduce Causal Refinement-Grouped Process-Reward RL to optimize latent refinement by comparing rewards from the longer-horizon imagined futures of plan edits made at the same refinement state. Experiments on the LIBERO and RoboCasa benchmarks show that PearlVLA performs competitively against strong existing methods.

1 INTRODUCTION

Vision-Language-Action models (VLAs) provide a unified framework for embodied intelligence by mapping visual observations, language instructions, and robot states to actions (Zitkovich et al., 2023; Black et al., 2024). Embodied manipulation, however, cannot be reduced to isolated observation-to-action mappings. Over long horizons, reasoning about scene geometry, object interactions, and future state transitions is essential, as errors in grasp pose, motion direction, or goal interpretation can compound into failure. A capable policy should therefore anticipate the likely consequence of its current plan before acting (Du et al., 2023).

A natural way to provide foresight is to place an intermediate signal before action decoding. Prior VLA systems have used textual reasoning traces or visual subgoal generation (Zawalski et al., 2024; Zhao et al., 2025), feedback from value functions or world models on decoded action proposals (Song et al., 2025; Huang et al., 2025), and recurrent latent computation (Tur et al., 2026). Although helpful, these mechanisms have different limitations: explicit generation adds inference before action decoding, action-level feedback ties each candidate or revision to an action decode, and latent recurrence receives no modeled consequence between updates. Efficient VLA methods avoid these intermediate stages by directly decoding actions from VLM representations (Kim et al., 2025), but this direct path provides no explicit mechanism to preview the future induced by the current plan or self-correct before action decoding. Figure 1(a–d) contrasts these design choices with ours (e–f).

For embodied control, latent deliberation concerns not only internal computation but also the anticipated physical consequences of the plan being refined. Recent work learns continuous latent action codes from visual transitions and uses a latent world model (LaWM) to predict latent visual subgoals without pixel reconstruction (Garrido et al., 2026). In existing single-query policies, the VLM estimates the code without access to the future observation, and the resulting subgoal conditions action generation rather than plan refinement (Chen et al., 2026).

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.

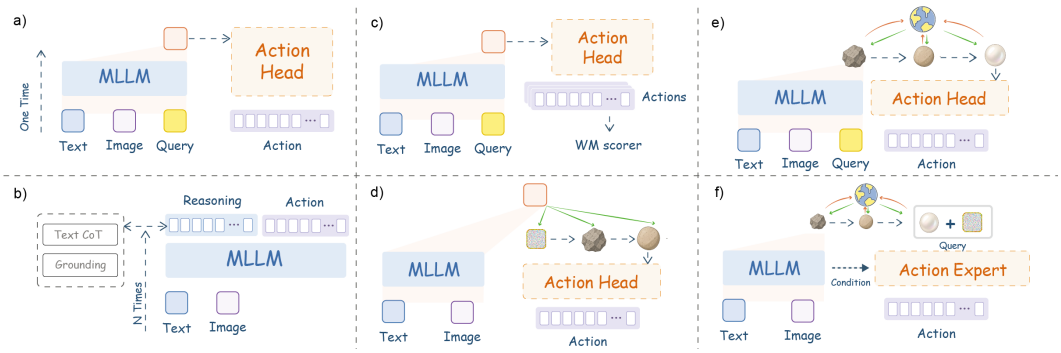


Figure 1: **Design choices for VLA policies.** (a) Direct action decoding. (b) Explicit intermediate reasoning. (c) Action-space world-model search. (d) Latent recurrent reasoning. (e) PearlVLA with a regression policy. (f) PearlVLA with a diffusion or flow policy.

To turn this one-way interface into a refinement loop, PearlVLA keeps the current visual feature u_t fixed while maintaining a writable VLM-derived latent plan. At refinement round k, the committed latent action code c_k queries the frozen LaWM for a latent visual subgoal $\hat{u}_k$. A future-guided plan refiner uses this subgoal and its change from the first prediction to update z_{k-1} into z_k, which then composes the next query c_{k+1}. Thus each revision reshapes the next LaWM query rather than merely consuming a static future feature. After K rounds, the refined plan is consumed by the host policy's action interface to produce an action chunk. We further introduce Causal Refinement-Grouped Process-Reward RL (CRG-PRL), which treats each latent-plan edit as an action in an inner decision process. For each candidate edit, the resulting latent visual subgoal is autoregressively extended, rendered, and scored to obtain a process reward. CRG-PRL compares edits sampled from the same refinement state, separating edit quality from state difficulty through group-relative advantages without a learned value function (Shao et al., 2024; Zheng et al., 2025).

Our contributions are summarized as follows:

1. We present PearlVLA, a VLA framework for progressive, future-guided refinement of a VLM-derived latent action plan before action decoding.
2. We introduce an anchored plan–code–future loop in which each plan update composes the next latent action code, elicits an updated modeled future from the frozen LaWM, and feeds that future into the next plan revision.
3. Building on supervised refinement, we introduce CRG-PRL, a critic-free method that optimizes intermediate plan edits by comparing the longer-horizon imagined outcomes of alternative edits from the same refinement state.
4. Experiments on LIBERO and RoboCasa show that PearlVLA outperforms strong baselines, achieving average success rates of 98.7% and 67.8%, respectively.

2 Related Work

Reasoning and refinement in VLAs. Language-model research shows that useful intermediate computation need not be fully verbalized. Coconut recurrently feeds hidden states back as continuous thoughts, while looped Transformers obtain reasoning depth by repeatedly applying shared computation (Hao et al., 2024; Saunshi et al., 2025). In VLAs, ECoT and CoT-VLA generate textual or visual intermediates before actions (Zawalski et al., 2024; Zhao et al., 2025). Diffusion Policy and RD-VLA spend additional inference computation on iterative action sampling or recurrent latent states (Chi et al., 2025; Tur et al., 2026). World-model feedback has also been incorporated into action refinement by LaDi-WM and ProgressVLA, with related work applying a single proposal-conditioned consequence update (Huang et al., 2025; Yan et al., 2026; Sun et al., 2026).

World models and latent actions. Predictive representations and latent actions provide compact dynamics abstractions for robot policies. WoG distills future-conditioned features into the VLM,

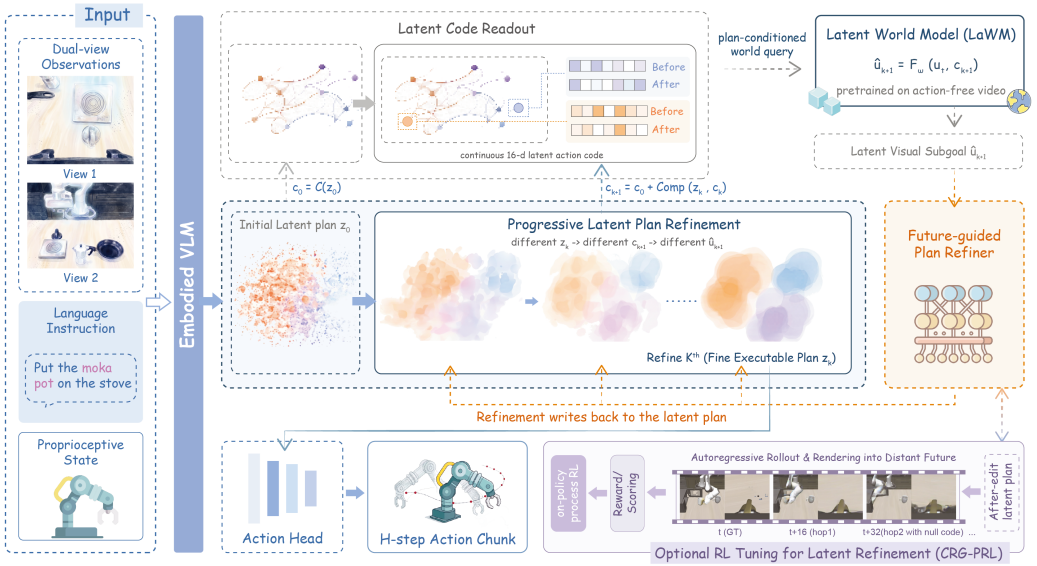


Figure 2: **The framework of PearlVLA.** The embodied VLM encodes its inputs into an initial latent plan. Over K rounds, each refinement step composes a latent-code query, retrieves a latent visual subgoal from the frozen LaWM, and writes the resulting future feedback back into the plan. The refined plan is then decoded into an H-step action chunk. The optional CRG-PRL branch further optimizes the refinement process during training.

while LAPO and UniVLA learn latent actions from visual transitions for pretraining and control (Su et al., 2026; Schmidt & Jiang, 2023; Bu et al., 2025). LaWAM retains the latent-action decoder as LaWM and conditions an action expert on its predicted latent visual subgoal (Chen et al., 2026). PearlVLA closes the latent predictive loop at the plan level: each LaWM response revises the writable VLM plan and thereby reshapes the next query, while the downstream action interface is invoked only after refinement.

3 METHOD

Figure 2 provides an overview of PearlVLA. Given the current RGB observation o_t, language instruction ℓ, and proprioceptive state s_t, a host-specific interface obtains an initial latent plan z_0 from the embodied VLM context. PearlVLA then couples three representation spaces: the writable VLM plan z, a continuous latent action code c, and the visual-feature space u of a frozen latent world model. At each round, the current code queries LaWM for a corresponding latent future, and the resulting feedback revises the plan before the next code is composed. After K rounds, the host action interface is conditioned on the refined plan together with its instantiation-specific context. We reserve z for the latent plan and c for the latent action code throughout.

For the OpenVLA instantiation, PearlVLA uses an OpenVLA-7B backbone following the OpenVLA-OFT design (Kim et al., 2024; 2025). A transformer-based proprio encoder converts proprioceptive history into tokens, which are fused with visual and language tokens in the multimodal sequence `[BOS] [vision] [proprio] [text] [meta_query]`; the final representations of eight meta-query tokens provide the OpenVLA plan initialization. For GR00T N1.5 (Bjorck et al., 2025), sixteen learnable plan queries cross-attend to the frozen VLM's instruction-token states following the three image-token blocks, and the resulting query representations provide the GR00T plan initialization. The subsequent plan–code–prediction loop is shared across both hosts.

3.1 LATENT WORLD MODEL AS A PREDICTIVE INTERFACE

PearlVLA obtains its predictive interface from a latent action model pretrained on videos without robot-action inputs (Garrido et al., 2026). Let $o_t^{\mathrm{wm}} \subseteq o_t$ denote the two-view subset shared by

LaWM, the renderer, and the reward branch; benchmark-specific view choices are given in Section 4.1. A frozen visual encoder E_ψ maps the current and horizon view pairs to u_t and $u_{t+\Delta}$. An inverse-dynamics encoder q_φ compresses their visual transition into a continuous 16-dimensional latent action code, while a forward-dynamics decoder F_ω predicts the corresponding horizon feature:

$$\begin{aligned} u_t &= E_\psi(o_t^{\mathrm{wm}}), & u_{t+\Delta} &= E_\psi(o_{t+\Delta}^{\mathrm{wm}}), \\ c &\sim q_\varphi(c \mid u_t, u_{t+\Delta}), & \hat{u}_{t+\Delta} &= F_\omega(u_t, c). \end{aligned} \tag{1}$$

Here Δ denotes the visual prediction offset of the latent action model; our experiments use $\Delta = 16$ frames. After pretraining, q_φ and F_ω are frozen, and F_ω serves as the latent world model (LaWM). LaWM is conditioned on the transition code c and returns a latent visual subgoal $\hat{u}_{t+\Delta}$. During policy training, the posterior mean $c^\star = \mathbb{E}_{q_\varphi(c|u_t,u_{t+\Delta})}[c]$ provides a hindsight target. At test time, the future feature is unavailable, so a host readout infers the VLM-derived initial foresight code c_0 from current context alone. Figure 3 illustrates the sensitivity of the frozen predictive interface to teacher, initial, and intermediate codes; refinement ablations evaluate the utility of repeated queries. LaWM pretraining and the training-only renderer are detailed in Appendix A.

3.2 Anchored Future-Guided Plan Refinement

The central role of the predicted future is to provide an update signal for the latent plan. Let C denote the host-specific readout from the initial plan to the LaWM code space. For the initial readout only, let $\tilde{z}_0$ denote the unmodified VLM plan underlying the writable initial state z_0; host-specific initialization details are deferred to Appendix B. The VLM-derived initial foresight code $c_0 = C(\tilde{z}_0)$ receives direct code supervision, while its uses inside the refinement loop are detached. Its predicted subgoal likewise provides a detached reference:

$$c_0 = C(\tilde{z}_0), \qquad \hat{u}_0 = \mathrm{sg}(F_\omega(u_t, c_0)), \qquad c_1 = \mathrm{sg}(c_0) + \mathrm{Composer}(z_0, \mathrm{sg}(c_0)). \tag{2}$$

For $k = 1, \ldots, K$, we write $\hat{u}_k \equiv \hat{u}_{t+\Delta}^{(k)} = F_\omega(u_t, c_k)$, where k indexes the refinement query rather than physical time; every query predicts the same offset Δ. At refinement round $k = 1, \ldots, K$, the committed code c_k queries LaWM for a latent visual subgoal. FutureEncoder summarizes the modeled transition from the current observation together with its change from the detached reference subgoal, and RefineNet maps this relative future feedback into a residual plan edit:

$$\begin{aligned} \hat{u}_k &= F_\omega(u_t, c_k), \\ \tau_k &= [\hat{u}_k - u_t\,;\, \hat{u}_k - \hat{u}_0], \qquad h_k = \mathrm{FutureEncoder}(\tau_k), \\ \delta_k &= \mathrm{RefineNet}(z_{k-1}, h_k, c_k), \\ z_k &= z_{k-1} + w_k \delta_k, \\ c_{k+1} &= \mathrm{sg}(c_0) + \mathrm{Composer}(z_k, c_k). \end{aligned} \tag{3}$$

The first component of τ_k represents the visual change predicted under the current code, while the second measures its deviation from the shared reference subgoal $\hat{u}_0$. Composer uses the revised plan and current code to form the next query relative to $\mathrm{sg}(c_0)$, avoiding an explicit cumulative update from c_k while retaining the code history. Fixed round-wise coefficients w_k control the residual write-back. Although LaWM remains frozen, gradients through each round-dependent prediction train the modules that generate subsequent queries. After K rounds, z_K combines the initial VLM context with a sequence of future-guided revisions and serves as a central conditioning signal for host action generation.

3.3 Host-Policy Action Learning

After K rounds, the host action interface receives the refined plan:

$$\hat{y}_{\mathrm{host}} = \mathcal{D}_{\mathrm{host}}(z_K; \chi_t), \tag{4}$$

where χ_t collects host-specific context other than the latent plan. Here $\hat{y}_{\mathrm{host}}$ denotes the host's supervised prediction: an action chunk for OpenVLA and a flow-matching velocity for GR00T N1.5. At deployment, the former is regressed in parallel, whereas the latter is integrated by the host-native flow procedure. In both cases, the host interface combines z_K with its native context χ_t to produce an H-step action chunk.

Supervised training combines the host objective with compact code-space supervision:

$$\mathcal{L}_{\mathrm{SFT}} = \lambda_{\mathrm{host}}\,\ell_{\mathrm{host}}(\hat{y}_{\mathrm{host}}, y^{\star}_{\mathrm{host}}) + \lambda_{\mathrm{init}}\,\mathrm{MSE}(c_0, c^{\star}) + \lambda_{\mathrm{ref}}\,\mathrm{MSE}(c_{K+1}, c^{\star}). \tag{5}$$

The host term trains the refined plan for downstream control, while the initial and terminal code terms calibrate the LaWM interface before and after refinement. Implementation details, loss weights, and benchmark-specific settings are given in Appendix B.

3.4 Causal Refinement-Grouped Process-Reward RL

The supervised objective constrains the final action and endpoint codes, but does not directly rank alternative edits at an intermediate refinement state. We therefore introduce Causal Refinement-Grouped Process-Reward RL (CRG-PRL). Rather than optimizing a multi-round return, CRG-PRL treats each refinement round as a round-indexed local decision with $\gamma = 0$. Define the inner state $x_k = (z_{k-1}, h_k, c_k, k)$. RefineNet receives the first three components; the round index selects the prescribed write-back and exploration scales w_k and σ_k. We write its trainable bottleneck mean as $\mu_\theta(x_k)$ and freeze the following up-projection U_{up}. The resulting Gaussian bottleneck policy and post-edit transition are

$$\begin{aligned} b_k \sim \pi^{b}_{\theta}(\cdot \mid x_k) = \mathcal{N}\big(\mu_\theta(x_k), \sigma_k^2 I\big)\,, \qquad \delta_k = U_{\mathrm{up}} b_k, \\ z_k = z_{k-1} + w_k \delta_k, \qquad c_{k+1} = \mathrm{sg}(c_0) + \mathrm{Composer}(z_k, c_k). \end{aligned} \tag{6}$$

Post-edit process reward. The pre-edit prediction $\hat{u}_k = F_\omega(u_t, c_k)$ is formed before b_k is sampled and is therefore action-independent within a same-state group. CRG-PRL instead scores the consequence induced by the post-edit code c_{k+1}. Let

$$\hat{u}^{\mathrm{rew}}_{k,1} = F_\omega(u_t, c_{k+1}), \qquad \hat{u}^{\mathrm{rew}}_{k,2} = F_\omega(\hat{u}^{\mathrm{rew}}_{k,1}, c_{\mathrm{null}}). \tag{7}$$

Here c_{null} is the learned no-code slot. The frozen renderer converts these two supported hops into the ordered dual-view sequence $O_k = [o^{\mathrm{wm}}_t, \hat{o}^{\mathrm{rew}}_{k,1}, \hat{o}^{\mathrm{rew}}_{k,2}]$. With ζ_t collecting the fixed observation, instruction, and reward-side context, the causal assignment is

$$\begin{aligned} b_k \to \delta_k \to z_k \to c_{k+1} \to (\hat{u}^{\mathrm{rew}}_{k,1}, \hat{u}^{\mathrm{rew}}_{k,2}) \to O_k \to r_k, \\ r_k = \mathcal{R}_{\mathrm{proc}}(\zeta_t, c_{k+1}). \end{aligned} \tag{8}$$

For $k = K$, the same construction uses c_{K+1} only in the training-time reward branch; the host action remains decoded from z_K. The complete renderer and Robometer definitions are given in Appendix D.3.

Same-state local branching. Each collection iteration first runs a deterministic base path under a rollout snapshot θ_{roll}, producing shared states $\{x^{\mathrm{base}}_k\}_{k=1}^{K}$. At every base state, we sample M one-round branches:

$$\begin{aligned} b^{(i)}_k &= \mu_{\theta_{\mathrm{roll}}}(x^{\mathrm{base}}_k) + \sigma_k \eta^{(i)}_k, \qquad \eta^{(i)}_k \sim \mathcal{N}(0, I), \\ z^{(i)}_k &= z^{\mathrm{base}}_{k-1} + w_k U_{\mathrm{up}} b^{(i)}_k, \\ c^{(i)}_{k+1} &= \mathrm{sg}(c_0) + \mathrm{Composer}\Big(z^{(i)}_k, c^{\mathrm{base}}_k\Big), \\ r^{(i)}_k &= \mathcal{R}_{\mathrm{proc}}\Big(\zeta_t, c^{(i)}_{k+1}\Big), \qquad i = 1, \dots, M. \end{aligned} \tag{9}$$

A branch is evaluated for one round and is not propagated into the next base state. All candidates in the group therefore share the pre-edit state, refinement round, and reward-side randomness; only the sampled bottleneck edit varies. Their rewards consequently provide a controlled within-model comparison of local edit quality.

Group-relative policy update. For each same-state group, let $\bar{r}_k$ and s^r_k denote the population mean and standard deviation of its M rewards. The group-relative advantage and compact policy objective are

$$\begin{aligned} \bar{r}_k &= \frac{1}{M}\sum_{i=1}^{M} r^{(i)}_k, \qquad s^r_k = \sqrt{\frac{1}{M}\sum_{i=1}^{M}\Big(r^{(i)}_k - \bar{r}_k\Big)^2}, \\ A^{(i)}_k &= \mathrm{sg}\left[\frac{r^{(i)}_k - \bar{r}_k}{s^r_k + \varepsilon}\right], \qquad \mathcal{L}_{\mathrm{PG}} = -\mathbb{E}_{k,i}\Big[A^{(i)}_k \log \pi^{b}_{\theta}\Big(\mathrm{sg}(b^{(i)}_k) \mid \mathrm{sg}(x^{\mathrm{base}}_k)\Big)\Big]. \end{aligned} \tag{10}$$

Here $\varepsilon > 0$ stabilizes groups with near-zero reward spread. The same-state mean provides a local baseline without a learned critic, following group-relative credit assignment (Shao et al., 2024; Zheng et al., 2025). The complete objective adds an SFT edit-proximity term and a supervised host-action anchor:

$$\mathcal{L}_{\mathrm{CRG}} = \mathcal{L}_{\mathrm{PG}} + \lambda_{\mathrm{prox}}\mathcal{L}_{\mathrm{prox}} + \lambda_{\mathrm{act}}\mathcal{L}_{\mathrm{act}}. \tag{11}$$

CRG-PRL uses fresh on-policy buffers and updates only the pre-expansion layers of RefineNet. Reward-side branching is training-only, and deployment follows the deterministic mean path. Here "causal" denotes a controlled model-internal intervention that holds the pre-edit state and frozen proxy pipeline fixed while varying only the bottleneck edit; it does not assert physical causality in the real environment. Full estimator, numerical, freezing, calibration, and guardrail details are provided in Appendix D.

4 Experiments

4.1 Experimental Setup

LIBERO and LIBERO-Plus. For the OpenVLA-based instantiation, we evaluate PearlVLA on the standard LIBERO benchmark (Liu et al., 2023), which comprises four generalization suites: Spatial for spatial relations, Object for object-centric generalization, Goal for goal-conditioned execution, and Long for long-horizon task composition. We train on the RLDS-formatted modified LIBERO dataset (Kim et al., 2024), which repackages the official human-teleoperated demonstrations from the four suites. Each suite contains 10 tasks and 50 expert demonstrations per task (500 per suite), and provides camera images, robot states, task instructions, and continuous delta end-effector actions. Unless otherwise stated, we follow the standard suite-specific protocol: one policy is trained separately for each suite, and the four suite success rates are averaged after evaluation. Task success rate is the primary metric. We further evaluate robustness on LIBERO-Plus (Fei et al., 2025), which extends the four suites with 10,030 perturbed evaluation episodes spanning controlled variations in layout, camera viewpoint, robot state, language, lighting, background, and sensor noise.

RoboCasa. For the GR00T-based instantiation, we evaluate PearlVLA on the standard 24-task RoboCasa kitchen benchmark (Nasiriany et al., 2024) under the 300-demonstration-per-task protocol, training one multi-task policy over all 24 tasks and evaluating 50 rollouts per task. We report task success rates for Pick-and-Place, Open-or-Close, Others, and all tasks.

Policy instantiations. The LIBERO experiments use an OpenVLA-7B policy following the OpenVLA-OFT design (Kim et al., 2024; 2025), with parallel eight-step action regression, whereas the RoboCasa experiments use GR00T N1.5 (Bjorck et al., 2025) with its native 16-step flow-matching action expert; both use $K = 3$ refinement rounds. During evaluation, each policy executes its complete predicted action chunk before the next policy query. For both instantiations, the LaWM, renderer, and reward branch share two views: third-person plus wrist for LIBERO, and external-left plus wrist for RoboCasa; the RoboCasa host VLM additionally consumes a second third-person view. We use frozen LaWM checkpoints pretrained on an equal mixture of action-free robot videos from LIBERO, LIBERO-90, and RoboCasa and then specialized to each benchmark, without using robot actions as model inputs; the renderer and Robometer (Liang et al., 2026) are used only for CRG-PRL reward construction and not at policy inference. Complete LaWM, policy-training, and implementation details are provided in Appendix A and Tables 6–7.

4.2 Results

LIBERO. PearlVLA uses the same OpenVLA-7B initialization and parallel continuous action-chunk output family as OpenVLA-OFT, but inserts $K = 3$ future-guided latent refinement rounds before its action head. The main table reports both the supervised PearlVLA model and the final model after CRG-PRL tuning. Although flow-matching action heads have been reported to outperform regression heads on standard VLA benchmarks at the cost of iterative inference (Wu et al., 2026; Physical Intelligence et al., 2025), PearlVLA keeps a parallel regression head and instead allocates additional inference compute to plan refinement before action decoding. We use OpenVLA-OFT (Kim et al., 2025) as the primary external reference because it is a strong LIBERO model with the same backbone initialization and action-output family. Table 1 also includes representative

VLA baselines as broader reference points; these methods differ in backbone, data mixture, action representation, and training recipe, so they should not be read as fully controlled comparisons.

Table 1: LIBERO suite-specific success rates (%). Blue shades mark the three highest distinct score levels within the regression/classification VLA group; the final row reports PearlVLA after CRG-PRL tuning.

Model	Spatial	Object	Goal	Long	Avg
Flow-matching or diffusion-based VLA					
FLOWER (Reuss et al., 2025)	97.5	99.1	96.1	94.9	96.9
VLANeXt (Wu et al., 2026)	99.0	99.2	96.6	94.6	97.4
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	97.0	99.0	98.0	96.0	97.5
ProgressVLA (Yan et al., 2026)	88.2	96.4	87.2	66.2	84.5
LaWAM (Chen et al., 2026)	99.4	99.6	98.4	97.0	98.6
Regression or classification-based VLA					
OpenVLA (Kim et al., 2024)	84.7	88.4	79.2	53.7	76.5
WorldVLA (Cen et al., 2025)	85.6	89.0	82.6	59.0	79.1
CoT-VLA (Zhao et al., 2025)	87.5	91.6	87.6	69.0	83.9
NORA (Hung et al., 2025)	92.2	95.4	89.4	74.6	87.9
RD-VLA (Tur et al., 2026)	92.0	99.0	96.0	84.8	93.0
UniVLA (Bu et al., 2025)	96.5	96.8	95.6	92.0	95.2
π_0-Fast (Pertsch et al., 2025)	96.4	96.8	88.6	60.2	85.5
π_0(reg) (Black et al., 2024)	97.8	98.2	94.6	90.2	95.2
OpenVLA-OFT (Kim et al., 2025)	97.6	98.4	97.9	94.5	97.1
PearlVLA (Ours)	99.2	**99.8**	98.2	96.6	98.5
PearlVLA + CRG-PRL (Ours)	**99.4**	**99.8**	**98.4**	**97.2**	**98.7**

Supervised PearlVLA improves over OpenVLA-OFT on all four suites, raising the average success rate from 97.1% to 98.5%. CRG-PRL further raises the average to 98.7%, with the largest absolute gain over OpenVLA-OFT on LIBERO-Long (2.7 points). Within the regression/classification group, the final model obtains the highest average and suite-specific success rates. Among all methods shown, it also edges the strongest flow-matching baseline, LaWAM, by 0.1 points on average. Since the broader baselines use different backbones and training recipes, these comparisons establish empirical rankings under the reported protocols rather than controlled attribution of every gain.

Controlled robustness on LIBERO-Plus. To assess robustness beyond the in-domain setting, we additionally evaluate supervised PearlVLA on LIBERO-Plus (Fei et al., 2025), which extends the four LIBERO suites with 10,030 perturbed evaluation episodes spanning controlled variations in layout, camera viewpoint, robot state, language, lighting, background, and sensor noise.

Table 2: LIBERO-Plus robustness under controlled perturbation settings. Results are shown in success rate (%). PearlVLA results are averaged over perturbation suites and three inference seeds.

Model	Camera	Robot	Language	Light	Background	Noise	Layout	Total
OpenVLA (Kim et al., 2024)	0.8	3.5	23.0	8.1	34.8	15.2	28.5	15.6
π_0-Fast (Pertsch et al., 2025)	65.1	21.6	61.0	73.2	73.2	74.4	68.8	61.6
OpenVLA-OFT (Kim et al., 2025)	56.4	31.9	79.5	88.7	93.3	75.8	74.2	69.6
PearlVLA (Ours)	70.4	48.3	78.9	94.4	92.9	79.8	79.3	76.3

PearlVLA improves over OpenVLA-OFT from 69.6% to 76.3% total success. The largest gains occur under camera and robot-state perturbations. These results complement standard LIBERO and suggest that latent refinement improves generalization under controlled perturbations beyond the in-domain protocol.

RoboCasa. To test whether the refinement mechanism is tied to the regression head and meta-query interface used on LIBERO, we instantiate PearlVLA on GR00T N1.5 (Bjorck et al., 2025), where a frozen VLM conditions a flow-matching action expert. The host-specific plan readout

and clean-prefix injection adapt the interface while retaining the same plan–code–prediction loop (Appendix B).

Our controlled host-policy baseline is vanilla GR00T N1.5 trained on the same supervised RoboCasa data, but without PearlVLA's pretrained LaWM or refinement modules. Two recent methods are included only as broader, non-controlled references.

Table 3: Results on the standard RoboCasa Kitchen 24-task benchmark with 300 demonstrations per task. Cited rows are reported by prior work; GR00T N1.5 and PearlVLA variants are evaluated in our implementation.

Method	Pick-and-Place	Open-or-Close	Others	All
Video Policy (Liang et al., 2025)	47.2	83.0	70.0	65.7
GR00T N1.5 + HAMLET (Koo et al., 2025)	48.5	87.7	68.0	66.4
GR00T N1.5 (vanilla)	51.0	82.3	66.8	65.4
PearlVLA (K=1)	50.3	86.0	68.2	66.7
PearlVLA (K=3)	52.5	88.0	67.2	67.5
PearlVLA (K=3) + CRG-PRL	53.0	87.7	67.8	**67.8**

PearlVLA improves over vanilla GR00T N1.5 from 65.4% to 66.7% with one refinement round and 67.5% with the default three rounds. CRG-PRL further raises the overall success rate to 67.8%. The gains at both evaluated refinement depths indicate that the same future-guided plan refinement remains useful when the final plan conditions a flow-matching action expert rather than a parallel regression head.

4.3 Ablations and Mechanistic Analysis

To separate fixed conditioning from plan-conditioned predictive feedback, we evaluate controlled supervised OpenVLA regression policies on LIBERO. The fixed-VLM control uses GR00T N1.5 output only as round-invariant RefineNet keys and values; the one-shot LaWM control concatenates latent tokens from one prediction, and the cached-LaWM control reuses one prediction across refinement rounds. PearlVLA instead re-queries LaWM from the current plan-derived code. Each value of K denotes a separately trained fixed-depth model.

Table 4: **Controlled mechanism ablation on LIBERO.** All variants use the OpenVLA regression policy under the same training and evaluation protocol, without CRG-PRL. Q_{LaWM} counts LaWM evaluations per policy query. The gray row marks the default supervised PearlVLA model. Results are four-suite average success rates (%).

Variant	Integration	Auxiliary signal	Signal update	K	Q_{LaWM}	Avg. ↑
Direct control	Direct decoding	None	–	0	0	96.8
Fixed-VLM control	Refiner K/V	GR00T N1.5 VLM output	Fixed	1	0	97.2
Fixed-VLM control	Refiner K/V	GR00T N1.5 VLM output	Fixed	3	0	97.3
One-shot LaWM	Latent-token concat.	LaWM prediction	Once	0	1	97.7
Cached-LaWM control	Refiner K/V	LaWM prediction	Once, then reused	1	1	97.6
Cached-LaWM control	Refiner K/V	LaWM prediction	Once, then reused	3	1	97.9
PearlVLA	Refiner K/V	LaWM prediction	From current plan	1	2	97.8
PearlVLA	Refiner K/V	LaWM prediction	From current plan	3	4	**98.5**
PearlVLA	Refiner K/V	LaWM prediction	From current plan	5	6	98.4

Direct decoding obtains 96.8%. From $K = 1$ to $K = 3$, the fixed-VLM control changes from 97.2% to 97.3%, and the cached-LaWM control from 97.6% to 97.9%, whereas plan-conditioned feedback improves from 97.8% to 98.5%. Its advantage over cached feedback therefore grows from 0.2 to 0.6 points, consistent with additional rounds being most useful when the evolving plan changes the next predictive query. One-shot latent-token concatenation reaches 97.7%, while the

$K = 5$ PearlVLA model reaches 98.4%, suggesting saturation around three rounds. Because plan-conditioned variants also use more LaWM evaluations than the cached controls, this is an end-to-end mechanism comparison rather than a compute-matched causal estimate.

Inference cost. The default $K = 3$ loop increases policy-query latency from 156.6 to 198.7 ms on OpenVLA and from 59.0 to 101.0 ms on GR00T N1.5, while adding 391 GFLOPs in each case. Peak memory increases by 0.9 and 0.8 GB, respectively, at the first refinement round and remains unchanged through $K = 3$; the renderer and reward model are training-only. Full K-wise measurements are reported in Tables 8 and 9.

5 Conclusion

We introduced PearlVLA, a VLA framework that turns a one-shot latent world-action interface into a closed deliberation loop. Rather than repeatedly processing a fixed condition or revising actions that have already been decoded, PearlVLA refines a latent plan before action generation. Each revision changes the consequence considered by the next round, while the host action interface is invoked only after deliberation is complete. We further introduced CRG-PRL, which provides process-level post-training for the intermediate revisions themselves.

Across LIBERO, LIBERO-Plus, and RoboCasa, PearlVLA improves both regression- and flow-based host policies and performs competitively with strong contemporary methods. Controlled comparisons show the strongest gains when the modeled consequence evolves with the plan, rather than remaining fixed or being reused across rounds, while our efficiency analysis makes the performance–cost trade-off explicit. Together, these results position consequence-guided latent plan revision as a complementary direction to recurrent latent computation, increasingly expressive action decoders, and action-space search.

The present study uses a fixed refinement depth and benchmark-specialized LaWM checkpoints trained on a limited robot-video mixture. Adaptive halting could allocate deliberation on demand. More broadly, progress in large-scale representation-space video prediction and latent action world modeling suggests that pretraining on substantially broader action-free interaction video could yield a more transferable predictive interface across tasks and environments (Assran et al., 2025; Garrido et al., 2026). Validating this direction in contact-rich real-world manipulation remains important.

AI USE STATEMENT

Generative AI tools were used only to aid and polish the writing, identify potentially relevant related work, and check the consistency of mathematical derivations written by the authors. All AI-assisted output and cited sources were reviewed by the authors, who take full responsibility for the final content.

ETHICS STATEMENT

This work involves no new human-subject study, personally identifiable information, or sensitive data, and we are not aware of ethics concerns specific to this research.

REPRODUCIBILITY STATEMENT

Section 4.1 specifies the benchmark and evaluation protocols, while Appendices A, B, C, and D provide the LaWM training details, policy architectures and hyperparameters, inference-cost measurements, and complete CRG-PRL formulation and implementation settings, respectively.

REFERENCES

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, et al. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. pi_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. Univla: Learning to act anywhere with task-centric latent actions. *arXiv preprint arXiv:2505.06111*, 2025.

Jun Cen, Chaohui Yu, Hangjie Yuan, Yuming Jiang, Siteng Huang, Jiayan Guo, Xin Li, Yibing Song, Hao Luo, Fan Wang, et al. Worldvla: Towards autoregressive action world model. *arXiv preprint arXiv:2506.21539*, 2025.

Jialei Chen, Kai Wang, Kang Chen, Shuaihang Chen, Feng Gao, Wenhao Tang, Zhiyuan Li, Weilin Liu, Zhuyu Yao, Boxun Li, et al. Lawam: Latent world action models for efficient dynamics-aware robot policies. *arXiv preprint arXiv:2606.15768*, 2026.

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.

Yilun Du, Sherry Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Josh Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation. *Advances in neural information processing systems*, 36:9156–9172, 2023.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, et al. Libero-plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025.

Quentin Garrido, Tushar Nagarajan, Basile Terver, Nicolas Ballas, Yann LeCun, and Michael Rabbat. Learning latent action world models in the wild. In *Forty-third International Conference on Machine Learning*, 2026.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Yuhang Huang, Jiazhao Zhang, Shilong Zou, Xinwang Liu, Ruizhen Hu, and Kai Xu. Ladi-wm: A latent diffusion-based world model for predictive manipulation. *arXiv preprint arXiv:2505.11528*, 2025.

Chia-Yu Hung, Qi Sun, Pengfei Hong, Amir Zadeh, Chuan Li, U Tan, Navonil Majumder, Soujanya Poria, et al. Nora: A small open-sourced generalist vision language action model for embodied tasks. *arXiv preprint arXiv:2504.19854*, 2025.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Myungkyu Koo, Daewon Choi, Taeyoung Kim, Kyungmin Lee, Changyeon Kim, Younggyo Seo, and Jinwoo Shin. Hamlet: Switch your vision-language-action model into a history-aware policy. *arXiv preprint arXiv:2510.00695*, 2025.

Hengtao Li, Pengxiang Ding, Runze Suo, Yihao Wang, Zirui Ge, Dongyuan Zang, Kexian Yu, Mingyang Sun, Hongyin Zhang, Donglin Wang, et al. Vla-rft: Vision-language-action reinforcement fine-tuning with verified rewards in world simulators. *arXiv preprint arXiv:2510.00406*, 2025.

Anthony Liang, Yigit Korkmaz, Jiahui Zhang, Minyoung Hwang, Abrar Anwar, Sidhant Kaushik, Aditya Shah, Alex S Huang, Luke Zettlemoyer, Dieter Fox, et al. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons. *arXiv preprint arXiv:2603.02115*, 2026.

Junbang Liang, Pavel Tokmakov, Ruoshi Liu, Sruthi Sudhakar, Paarth Shah, Rares Ambrus, and Carl Vondrick. Video generators are robot policies. *arXiv preprint arXiv:2508.00795*, 2025.

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.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

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.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. In *International Conference on Learning Representations*, volume 2024, pp. 1862–1874, 2024.

Moritz Reuss, Hongyi Zhou, Marcel Rühle, Ömer Erdinç Yağmurlu, Fabian Otto, and Rudolf Lioutikov. Flower: Democratizing generalist robot policies with efficient vision-language-action flow policies. *arXiv preprint arXiv:2509.04996*, 2025.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*, 2025.

Dominik Schmidt and Minqi Jiang. Learning to act without actions. *arXiv preprint arXiv:2312.10812*, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. Dinov3. *arXiv preprint arXiv:2508.10104*, 2025.

Haoming Song, Delin Qu, Yuanqi Yao, Qizhi Chen, Qi Lv, Yiwen Tang, Modi Shi, Guanghui Ren, Maoqing Yao, Bin Zhao, et al. Hume: Introducing system-2 thinking in visual-language-action model. *arXiv preprint arXiv:2505.21432*, 2025.

Yue Su, Sijin Chen, Haixin Shi, Mingyu Liu, Zhengshen Zhang, Ningyuan Huang, Weiheng Zhong, Zhengbang Zhu, Yuxiao Liu, and Xihui Liu. World guidance: World modeling in condition space for action generation. *arXiv preprint arXiv:2602.22010*, 2026.

Zhenguo Sun, Yu Sun, Hande Huang, and Alois Knoll. ω-EVA: Envision, verify, and act with latent interactive world models. *arXiv preprint arXiv:2606.09457*, 2026.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Yalcin Tur, Jalal Naghiyev, Haoquan Fang, Wei-Chuan Tsai, Jiafei Duan, Dieter Fox, and Ranjay Krishna. Recurrent-depth vla: Implicit test-time compute scaling of vision-language-action models via latent iterative reasoning. *arXiv preprint arXiv:2602.07845*, 2026.

Xiao-Ming Wu, Bin Fan, Kang Liao, Jian-Jian Jiang, Runze Yang, Yihang Luo, Zhonghua Wu, Wei-Shi Zheng, and Chen Change Loy. Vlanext: Recipes for building strong vla models. *arXiv preprint arXiv:2602.18532*, 2026.

Hongyu Yan, Qiwei Li, Jiaolong Yang, and Yadong Mu. Progressvla: Progress-guided diffusion policy for vision-language robotic manipulation. *arXiv preprint arXiv:2603.27670*, 2026.

Michał Zawalski, William Chen, Karl Pertsch, Oier Mees, Chelsea Finn, and Sergey Levine. Robotic control via embodied chain-of-thought reasoning. *arXiv preprint arXiv:2407.08693*, 2024.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, et al. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 1702–1713, 2025.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Fangqi Zhu, Zhengyang Yan, Zicong Hong, Quanxin Shou, Xiao Ma, and Song Guo. Wmpo: World model-based policy optimization for vision-language-action models. *arXiv preprint arXiv:2511.09515*, 2025.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pp. 2165–2183. PMLR, 2023.

A LATENT WORLD MODEL

A.1 LAWM

PearlVLA uses the frozen forward decoder of a latent action model as its latent world model (LaWM). A frozen DINOv3 ViT-B/16 visual encoder E_ψ (Siméoni et al., 2025) maps the two-view observations o_t^{wm} and $o_{t+\Delta}^{\text{wm}}$ into features u_t and $u_{t+\Delta}$. The inverse-dynamics encoder q_φ and forward-dynamics decoder F_ω are trained jointly:

$$c \sim q_\varphi(c \mid u_t, u_{t+\Delta}), \qquad \hat{u}_{t+\Delta} = F_\omega(u_t, c). \tag{12}$$

The encoder compresses the observed visual transition into a continuous 16-dimensional latent action code, while the decoder predicts the corresponding future features from the current features and that code. After pretraining, both q_φ and F_ω are frozen: q_φ supplies the hindsight teacher code $c^\star$ during policy training, while F_ω is queried as LaWM; robot actions are not among its inputs. A separately trained renderer R_ξ, written here as its complete inference map, predicts a future VAE latent from the current-frame VAE latent and a LaWM visual subgoal:

$$v_t = E_{\text{vae}}(o_t^{\text{wm}}), \qquad \hat{v}_{t+\Delta} = R_\xi(v_t, \hat{u}_{t+\Delta}), \qquad \hat{o}_{t+\Delta} = D_{\text{vae}}(\hat{v}_{t+\Delta}). \tag{13}$$

For a second hop, $\hat{v}_{t+\Delta}$ becomes the renderer's present input. Rendering is used only for visualization and reward computation, not during policy deployment.

A.2 LATENT-CODE GEOMETRY AND RENDERED PREDICTIONS

Figure 3 provides two qualitative checks on the predictive interface. The left panel projects trajectories of the VLM foresight code c_0 and hindsight teacher code $c^\star$ into three principal components. Their separation visualizes the partial-information gap: the VLM infers c_0 from the current observation and instruction, whereas $c^\star$ is encoded from the observed transition $(u_t, u_{t+\Delta})$. The right panel queries the same frozen LaWM and renderer with $c^\star$, c_0, and an intermediate candidate. The resulting predictions are distinct yet locally coherent, which is consistent with code sensitivity and local coherence in the frozen predictive interface. This visualization motivates repeated plan-conditioned queries; performance along the actual refinement sequence is established by the refinement ablations rather than by the visualization alone.

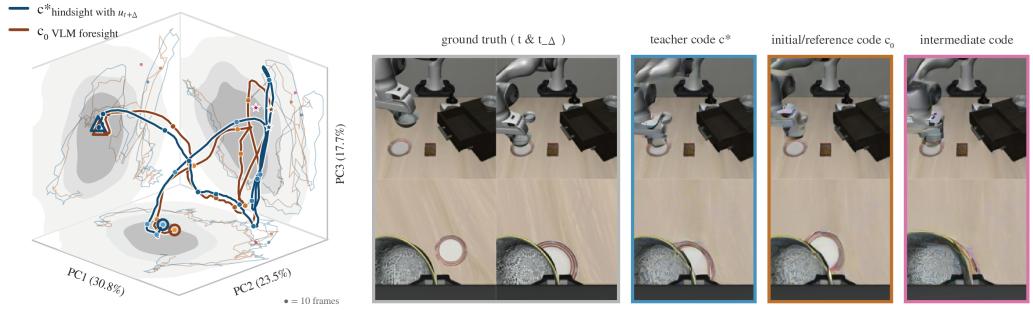


Figure 3: **Geometry of the latent action code space and rendered visual transitions.** *Left:* Three-dimensional PCA trajectories of the hindsight teacher code $c^\star$ and VLM foresight code c_0. *Right:* Ground-truth observations, followed by visual subgoals predicted by the frozen LaWM F_ω under $c^\star$, c_0, and an intermediate candidate code, then rendered by R_ξ.

A.3 TRAINING CONFIGURATION

LaWM pretraining first uses an equal mixture of LIBERO, LIBERO-90, and RoboCasa videos, and then specializes separate checkpoints for LIBERO and RoboCasa. The training objective combines horizon-feature prediction with a weak KL regularizer. Both LaWM and the renderer are trained with a two-hop extension. For LaWM, the first-hop prediction becomes the next current feature and 50% of second-hop conditions use c_{null}. For the renderer, the first predicted VAE latent becomes the next "present" input, paired with the second-hop subgoal predicted by LaWM with c_{null}. Table 5 summarizes the configuration.

Table 5: Training configuration of LaWM and the subgoal renderer.

Setting	Value
LaWM pretraining	
Pretraining data	LIBERO, LIBERO-90, and RoboCasa videos; no robot actions as model inputs
Training curriculum	30K mixed; then to 50K (LIBERO) / 60K (RoboCasa)
Visual input	Two 224×224 views; 16-frame offset
Backbone and code	Frozen DINOv3 ViT-B/16; continuous 16-D code
IDM / FDM	12-layer encoder / 16-layer residual AdaLN decoder (width 768)
Objective	DINO-feature MSE $+ 10^{-5}$ KL
Two-hop extension	First-hop prediction reused as the next current feature; 50% use c_{null}
Optimization	AdamW; learning rate 2×10^{-4}; global batch size 72
Subgoal renderer	
Model and objective	Frozen SDXL VAE (Podell et al., 2024); 12-layer DiT; future VAE-latent noise prediction
Two-hop extension	First predicted VAE latent reused as the next present input; paired with a c_{null}-conditioned LaWM subgoal
Optimization	AdamW; learning rate 2×10^{-4}; global batch size 64; 60K steps

B Policy Architecture and Implementation Details

Shared refinement modules. Composer is implemented as cross-attention whose query combines a normalized refinement state with a projection of the current code and attends to the latent-plan tokens. FutureEncoder is a two-layer pre-norm Transformer that applies self-attention over visual tokens. RefineNet projects the plan state to width 384 and adds a learned projection of c_k to each plan-token query before applying four blocks of self-attention, gated cross-attention to the future tokens, and an MLP. The pooled future summary provides AdaLN conditioning, and a linear up-projection returns one 4096-dimensional residual for each plan token.

B.1 OpenVLA / LIBERO

Table 6: Core supervised-policy settings for LIBERO.

Hyperparameter	Value
Optimization	
Optimizer	AdamW
Learning rate	2×10^{-4}, cosine decay to $0.05\times$ the base rate
Global batch size	32
Training steps	20K for Spatial/Object; 24K for Goal/Long
Training objectives	
Action regression	Normalized action-chunk MSE, scaled by 25
Initial-code distillation	$c_0 = C(\tilde{z}_0)$ vs. target $c^\star$, MSE scaled by 0.4
Refined-code supervision	Terminal c_{K+1} (c_4 for $K = 3$) vs. target $c^\star$, MSE scaled by 0.1
Inputs and action output	
Base VLA	OpenVLA-7B with LoRA rank 16
Observation inputs	Third-person and wrist images, language, and 8-step proprioception
Image resolution	224×224 pixels per view
Image augmentation	93.3% random crop and color jitter
Action output	7-dimensional actions in an 8-step chunk
Action decoder	Query-transformer regression head
Latent refinement	
Refinement rounds	$K = 3$
Loop-state perturbation	c_0 and $\hat{u}_0$ from clean $\tilde{z}_0$; writable z_0 uses a small Gaussian perturbation with $\rho_{\text{init}} = 0.1$
Plan update	Fixed round-wise scalars with nominal update RMS of 15%, 6%, and 3%
Compute	
Hardware	$4\times$ NVIDIA A100 40GB GPUs

PearlVLA adopts the OpenVLA-OFT architecture (Kim et al., 2024; 2025): an OpenVLA-7B backbone with fused SigLIP–DINOv2 features (Oquab et al., 2023), an LLaMA-2 7B language model (Touvron et al., 2023), and a three-layer GELU MLP projector, adapted via LoRA for parallel action-chunk regression. A transformer-based proprio encoder converts proprioceptive history into tokens, which are fused with the visual and language inputs in the multimodal sequence `[BOS] [vision] [proprio] [text] [meta_query]`. Eight learnable meta-query tokens use a prefix-LM attention mask, and their final-layer representations form the clean initial plan $\tilde{z}_0$.

$c_0 = C(\tilde{z}_0)$ and $\hat{u}_0 = \mathrm{sg}(F_\omega(u_t, c_0))$ are computed from the clean plan; all paths from c_0 into the refinement loop use $\mathrm{sg}(c_0)$, and only the writable loop state z_0 is perturbed ($\rho_{\mathrm{init}} = 0.1$). Applied after the trainable VLM, this perturbation discourages supervised optimization from bypassing refinement through VLM adaptation. Over K rounds, the code queries the frozen LaWM for a DINOv3 latent subgoal, and the refiner applies scheduled residual updates from the plan, the subgoal, and its change relative to the detached initial prediction $\hat{u}_0$. An AdaLN-conditioned query transformer then regresses the eight-step action chunk in parallel. Settings are reported in Table 6.

B.2 GR00T N1.5 / RoboCasa

For RoboCasa, PearlVLA is built on GR00T N1.5 (Bjorck et al., 2025). Its Eagle-2.5 VLM, comprising a SigLIP vision encoder and a Qwen3 language model, remains frozen; no LoRA is applied and no VLM weight is updated. A lightweight query transformer uses sixteen learnable plan queries to cross-attend to the frozen VLM's instruction-token states following the three image-token blocks, producing sixteen latent-plan tokens. A Q-Former-style readout maps these tokens to the same 16-dimensional LaWM code space. The plan is refined through the same K-round LaWM loop used by the OpenVLA instantiation.

Action decoding uses the native GR00T N1.5 flow-matching expert, initialized from its pretrained weights and fully fine-tuned. The refined plan is projected to the expert width and inserted as a separate clean prefix beside the proprioceptive prefix. Noised action queries remain the variables of the flow-matching objective, conditioned on the frozen VLM output through cross-attention. The interpolation schedule, flow-time conditioning, velocity target, and four-step Euler integrator are unchanged, and the expert outputs a 16-step, 12-dimensional mobile-manipulation action chunk.

During training, the complete VLM key-value sequence is replaced by a shared learned null sequence with probability 0.10. This sample-level dropout keeps the pretrained expert from ignoring the latent-plan prefix. Dropout is disabled at inference. Table 7 summarizes the training settings.

Table 7: Core supervised-policy settings for RoboCasa.

Hyperparameter	Value
Optimization	
Optimizer	AdamW
Learning rate	1×10^{-4}, followed by cosine decay to $0.05\times$ the base rate
Global batch size	32
Training steps	60K
Training objectives	
Action regression	Flow-matching velocity MSE, scaled by 10
Initial-code distillation	MSE between c_0 and the 16-dim target code $c^\star$, scaled by 0.4
Refined-code supervision	MSE between terminal c_{K+1} (c_4 for $K = 3$) and $c^\star$, scaled by 0.1
Inputs and action output	
Base VLA	GR00T N1.5 with a frozen Eagle-2.5 VLM; no LoRA; fully fine-tuned flow-matching action expert
Observation inputs	Two third-person images, one wrist image, language instruction, and proprioceptive state
Image resolution	224×224 pixels per view
Image augmentation	93.3% random crop and color jitter
Latent-code readout	Q-Former-style readout from 16 latent plan tokens to a 16-dim code
Action output	12-dimensional mobile-manipulation actions in a 16-step action chunk
Action decoder	Flow-matching action expert
VLM-KV condition dropout	Training only: sample-level probability 0.10 of replacing the complete VLM KV sequence with shared learned null tokens
Latent refinement	
Refinement rounds	$K = 3$
Plan update	Fixed round-wise scalars with nominal update RMS of 15%, 6%, and 3%
Compute	
Hardware	$4\times$ NVIDIA A100 40GB GPUs

C Computational Overhead and Inference Efficiency

C.1 Measurement Setup

All measurements were collected on one NVIDIA A100 40GB GPU with batch size one and PyTorch eager execution. The host VLM used BF16 autocast, and all remaining modules used FP32. The $K = 0$ configuration comprises one VLM forward pass and one action-decoding pass without DINOv3 feature extraction or latent refinement. For $K \geq 1$, DINOv3 feature extraction is performed once; the loop then computes the detached baseline prediction $\hat{u}_0$ and K live refinement predictions before the same single action decode. Thus K denotes refinement rounds, while the number of LaWM queries is $K + 1$. The subgoal renderer and reward model are excluded because neither is used during deployment; CRG-PRL adds no deployment query.

We report policy-query latency, peak GPU memory, GFLOPs, and effective action throughput, computed as $H \times 1000/\mathrm{latency}$. Consistent with our evaluation protocol, the complete predicted action chunk is executed without temporal ensembling before the next policy query. Effective action throughput is not the policy-query rate, and values from different action-chunk lengths are therefore not directly comparable.

C.2 Efficiency Results

Tables 8 and 9 report results for the OpenVLA-7B / LIBERO ($H = 8$) and GR00T N1.5 / RoboCasa ($H = 16$, four Euler steps) instantiations, respectively.

Table 8: Inference efficiency on OpenVLA-7B / LIBERO.

Configuration	Latency (ms) ↓	Throughput (Hz) ↑	Peak memory (GB) ↓	GFLOPs ↓
VLM + action head ($K = 0$)	156.6	51	15.4	6,487
PearlVLA ($K = 1$)	176.3	45	16.3	6,663
PearlVLA ($K = 2$)	187.5	43	16.3	6,770
PearlVLA ($K = 3$)	198.7	40	16.3	6,878

Table 9: Inference efficiency on GR00T N1.5 / RoboCasa.

Configuration	Latency (ms) ↓	Throughput (Hz) ↑	Peak memory (GB) ↓	GFLOPs ↓
VLM + action expert ($K = 0$)	59.0	271	5.4	2,089
PearlVLA ($K = 1$)	78.7	203	6.2	2,266
PearlVLA ($K = 2$)	89.9	178	6.2	2,373
PearlVLA ($K = 3$)	101.0	158	6.2	2,480

The three-round refinement loop adds about 42ms on both instantiations. The first refinement step accounts for about 20ms and 176 GFLOPs, covering one-time DINOv3 extraction, $\hat{u}_0$, and the first live LaWM query; each later round adds one live query and about 11ms and 107 GFLOPs. Peak memory rises by 0.9GB for OpenVLA and 0.8GB for GR00T, and does not grow with further rounds. With this overhead, LaWM-guided refinement provides effective action throughputs of 40Hz on OpenVLA and 158Hz on GR00T.

D Causal Refinement-Grouped Process-Reward RL

CRG-PRL optimizes the refinement operator already learned by the supervised PearlVLA policy. The reward is a proxy produced by a frozen LaWM, renderer, and robot process reward model. Its central design follows one principle: edits should only be compared under the *same* refinement state, the *same* round, and the *same* reward pipeline. This section defines the inner decision process, same-state local branching, post-edit process reward, and critic-free objective.

D.1 Inner MDP and Policy

For a fixed observation and instruction, let $\tilde{z}_0$ denote the clean initial plan and let $z_0 = \mathcal{I}_{\text{host}}(\tilde{z}_0)$ denote the writable loop state, where $\mathcal{I}_{\text{host}}$ applies the OpenVLA perturbation with scale $\rho_{\text{init}} = 0.1$ and is the identity for GR00T. The supervised loop initializes

$$\begin{aligned} &c_0 = C(\tilde{z}_0), \\ &\hat{u}_0 = \text{sg}(F_\omega(u_t, c_0)), \qquad c_1 = \text{sg}(c_0) + \text{Composer}(z_0, \text{sg}(c_0)). \end{aligned} \tag{14}$$

At round $k = 1, \ldots, K$, it queries $\hat{u}_k = F_\omega(u_t, c_k)$ and constructs the relative-imagination input and frozen future summary

$$\tau_k = [\hat{u}_k - u_t \,;\, \hat{u}_k - \hat{u}_0], \qquad h_k = \text{FutureEncoder}(\tau_k), \tag{15}$$

where $\hat{u}_0$ is the shared first-prediction reference. To avoid overloading the renderer parameter ξ, denote the inner state by

$$x_k = (z_{k-1}, h_k, c_k, k). \tag{16}$$

The round index makes the finite-horizon state explicit and selects the prescribed write-back and exploration scales introduced below; the shared RefineNet mean itself receives only (z_{k-1}, h_k, c_k). RefineNet maps this state to a bottleneck mean $\mu_\theta(x_k) \in \mathbb{R}^{N_p \times 384}$, where $N_p = 8$ for LIBERO and $N_p = 16$ for RoboCasa. CRG-PRL defines a fixed-variance Gaussian policy in this bottleneck:

$$b_k \sim \pi_\theta^b(\cdot \mid x_k) = \mathcal{N}\big(\mu_\theta(x_k), \sigma_k^2 I\big), \qquad \delta_k = U_{\text{up}} b_k, \tag{17}$$

where the frozen linear up-projection U_{up} maps the bottleneck to $N_p \times 4096$. The resulting transition is

$$z_k = z_{k-1} + w_k \delta_k, \qquad c_{k+1} = \text{sg}(c_0) + \text{Composer}(z_k, c_k). \tag{18}$$

Only the pre-expansion layers of RefineNet are updated during CRG-PRL; FutureEncoder and U_{up} remain frozen. Deployment disables exploration and uses $b_k = \mu_\theta(x_k)$.

The exploration scale retains one dimensionless hyperparameter across rounds. Let μ_{SFT} denote the frozen supervised RefineNet mean used as the calibration reference, and define

$$\nu_k = \text{EMA}[\text{RMS}(w_k U_{\text{up}} \mu_{\text{SFT}}(x_k))], \qquad g_{\text{up}} = \sqrt{\frac{\|U_{\text{up}}\|_F^2}{4096}}. \tag{19}$$

The round-specific scale is

$$\sigma_k = \frac{\sigma_{\text{norm}} \nu_k}{w_k g_{\text{up}}}. \tag{20}$$

The statistics ν_k are estimated once from frozen SFT rollouts before CRG-PRL, and the resulting σ_k values remain fixed throughout all updates. Thus the RMS of the realized write-back noise is approximately σ_{norm} times the aggregate natural SFT edit magnitude on the calibration set in every round; it is not a per-state equality. We use $\gamma = 0$, yielding a refreshed local-ranking surrogate at states along the deterministic deployment trajectory rather than an objective over multi-round returns.

D.2 Same-State Local Branching

Each outer iteration collects fresh data under the current rollout snapshot θ_{roll}. It first runs a deterministic base path:

$$\begin{aligned} b_k^{\text{base}} &= \mu_{\theta_{\text{roll}}}(x_k^{\text{base}}), \\ z_k^{\text{base}} &= z_{k-1}^{\text{base}} + w_k U_{\text{up}} b_k^{\text{base}}, \\ c_{k+1}^{\text{base}} &= \text{sg}(c_0) + \text{Composer}\big(z_k^{\text{base}}, c_k^{\text{base}}\big). \end{aligned} \tag{21}$$

At every base state, we draw $M = 8$ one-round branches,

$$b_k^{(i)} = \mu_{\theta_{\text{roll}}}(x_k^{\text{base}}) + \sigma_k \eta_k^{(i)}, \qquad \eta_k^{(i)} \sim \mathcal{N}(0, I), \quad i = 1, \ldots, M. \tag{22}$$

All branches in a group share the observation features, latent plan, committed code, first prediction, current imagined future, instruction, and frozen reward pipeline; only $b_k^{(i)}$ varies. A branch survives for one round and is used only to evaluate that edit. It is not propagated into the next base state, which prevents later-round groups from mixing different starting states. Dropout is disabled during collection, and shared reward-side randomness is used within each group.

The buffer stores detached base states, sampled bottlenecks, and advantages. It stores $b_k^{(i)}$ itself rather than reconstructing it from $\eta_k^{(i)}$ during optimization; otherwise the sampled action and current mean would share a gradient path and invalidate the score-function estimator. With $B = 8$, $K = 3$, and $M = 8$, each distributed update contains $BKM = 192$ conditional branches. The complete fresh buffer is used for one backward pass and one optimizer update, and is then discarded.

D.3 Causal Process Reward

The current imagined future $\hat{u}_k$ precedes the sampled bottleneck and is therefore constant within a group. CRG-PRL instead rewards the consequence predicted after the edit:

$$\begin{aligned} \delta_k^{(i)} &= U_{\text{up}} b_k^{(i)}, \\ z_k^{(i)} &= z_{k-1}^{\text{base}} + w_k \delta_k^{(i)}, \\ c_{k+1}^{(i)} &= \text{sg}(c_0) + \text{Composer}\left(z_k^{(i)}, c_k^{\text{base}}\right), \\ \hat{u}_{k,1}^{\text{rew},(i)} &= F_\omega(u_t, c_{k+1}^{(i)}), \\ \hat{u}_{k,2}^{\text{rew},(i)} &= F_\omega(\hat{u}_{k,1}^{\text{rew},(i)}, c_{\text{null}}). \end{aligned} \tag{23}$$

Here the hop index $j \in \{1, 2\}$ corresponds to physical time $t + j\Delta$. Starting from $v_t = E_{\text{vae}}(o_t^{\text{wm}})$, the frozen renderer and VAE decoder produce the reward images recursively:

$$\begin{aligned} \hat{v}_{k,0}^{\text{rew},(i)} &= v_t, \\ \hat{v}_{k,j}^{\text{rew},(i)} &= R_\xi\left(\hat{v}_{k,j-1}^{\text{rew},(i)}, \hat{u}_{k,j}^{\text{rew},(i)}\right), \\ \hat{o}_{k,j}^{\text{rew},(i)} &= D_{\text{vae}}\left(\hat{v}_{k,j}^{\text{rew},(i)}\right), \qquad j = 1, 2. \end{aligned} \tag{24}$$

Robometer (Liang et al., 2026) scores the resulting ordered three-frame, dual-view sequence conditioned on the instruction:

$$\begin{aligned} O_k^{(i)} &= \left[o_t^{\text{wm}}, \hat{o}_{k,1}^{\text{rew},(i)}, \hat{o}_{k,2}^{\text{rew},(i)}\right], \\ \left(p_{k,i}^{\text{prog}}, p_{k,i}^{\text{succ}}\right) &= \text{Robometer}_{\text{final}}\left(O_k^{(i)}, \ell\right), \\ r_k^{(i)} &= p_{k,i}^{\text{prog}} + 0.2\, p_{k,i}^{\text{succ}}. \end{aligned} \tag{25}$$

For LIBERO, this path uses the third-person and wrist views; for RoboCasa, it uses the external-left and wrist views. The second RoboCasa third-person view is consumed only by the host VLM. For the final edit, LaWM is queried with c_{K+1} only by the training-time reward branch; c_{K+1} also remains the terminal SFT target, while the host action is decoded from z_K. Reward rollout, rendering, and scoring run without gradients.

The term "causal" refers to this controlled model-internal intervention: the pre-edit state and frozen proxy pipeline are held fixed while only the bottleneck edit changes. It does not assert that the proxy reward establishes physical causality in the real environment. We therefore monitor latent-code support, reward spread, renderer stability, and final-action sensitivity to detect proxy exploitation.

D.4 Group-Relative Advantage

For each sample-round group, rewards are standardized using the population standard deviation:

$$\bar{r}_k = \frac{1}{M}\sum_{i=1}^{M} r_k^{(i)}, \qquad s_k^r = \sqrt{\frac{1}{M}\sum_{i=1}^{M}\left(r_k^{(i)} - \bar{r}_k\right)^2}, \qquad A_k^{(i)} = \text{sg}\left[\frac{r_k^{(i)} - \bar{r}_k}{s_k^r + 10^{-6}}\right]. \tag{26}$$

This same-state baseline centers additive reward offsets specific to the state and round without a learned critic. No explicit skip threshold is used: a group with identical rewards has zero advantages, while 10^{-6} stabilizes near-zero spread. Because both the mean and standard deviation are estimated from the same finite group, this is a biased group-normalized surrogate rather than an unbiased gradient estimator of raw expected reward. Its role is local ranking under a matched refinement state.

D.5 Group-Relative Policy Objective

Let B denote the global task-sample batch size and let $d_b = 384N_p$. Because the covariance is fixed, the bottleneck log-density is

$$\log \pi_\theta^b(b_k^{(i)} \mid x_k) = -\frac{\left\|b_k^{(i)} - \mu_\theta(x_k)\right\|_F^2}{2\sigma_k^2} - d_b \log \sigma_k - \frac{d_b}{2}\log(2\pi). \tag{27}$$

Restoring the task-sample index n, the fresh-buffer score-function objective is

$$\mathcal{L}_{\text{PG}} = -\frac{1}{BKM}\sum_{n=1}^{B}\sum_{k=1}^{K}\sum_{i=1}^{M} \text{sg}\left(A_{nk}^{(i)}\right) \log \pi_\theta^b\left(\text{sg}(b_{nk}^{(i)}) \mid \text{sg}(x_{nk}^{\text{base}})\right). \tag{28}$$

Event dimensions are summed to form one joint log-probability for each branch. The sampled bottleneck, base state, and advantage are detached; gradients flow only through the current policy mean. Because each buffer is collected from the current policy, used for one optimizer update, and then discarded, no importance ratio, clipping, replay, or value loss is required. This objective

uses the group-relative credit assignment of GRPO (Shao et al., 2024) without its completion-level sampling unit. The bottleneck mean is cast to FP32, Gaussian noise is sampled in FP32, and the resulting $b_k^{(i)}$ is stored in FP32. The bottleneck difference, squared-norm reduction, and Gaussian log-probability are likewise evaluated in FP32.

D.6 SFT Proximity, Action Anchor, and Total Objective

Two guardrails keep the optimized refiner close to supervised behavior. First, the deterministic current-policy and SFT plan edits at a saved base state are

$$\Delta z_k^{\theta} = w_k U_{\mathrm{up}} \mu_{\theta}(x_k), \qquad \Delta z_k^{\mathrm{SFT}} = w_k U_{\mathrm{up}} \mu_{\mathrm{SFT}}(x_k). \tag{29}$$

We penalize their realized difference:

$$\mathcal{L}_{\mathrm{prox}} = \frac{1}{K} \sum_{k=1}^{K} \mathbb{E}_{x_k} \left[\frac{\mathrm{RMS}^2 \left(\Delta z_k^{\theta} - \Delta z_k^{\mathrm{SFT}} \right)}{\nu_k^2} \right]. \tag{30}$$

This term is evaluated on saved deterministic base states and requires no new LaWM rollout. Policy-to-SFT KL remains a diagnostic rather than a loss.

Second, a fresh deterministic path is anchored to the supervised host target:

$$\mathcal{L}_{\mathrm{act}} = \ell_{\mathrm{host}} \left(\mathcal{D}_{\mathrm{host}}(z_K^{\theta}; \chi_t), y_{\mathrm{host}}^{\star} \right). \tag{31}$$

For OpenVLA, ℓ_{host} is the unscaled per-element mean action regression loss; for GR00T, it is the unscaled per-element mean flow-matching-velocity loss. The host decoder remains frozen, but gradients propagate through it to the refined plan. Removing the host-specific supervised multipliers of 25 and 10 prevents these implementation scales from being folded into the shared action-anchor weight. The GR00T anchor otherwise retains the host-native flow-matching construction; VLM-KV condition dropout is disabled throughout CRG-PRL.

On the fresh deterministic anchor path, frozen U_{up}, Composer, LaWM, FutureEncoder, and the host decoder retain their input Jacobians, so the action loss can reach earlier refinement rounds. Stop-gradient is applied only to c_0 and $\hat{u}_0$, the complete reward branch, and the stored rollout states, bottleneck samples, and advantages.

The total objective is

$$\mathcal{L}_{\mathrm{CRG}} = \mathcal{L}_{\mathrm{PG}} + 0.01\, \mathcal{L}_{\mathrm{prox}} + 0.1\, \mathcal{L}_{\mathrm{act}}. \tag{32}$$

The three terms are normalized independently: $\mathcal{L}_{\mathrm{PG}}$ over BKM branches, $\mathcal{L}_{\mathrm{prox}}$ over BK base states, and $\mathcal{L}_{\mathrm{act}}$ over B task samples and all action coordinates. In distributed training each GPU uses the corresponding local denominators $B_{\mathrm{local}}KM = 48$, $B_{\mathrm{local}}K = 6$, and $B_{\mathrm{local}} = 2$; DDP then averages the four local gradients. This is equivalent to the stated global means and avoids an additional factor-of-four shrinkage. Consequently, the loss weights do not change with the number of branches, rounds, or GPUs. Fixed covariance makes the entropy constant. Explicit SFT-KL and entropy terms are therefore omitted; policy-to-SFT KL is retained as a diagnostic.

D.7 Relation to Group-Relative LLM RL

CRG-PRL borrows group-relative, critic-free credit assignment from GRPO and GSPO (Shao et al., 2024; Zheng et al., 2025), but changes the decision and credit-assignment units. GRPO and GSPO compare complete prompt-conditioned completions or sequences; CRG-PRL compares one-round continuous bottleneck edits at a fixed refinement state. WMPO optimizes action trajectories inside a learned pixel world model and discards that model at deployment (Zhu et al., 2025), while VLA-RFT uses a frozen video world model to provide verified training rewards (Li et al., 2025). CRG-PRL instead optimizes an edit operator inside the deployed refinement loop. The renderer and Robometer are training-only, whereas LaWM remains part of policy inference.

D.8 Implementation Details

Table 10 summarizes the CRG-PRL configuration. Branches from the same round are evaluated in parallel and are not counted as additional task samples. Reward-side branching, rendering, and scoring are used only during training; deployment follows one deterministic three-round refinement path.

Table 10: Core settings of CRG-PRL.

Setting	Value
Refinement and branches	$K = 3$ rounds; $M = 8$ same-state branches per round
Bottleneck policy	LIBERO: 8×384; RoboCasa: 16×384; fixed isotropic covariance
Deterministic expansion	Frozen linear up-projection to 8×4096 or 16×4096
Exploration	$\sigma_{\text{norm}} = 0.06$, converted to round-specific σ_k by Eq. 20, pre-calibrated once and then fixed
Updated parameters	RefineNet except the final up-projection; learning rate 1×10^{-5}
Frozen parameters	Host VLM and interfaces, C, FutureEncoder, up-projection, Composer, w_k, LaWM, renderer, Robometer, host policy's action interface, and SFT RefineNet reference
Reward	Dual-view $[o_t^{\text{wm}}, \hat{o}_{k,1}^{\text{rew}}, \hat{o}_{k,2}^{\text{rew}}]$; final-frame progress $+ 0.2\times$ success from frozen Robometer
Loss weights	$\lambda_{\text{prox}} = 0.01$ and $\lambda_{\text{act}} = 0.1$; no value, explicit SFT-KL, or entropy loss
Group advantage	Population standard deviation plus 10^{-6}; no explicit group skipping
Policy update	Fresh-buffer group-relative policy gradient with $\gamma = 0$; no replay, importance ratio, or clipping
Optimization and compute	2000 outer iterations and optimizer updates; two task samples per GPU; $4\times$ NVIDIA A100 40GB GPUs; global rollout batch size 8
Diagnostics	Raw reward, per-round group standard deviation, policy-to-SFT KL, joint log-probability, policy-gradient norm, realized edit RMS, code-support distance, zero-advantage group rate, and the correlation between terminal c_{K+1} reward and host-action quality