MorphAct: Contextual Parameterization of Vision–Language–Action Models

Anonymous authors
Paper under double-blind review

Abstract

Vision–language–action (VLA) models seek to support diverse robot behaviors within a shared policy. Conventional fine-tuning, however, fits a single static update to operations whose perceptual and control demands change even within one task. When these operations favor competing parameter adjustments, fitting their mixture can leave individual operations insufficiently adapted. Our atomic-task diagnostics reveal competing update directions in both vision–language and action parameters, with source-improving updates increasing loss on other tasks. This motivates **MorphAct**[1], a contextual parameterization framework that *shares an adaptation rule rather than a fixed update*, allowing adaptation to specialize without splitting the policy into operation-specific experts. Two coupled generators produce context-dependent low-rank updates for the vision–language and action modules. Their coupling follows the policy's computational dependency: vision–language updates shape the representations that condition action-side parameter generation. Both generators are trained jointly through the native action objective, without operation labels or expert-adapter targets. MorphAct improves average success rates across three VLA backbones on LIBERO and Meta-World, with gains of up to 15.13%. In real-world evaluation, it achieves gains of up to 17% over π_0. Ablations support the benefits of contextual parameter generation and representation coupling.

1 Introduction

Vision–language–action (VLA) models combine pre-trained multimodal representations with action prediction to support diverse robot behaviors within a shared policy (Brohan et al., 2023a; Kim et al., 2025b; Black et al., 2025). Conventional adaptation uses full fine-tuning or low-rank updates, producing learned parameters that remain fixed across interaction contexts during execution (Hu et al., 2022; Kim et al., 2025a). Yet the demands on a policy vary not only across tasks but also within a single trajectory. For example, grasping requires motion relative to the object, whereas placement requires motion relative to its destination—a distinction explicitly exploited by object-centric demonstration generation (Mandlekar et al., 2023). A shared update must accommodate the training signals from these different operations, even when the task instruction remains unchanged. Prior work in multi-task learning shows that gradient interference can hinder joint optimization (Yu et al., 2020a), raising the question of how one policy can accommodate different adaptation demands across manipulation contexts. The issue is not whether a fixed policy can express diverse behaviors, but whether one shared static update adapts them effectively under a practical budget.

We investigate this issue using demonstrations from the 65 atomic tasks in RoboCasa365 (Nasiriany et al., 2026). At a common policy checkpoint, we examine gradient alignment, the effect of a single update on other tasks, and the contributions of individual parameter groups. The diagnostics reveal competing update directions in both the vision–language and action modules. Every source-improving update also increases loss on batches from other tasks. These findings motivate a shared adaptation mechanism that can produce different parameter updates for different manipulation contexts.

To address this challenge, we introduce **MorphAct**, a contextual parameterization framework that *shares an adaptation mechanism rather than a fixed parameter update* (Figure 1). It generates

[1] "Morph," rooted in Greek *morphē* ("form"), means "to transform."

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.

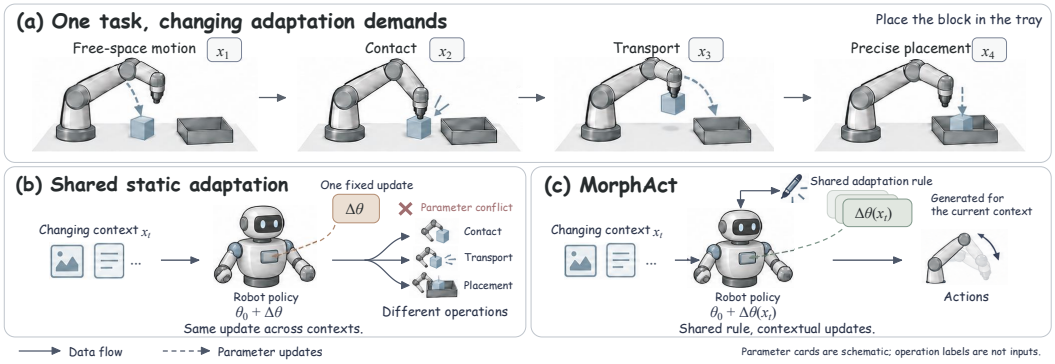


Figure 1: **Share the adaptation rule, not a fixed update.** A manipulation trajectory contains changing interaction contexts with different perceptual and control demands. Static adaptation applies one shared update across contexts, whereas MorphAct generates context-dependent updates for the vision–language and action modules through a shared adaptation mechanism. Operation labels are illustrative only and are not used as model inputs; parameter blocks are schematic.

context-dependent low-rank updates for the vision–language and action modules. Because adapting the vision–language module changes the representations used for control, MorphAct couples the two generators by conditioning action-side parameter generation on the adapted representations. Both generators are trained jointly through the native action objective, without operation labels or expert-adapter targets. We evaluate MorphAct on X-VLA, π_0, and $\pi_{0.5}$ (Zheng et al., 2026; Black et al., 2025; Physical Intelligence et al., 2025) using LIBERO (Liu et al., 2023) and Meta-World (Yu et al., 2020b), comparing each backbone's baseline with MorphAct. Across all three backbones, MorphAct raises Meta-World average success by 7.27–15.13 percentage points; on LIBERO, X-VLA, π_0, and $\pi_{0.5}$ gain 3.55, 2.50, and 0.50 points in mean success, respectively. Real-world evaluation with π_0 further shows gains of 13.00–17.00 percentage points across four manipulation tasks. Controlled X-VLA ablations on LIBERO show that fixed generation conditions, single-module adaptation, and action-side generation conditioned on pre-adaptation features each reduce mean success. Our analysis connects these results to the task-dependent adaptation demands identified in the diagnostic study.

Our contributions are threefold:

- We diagnose conflicting adaptation demands across atomic manipulation tasks through gradient alignment, one-step transfer, and parameter-group contributions.
- We introduce MorphAct, which generates context-dependent low-rank updates for the vision–language and action modules, connects their generation through adapted representations, and learns both generators jointly through the native action objective.
- We evaluate MorphAct on X-VLA, π_0, and $\pi_{0.5}$ using LIBERO and Meta-World, and on π_0 in four real-world manipulation tasks. Ablations examine the roles of contextual conditioning and representation coupling.

2 Conflicting Adaptation Demands in VLA Models

A shared VLA must support tasks with different perceptual and control demands. We examine how their learning signals interact and identify operation-level parameter conflicts in both vision–language and action parameters. The analysis proceeds from update directions to their effects: we first compare task gradients, then measure how an update from one task changes losses on other tasks. Finally, we examine the contributions of individual parameter groups to understand what a module-level summary can hide.

Setup. We use demonstrations from the 65 atomic tasks in RoboCasa365 (Nasiriany et al., 2026), with each native task ID defining one comparison group. All measurements start from a fixed RoboCasa-pretrained $\pi_{0.5}$ checkpoint and use the native action loss $\mathcal{L}$ over the full vision–language (VLM) and action parameter sets. In each sampling round, batches U_i and V_i are drawn from disjoint

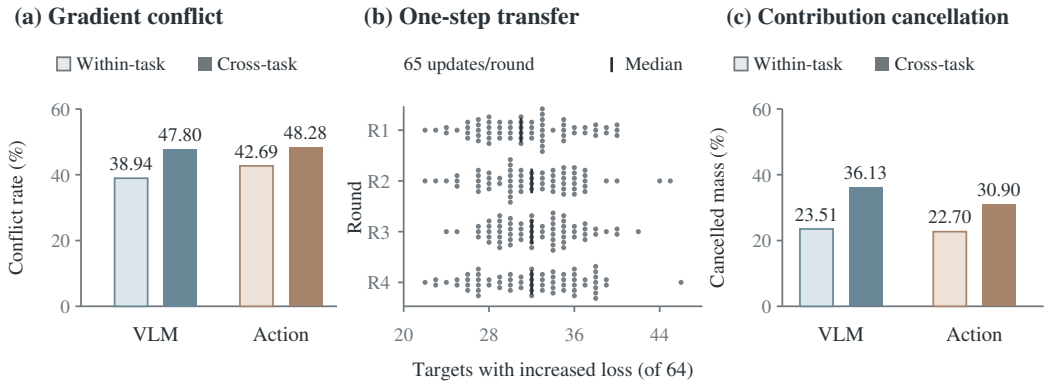


Figure 2: **Adaptation demands differ across atomic tasks and within modules.** (a) Negative-gradient-cosine frequency within and across tasks. (b) Number of other tasks whose sampled target batch has increased loss after a source update. All 260 updates reduce source-batch loss; dots represent updates and ticks mark round medians. (c) Fraction of absolute alignment contribution cancelled by opposing parameter groups. All measurements resample batches at one checkpoint: 16 rounds for (a,c) and four for (b).

episodes of task i. For gradient comparisons, (U_i, U_j) captures cross-task interactions, while (U_i, V_i) provides a reference for variation within the same task. For one-step transfer, U_i supplies the update and V_j supplies the target evaluation. Appendix A provides the sampling protocol, numerical settings, and metric definitions.

2.1 Conflicting Directions in Both Modules

We first ask whether different tasks favor compatible parameter changes. At the common checkpoint, we compute task-batch gradients separately for the VLM and action modules and compare their cosine similarity. A negative cosine indicates competing local descent directions (Yu et al., 2020a). Across 16 sampling rounds, negative alignment occurs in 47.80% of cross-task VLM comparisons and 48.28% of action comparisons, versus 38.94% and 42.69% within tasks (Figure 2(a)). The within-task reference shows that update directions also vary among episodes of the same task, while cross-task comparisons produce negative alignment more frequently. Competing adaptation demands therefore affect both the representations used for control and the action mapping.

2.2 One Update Helps Some Tasks and Hurts Others

Gradient alignment describes local update directions; we next measure the loss changes produced by an actual update. For each source batch U_i, we take one full-parameter gradient step and evaluate the target batches V_j, restarting from the common checkpoint for every source. Target observations, action targets, flow times, and noise remain fixed before and after the step. Across four rounds, all 260 updates reduce their source-batch loss, with a mean relative reduction of 2.38%. Yet each update increases loss on batches from 22–46 of the other 64 tasks, with a median of 32 (Figure 2(b)). Every source-improving update thus has both helpful and harmful effects on other tasks: improving the source does not produce uniformly beneficial transfer across manipulation contexts.

2.3 Module Averages Hide Opposing Contributions

The preceding comparisons summarize each module as a whole. To examine the interactions within these summaries, we decompose gradient alignment across 47 VLM groups and 19 action groups, including individual Transformer blocks (Shi et al., 2023). Each group's gradient inner product is normalized by the same full-module gradient norms, so the contributions sum to the module-level cosine. Positive and negative contributions can offset one another. We sum the absolute contribution mass canceled within each pair and divide by the total absolute contribution mass across comparisons. For cross-task pairs, cancellation reaches 36.13% in the VLM and 30.90% in the action module,

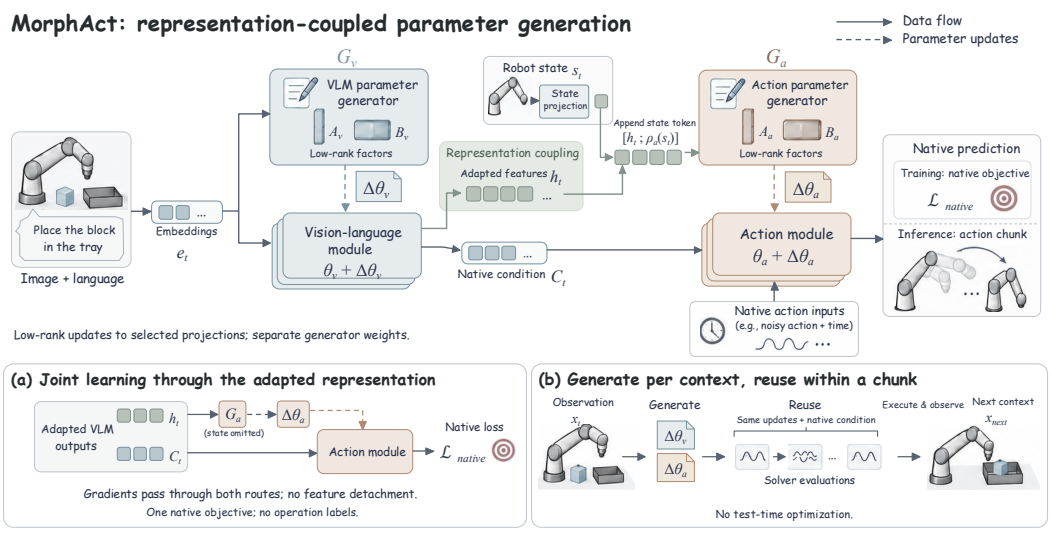


Figure 3: **Representation-coupled parameter generation.** G_v generates vision–language updates from multimodal embeddings. The adapted VLM pass produces h_t for action-side parameter generation and C_t for native action conditioning. G_a generates action updates from h_t and robot state. Solid arrows carry data; dashed arrows inject updates. Both generators learn through the action objective, and their outputs are reused within each action chunk.

compared with 23.51% and 22.70% within tasks (Figure 2(c)). A small module-level cosine can therefore reflect opposing parameter-group contributions rather than uniformly weak interactions. Appendix A.3 reports the complete group definitions and decomposition.

Implication for MorphAct. Together, these results show that adaptation demands differ across tasks and act on both representation and action parameters. An update that serves one context can work against another, with opposing contributions hidden by module-level summaries. This motivates sharing the adaptation mechanism while allowing its updates to depend on the current context. MorphAct implements this principle through two generators connected by the adapted visual–language representation.

3 MorphAct: Contextual Parameterization of VLA Models

MorphAct generates context-dependent updates for the vision–language and action modules through two generators connected by adapted representations (Figure 3). We first describe their forward computation and low-rank parameterization, then explain how the action objective trains the connected generators and how their outputs are reused during execution. The formulation preserves the base policy's conditioning interface, objective, and action-prediction procedure.

3.1 Representation-Coupled Generation

Let $x_t = (o_t, \ell, s_t)$ denote the observation, instruction, and robot state at replanning time t. We denote the base policy parameters by θ, with vision–language and action modules parameterized by θ_v and θ_a. MorphAct generates $\Delta\theta(x_t) = (\Delta\theta_{v,t}, \Delta\theta_{a,t})$ for selected projections and predicts an action chunk as $\widehat{Y}_t = \pi_{\theta+\Delta\theta(x_t)}(x_t)$, suppressing sampling randomness. The generator weights are shared across contexts, while their outputs determine the context-dependent effective parameters.

The two updates follow the policy's computational dependency: vision–language adaptation changes the representations used for control, so action-side generation conditions on the adapted features. Given multimodal embeddings $e_t = E(o_t, \ell)$, the forward computation is

$$\begin{aligned} \Delta\theta_{v,t} &= G_v(e_t; \phi_v), \qquad (h_t, C_t) = F_v(e_t; \theta_v + \Delta\theta_{v,t}), \\ u_{a,t} &= C_a(h_t, s_t; \rho_a), \qquad \Delta\theta_{a,t} = G_a(u_{a,t}; \phi_a). \end{aligned} \tag{1}$$

Here, h_t supplies the action generator's visual–language condition, whereas C_t retains the action module's native conditioning path. Both are outputs of the same adapted VLM pass. In π_0 and $\pi_{0.5}$, they correspond to final multimodal prefix tokens and layer-wise prefix key–value states, respectively. The constructor C_a appends a robot-state token projected by ρ_a to h_t. Thus, the generators have separate parameters but are connected through the representation produced by upstream adaptation.

Structured low-rank synthesis. Both generators use the structured synthesis primitive of HYWU (Wu et al., 2026), with distinct target layouts for the two modules. Queries encode layer and projection/rank identities; generator blocks combine cross-layer attention, within-layer parameter attention, and cross-attention to the condition tokens. Separate A/B readouts then produce the factors for each selected block. This preserves token-level conditioning rather than reducing the context to a single pooled vector.

For block j, the generated factors are $A_{j,t} \in \mathbb{R}^{d_{\mathrm{in},j} \times r_j}$ and $B_{j,t} \in \mathbb{R}^{r_j \times d_{\mathrm{out},j}}$. With scaling coefficient α_j, the update and its residual contribution for a row-vector input u are

$$\Delta W_{j,t} = \frac{\alpha_j}{r_j} A_{j,t} B_{j,t}, \qquad \delta y_{j,t} = \frac{\alpha_j}{r_j} (u A_{j,t}) B_{j,t}. \tag{2}$$

The factored computation avoids materializing dense updates. Q/K/V projections use output-head blocks with their actual query and key/value head counts; attention-output projections sum input-head contributions, while feed-forward projections use matrix-level factors. These layouts accommodate the modules' different widths and depths. The rank r_j is defined per generated block, not for the assembled multi-head projection. Appendix B specifies the factor geometry and output budgets.

3.2 Learning through the Adapted Representation

MorphAct trains both generators jointly through the base policy's action objective, using the same adaptation targets as the corresponding LoRA configuration. Let $\Phi = (\phi_v, \phi_a, \rho_a, \psi)$ denote the generator parameters, state projection, and projector parameters ψ retained as trainable in that configuration. The remaining VLM and action backbone parameters are frozen. Generated factors are differentiable intermediate outputs, not independent optimization variables. For demonstrations $(x_t, Y_t) \sim \mathcal{D}$, we optimize

$$\min_{\Phi} \ \mathbb{E}_{(x_t,Y_t)\sim\mathcal{D},\,\xi} \left[\mathcal{L}_{\mathrm{native}}\big(\widehat{y}_\Phi(x_t, Y_t; \xi), Y_t; \xi\big) \right], \tag{3}$$

where ξ contains the randomness required by the native training objective, and $\widehat{y}_\Phi$ denotes its training-time prediction. For π_0 and $\pi_{0.5}$, the sampled time and noise $\xi = (\tau, \epsilon)$ define $z_\tau = \tau\epsilon + (1-\tau)Y_t$, and the model predicts the velocity $\epsilon - Y_t$, rather than the clean inference chunk $\widehat{Y}_t$. This is conditional flow matching along a linear action–noise path (Lipman et al., 2023). No operation-classification, expert-weight reconstruction, or gradient-conflict loss is added.

The adapted representation participates in learning through two routes: native action conditioning and action-side parameter generation. Writing $r_t = (h_t, C_t)$ and the per-sample loss as $\ell_t(r_t, \Delta\theta_{a,t})$, with other inputs suppressed, the upstream gradient is

$$\nabla_{\phi_v} \ell_t = J_{r_t,\phi_v}^\top \Big[\frac{\partial \ell_t}{\partial r_t} + J_{\Delta\theta_{a,t},r_t}^\top \frac{\partial \ell_t}{\partial \Delta\theta_{a,t}} \Big], \qquad J_{y,z} = \frac{\partial y}{\partial z}. \tag{4}$$

The first term differentiates through action conditioning while holding the action update fixed; the second accounts for the update's dependence on adapted features. Neither route is detached. The action objective therefore trains the upstream generator through both the representations used for control and the parameters generated from those representations.

3.3 Context-Level Adaptation, Chunk-Level Reuse

Action-sequence prediction underlies both ACT's action chunking and Diffusion Policy's receding-horizon control (Zhao et al., 2023; Chi et al., 2023). Parameter generation depends on the current observation context, not on noisy actions or internal solver time. At each replanning step, MorphAct generates the VLM update, computes h_t and C_t, and generates the action update. These quantities are then reused throughout action-chunk prediction and refreshed at the next replanning observation, without test-time optimization. In π_0 and $\pi_{0.5}$, the single adapted prefix pass supplies both representation outputs, while the action expert retains its native state and time interfaces, including $\pi_{0.5}$'s time-conditioned normalization. Appendix C provides the inference procedure.

3.4 Implementation Details

Both A/B readouts use small, nonzero weight initialization and zero biases, keeping initial residuals small without blocking gradients to the generator trunk. Generator parameters are stored in FP32, while internal attention and feed-forward layers support BF16 computation. Factor readouts, low-rank products, scaling, and residual accumulation are computed in FP32 before conversion to the backbone output dtype. Training uses AdamW with learning-rate warmup, cosine decay, and global gradient clipping; updates with non-finite losses or gradient norms are skipped.

For variable-length conditions, valid tokens are compacted using the supplied masks and grouped by sequence length. The original batch order is restored after generation, preserving gradients through the adapted features. Non-reentrant activation checkpointing reduces training memory. Algorithms 1 and 2 in Appendix C provide the complete training and inference procedures, respectively.

4 Experiments

We evaluate MorphAct on LIBERO (Liu et al., 2023), Meta-World (Yu et al., 2020b), and four real-world manipulation tasks on two robot platforms. These closed-loop evaluations complement the atomic-task analysis in Section 2 by measuring complete task execution. Section 5 examines the contributions of individual design choices through ablations.

4.1 Evaluation Setting

We evaluate MorphAct in simulation and on real robots, using closed-loop task success as the primary metric. In simulation, we compare MorphAct with the corresponding baselines for X-VLA, π_0, and $\pi_{0.5}$ under the same task settings. On LIBERO, we train jointly on Spatial, Object, Goal, and Long (LIBERO-10), each containing ten tasks, and evaluate 50 rollouts per task. We report suite-level success and their arithmetic average as the Mean score. On Meta-World, we train jointly on all 50 tasks in MT50 and evaluate twenty rollouts per task. Results are reported for four difficulty groups: Easy (28 tasks), Medium (11), Hard (6), and Very Hard (5). The Avg. score is the unweighted arithmetic mean of the four difficulty-group success rates. For real-world evaluation, we compare π_0 with and without MorphAct on four manipulation tasks using the SO-101 and xArm6 platforms. Each method is trained on 100 demonstration episodes per task and evaluated over 100 physical rollouts per task. Section 4.4 describes the tasks and robot setups.

4.2 LIBERO

Table 1 shows that MorphAct improves mean success on all three backbones. On X-VLA, success rises from 93.05% to 96.60%, a gain of 3.55 percentage points. On π_0, it increases from 94.15% to 96.65%, a gain of 2.50 points. MorphAct also improves $\pi_{0.5}$ from 96.85% to 97.35%, adding 0.50 points to the highest baseline mean in this comparison.

Long shows the largest gain for each backbone: 9.00 points for X-VLA, 7.20 for π_0, and 2.20 for $\pi_{0.5}$. The other suites show backbone-dependent effects. MorphAct improves all four suites on π_0 and three on X-VLA, with X-VLA's Goal score changing from 96.00% to 95.60%. For $\pi_{0.5}$, it improves Object and Long, matches Goal, and lowers Spatial from 98.80% to 97.00%. Thus, the higher mean scores combine a consistent improvement on Long with more varied changes on the remaining suites.

4.3 Meta-World

Table 2 reports performance under joint training on MT50. MorphAct improves average success from 55.54% to 62.81% on X-VLA, from 50.75% to 64.48% on π_0, and from 48.70% to 63.83% on $\pi_{0.5}$. The corresponding gains are 7.27, 13.73, and 15.13 percentage points. All three backbones therefore benefit from MorphAct on this multi-task benchmark.

The improvements span the difficulty groups rather than appearing only in the aggregate score. X-VLA and $\pi_{0.5}$ improve in all four groups, while π_0 improves in three and matches its baseline on Hard. Some of the largest gains occur on the harder groups: π_0 rises from 20.00% to 58.00% on Very Hard, and $\pi_{0.5}$ rises from 38.30% to 65.00% on Hard. For X-VLA, the largest gain is instead on Easy,

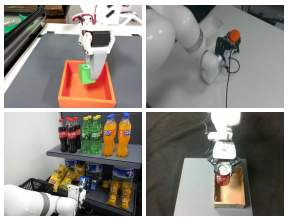
Backbone	Method	Spatial ↑	Object ↑	Goal ↑	Long ↑	Mean ↑
X-VLA	Baseline	95.40	96.60	**96.00**	84.20	93.05
	MorphAct	**99.40**	**98.20**	95.60	**93.20**	**96.60**
π_0	Baseline	96.80	98.80	95.80	85.20	94.15
	MorphAct	**98.00**	**99.60**	**96.60**	**92.40**	**96.65**
$\pi_{0.5}$	Baseline	**98.80**	98.20	**98.00**	92.40	96.85
	MorphAct	97.00	**99.80**	**98.00**	**94.60**	**97.35**

Table 1: **LIBERO closed-loop success rates (%).** Long denotes LIBERO-10; Mean averages the four suite scores. Shaded rows denote MorphAct. Bold indicates the best result within each backbone and column, including ties.

from 72.50% to 86.07%. These results show benefits across difficulty levels, with the distribution of gains depending on the backbone.

Backbone	Method	Easy ↑	Medium ↑	Hard ↑	Very Hard ↑	Avg. ↑
X-VLA	Baseline	72.50	46.36	48.30	55.00	55.54
	MorphAct	**86.07**	**55.00**	**54.17**	**56.00**	**62.81**
π_0	Baseline	77.90	51.80	**53.30**	20.00	50.75
	MorphAct	**83.90**	**62.70**	**53.30**	**58.00**	**64.48**
$\pi_{0.5}$	Baseline	70.50	45.00	38.30	41.00	48.70
	MorphAct	**85.40**	**50.90**	**65.00**	**54.00**	**63.83**

Table 2: **Meta-World closed-loop success rates (%).** Avg. is the unweighted arithmetic mean of the Easy, Medium, Hard, and Very Hard success rates. Each task is evaluated over 20 rollouts. Shaded rows denote MorphAct. Bold indicates the best result within each backbone and column, including ties.



Task	Baseline	MorphAct	Gain
Block	80.00%	**93.00%**	+13.00
Orange	65.00%	**80.00%**	+15.00
Oreo	60.00%	**77.00%**	+17.00
Can	75.00%	**88.00%**	+13.00

Figure 4: **Real-world manipulation results.** Left: task examples, read row-wise in the order listed in the table. Right: success rates of the π_0 baseline and π_0 + MorphAct. Gains are absolute differences in percentage points (pp). Each method is evaluated over 100 physical rollouts per task.

4.4 Real-World Manipulation

To evaluate MorphAct beyond simulation, we compare π_0 with and without MorphAct on four physical tasks: block placement, orange pick-and-place, Oreo retrieval, and can placement. Block placement uses an SO-101, while the other three tasks use an xArm6. For each task, both methods are trained on 100 demonstration episodes and evaluated over 100 physical rollouts per method. Success requires completion of the specified manipulation objective. Figure 4 pairs representative rollout frames with the success rates for each task. MorphAct improves success on all four tasks by 13.00–17.00 percentage points. The largest gain is on Oreo retrieval, where success increases from 60.00% to 77.00%; block placement and can placement each improve by 13.00 points, and orange pick-and-place improves by 15.00 points. These results complement the simulation comparisons with improvements during closed-loop execution on both physical robot platforms.

Variant	Spatial ↑	Object ↑	Goal ↑	Long ↑	Mean ↑
MorphAct^{A,C}	**99.40**	**98.20**	**95.60**	**93.20**	**96.60**
Constant^{A}	93.40	90.00	89.00	77.60	87.50
Task-ID^{A}	90.00	90.80	91.20	81.00	88.25
VLM-only^{B}	40.00	59.00	42.00	33.00	43.50
Action-only^{B}	88.80	92.00	73.80	69.40	81.00
Unadapted^{B,C}	91.00	97.80	83.60	71.60	86.00

Table 3: **X-VLA ablation success rates on LIBERO (%).** Superscripts identify generation conditions (A, Section 5.1), adapted modules (B, Section 5.2), and feature coupling (C, Section 5.3). Mean averages the four suite scores. The shaded row denotes MorphAct. Bold marks column-wise best scores.

4.5 Inference Cost

MorphAct generates updates once per replanning context and reuses them throughout action-chunk prediction, keeping parameter generation outside the solver loop. Under matched hardware and execution settings, end-to-end chunk latency increases by approximately 18.00–21.00% relative to shared LoRA on π_0 and $\pi_{0.5}$. This measurement covers observation preprocessing and complete action-chunk prediction, including parameter generation and low-rank residual computation. Appendix D.2 reports generator parameter counts, active-factor counts, and measured chunk latency, inference memory, and control rate.

5 Analysis

We examine three design choices in MorphAct: the condition used for parameter generation, the modules receiving updates, and the features connecting the two generators. Ablations use X-VLA on LIBERO under the task setup and evaluation protocol in Section 4.1. Table 3 reports success rates, with superscripts linking each variant to the comparisons below.

5.1 Context Beyond Task Identity

Comparison A varies the generation condition while retaining the policy's normal image, language, and robot-state inputs. Constant uses one fixed condition across tasks, whereas Task-ID uses one per task. Task-ID raises mean success from 87.50% to 88.25%, improving Object, Goal, and Long but reducing Spatial by 3.40 percentage points. MorphAct reaches 96.60%, exceeding Constant and Task-ID by 9.10 and 8.35 points on average, with improvements on every suite. The largest gains occur on Long, at 15.60 and 12.20 points, respectively. Both fixed-condition variants trail the X-VLA baseline's 93.05% mean success in Table 1, by 5.55 and 4.80 points. They retain MorphAct's generators rather than reverting to the baseline's adaptation procedure. Learning updates indirectly through generators changes optimization, which may contribute to their lower scores when conditions remain fixed. MorphAct instead lets updates respond to the current observation and robot state. The comparison supports contextual generation beyond assigning a fixed update to each task.

5.2 Adapting Both Modules

Comparison B examines the contribution of each adapted module. VLM-only and Action-only disable generated updates in the other module but retain both modules in the policy. Action-only achieves 81.00% mean success versus 43.50% for VLM-only, making action-side generation the stronger single-module choice. We then compare Action-only with Unadapted to separate upstream adaptation from feature coupling. Both condition the action generator on features computed without the generated VLM update. Unadapted additionally adapts the representation supplied to the action predictor, raising mean success from 81.00% to 86.00% with improvements on all four suites. Upstream adaptation therefore benefits action prediction even before its output is used to generate action updates.

5.3 Coupling through Adapted Features

Comparison C tests the features supplied to the action generator. Unadapted and MorphAct both generate updates for both modules and use adapted features for action prediction. Unadapted conditions action-side generation on pre-adaptation features, while MorphAct uses the adapted VLM output. This raises mean success from 86.00% to 96.60%, a gain of 10.60 percentage points. The suite-level results distinguish this effect from upstream adaptation alone. On Object, adding VLM updates raises success from Action-only's 92.00% to Unadapted's 97.80%, and coupling adds another 0.40 points. On Long, upstream adaptation adds 2.20 points, while coupling adds 21.60 points. Coupling also improves Goal by 12.00 points. Both feature sources support high success on Object, but Goal and Long benefit more from generating action updates that respond to the representation produced by upstream adaptation.

Summary. The ablations support current-context conditioning, adaptation of both modules, and coupling through adapted VLM features. Task identity alone provides a small average gain over constant conditioning, whereas current-context generation improves all four suites. The progression from Action-only to Unadapted to MorphAct separates the benefit of adapting the representation from the further benefit of using it to generate action updates. Together, these comparisons show that parameter generation depends on both the current context and the adapted features used for control.

6 Related Work

VLA models. Generalist robot policies scale learning across tasks and embodiments (Brohan et al., 2023b; Open X-Embodiment Collaboration, 2024), while Octo supports adaptation to new observation and action spaces (Ghosh et al., 2024). VIMA specifies manipulation tasks through interleaved textual and visual prompts (Jiang et al., 2023). RT-2, OpenVLA, and the π family connect pretrained vision–language representations to robot actions (Brohan et al., 2023a; Kim et al., 2025b; Black et al., 2025; Physical Intelligence et al., 2025). MorphAct studies how representation and action modules can adapt to changing manipulation contexts within a shared policy.

Parameter-efficient fine-tuning. Adapters and prefix-tuning learn small task-specific components while freezing pretrained weights (Houlsby et al., 2019; Li & Liang, 2021). LoRA learns low-rank weight updates (Hu et al., 2022), and DoRA separates weight magnitude and direction, applying low-rank adaptation to the latter (Liu et al., 2024). For VLA models, OpenVLA-OFT studies efficient decoding and fine-tuning objectives (Kim et al., 2025a), X-VLA uses embodiment-specific soft prompts (Zheng et al., 2026), and VLA-Adapter introduces Bridge Attention between vision–language features and a lightweight action policy (Wang et al., 2026). MorphAct generates context-dependent low-rank updates while retaining the host policy's action interface and objective.

Parameter sharing and generation. Multi-task learning studies structural sharing (Yang et al., 2020; Sun et al., 2022), joint optimization (Sener & Koltun, 2018; Chen et al., 2018; Yu et al., 2020a; Liu et al., 2021), and task transfer and layerwise conflicts (Fifty et al., 2021; Shi et al., 2023). Hypernetworks generate parameters (Ha et al., 2017) from task embeddings, encoder features, examples, or images (Karimi Mahabadi et al., 2021; Ivison & Peters, 2022; Phang et al., 2023; Ruiz et al., 2024). In robotics, Hyper-LoRA generates instruction-conditioned updates (Xiong et al., 2024), HyperVLA generates compact-policy parameters once per episode for efficient inference (Xiong et al., 2025), and PhaseLoRA modulates action-side LoRA per chunk (Guo et al., 2026). Using HY-WU's structured LoRA synthesis (Wu et al., 2026), MorphAct generates updates for both modules per context, conditioning action-side generation on adapted VLM features and training both generators jointly through the native action objective.

7 Conclusion

MorphAct generates context-dependent low-rank updates for vision–language and action modules through a shared adaptation mechanism. Its generators are coupled through adapted representations and trained jointly with the native action objective. Atomic-task diagnostics reveal conflicting adaptation demands in both modules, while evaluations on LIBERO, Meta-World, and four real-world tasks demonstrate improved closed-loop manipulation. Ablations further support contextual generation, adaptation of both modules, and coupling through adapted VLM features.

AI USE STATEMENT

AI tools were used to assist with writing and figure preparation.

REPRODUCIBILITY STATEMENT

Section 3 describes the model and implementation. Appendices B and C provide the parameter layouts, backbone interfaces, and training and inference algorithms. Appendix A specifies the atomic-task analysis protocol and metric definitions. Evaluation settings and ablation configurations are described in Sections 4.1, 4.4, and 5. Generator parameter counts, active-factor counts, and execution measurements are reported in Appendix D.2.

REFERENCES

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.010. URL https://www.roboticsproceedings.org/rss21/p010.html.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023a.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael S Ryoo, Grecia Salazar, Pannag R Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan H Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: Robotics Transformer for Real-World Control at Scale. In *Proceedings of Robotics: Science and Systems*, 2023b. doi: 10.15607/RSS.2023.XIX.025. URL https://www.roboticsproceedings.org/rss19/p025.html.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 794–803. PMLR, 2018. URL https://proceedings.mlr.press/v80/chen18a.html.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin CM Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.026. URL https://www.roboticsproceedings.org/rss19/p026.html.

Chris Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. Efficiently identifying task groupings for multi-task learning. *Advances in Neural Information Processing Systems*, 34: 27503–27516, 2021.

Dibya Ghosh, Homer Rich Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Quan Vuong, Ted Xiao, Pannag R Sanketi, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An Open-Source Generalist Robot Policy. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.090. URL https://www.roboticsproceedings.org/rss20/p090.html.

Yufei Guo, Yinan Wu, Haoran Duan, Guiguang Ding, and Jungong Han. PhaseLoRA: Control-Regime-Conditioned Low-Rank Adaptation for Continuous-Action Vision-Language-Action Policies. *arXiv preprint arXiv:2608.15285*, 2026. URL https://arxiv.org/abs/2608.15285.

David Ha, Andrew M. Dai, and Quoc V. Le. HyperNetworks. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1609.09106.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799. PMLR, 2019. URL https://proceedings.mlr.press/v97/houlsby19a.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.09685.

Hamish Ivison and Matthew E. Peters. Hyperdecoders: Instance-specific decoders for multi-task NLP. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 1715–1730. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.findings-emnlp.124. URL https://aclanthology.org/2022.findings-emnlp.124/.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. VIMA: Robot Manipulation with Multimodal Prompts. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 14975–15022. PMLR, 2023. URL https://proceedings.mlr.press/v202/jiang23b.html.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 565–576. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-long.47. URL https://aclanthology.org/2021.acl-long.47/.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success. In *Proceedings of Robotics: Science and Systems*, 2025a. doi: 10.15607/RSS.2025.XXI.017. URL https://www.roboticsproceedings.org/rss21/p017.html.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An Open-Source Vision-Language-Action Model. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2025b. URL https://proceedings.mlr.press/v270/kim25c.html.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353/.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-Averse Gradient Descent for Multi-task Learning. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/9d27fdf2477ffbff837d73ef7ae23db9-Abstract.html.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 36, pp. 44776–44791, 2023. doi: 10.52202/075280-1939. URL https://doi.org/10.52202/075280-1939.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32100–32121. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bn.html.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. MimicGen: A Data Generation System for Scalable Robot Learning using Human Demonstrations. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1820–1864. PMLR, 2023. URL https://proceedings.mlr.press/v229/mandlekar23a.html.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A Large-Scale Simulation Framework for Training and Benchmarking Generalist Robots. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2603.04356.

Open X-Embodiment Collaboration. Open X-Embodiment: Robotic Learning Datasets and RT-X Models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903, 2024. doi: 10.1109/ICRA57147.2024.10611477. URL https://doi.org/10.1109/ICRA57147.2024.10611477.

Jason Phang, Yi Mao, Pengcheng He, and Weizhu Chen. HyperTuning: Toward adapting large language models without back-propagation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 27854–27875. PMLR, 2023. URL https://proceedings.mlr.press/v202/phang23a.html.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization. *arXiv preprint arXiv:2504.16054*, 2025. URL https://arxiv.org/abs/2504.16054.

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Wei Wei, Tingbo Hou, Yael Pritch, Neal Wadhwa, Michael Rubinstein, and Kfir Aberman. HyperDreamBooth: HyperNetworks for fast personalization of text-to-image models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6527–6536, June 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Ruiz_HyperDreamBooth_HyperNetworks_for_Fast_Personalization_of_Text-to-Image_Models_CVPR_2024_paper.html.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems 31*, pp. 527–538, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/432aca3a1e345e339f35a30c8f65edce-Abstract.html.

Guangyuan Shi, Qimai Li, Wenlong Zhang, Jiaxin Chen, and Xiao-Ming Wu. Recon: Reducing conflicting gradients from the root for multi-task learning. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ivwZO-HnzG_.

Lingfeng Sun, Haichao Zhang, Wei Xu, and Masayoshi Tomizuka. PaCo: Parameter-compositional multi-task reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022. doi: 10.52202/068431-1562. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/86b8ad667206fb9a52ae575fbf1cd6be-Abstract-Conference.html.

Yihao Wang, Pengxiang Ding, Lingxiao Li, Can Cui, Zirui Ge, Xinyang Tong, Wenxuan Song, Han Zhao, Wei Zhao, Pengxu Hou, Siteng Huang, Yifan Tang, Wenhui Wang, Ru Zhang, Jianyi Liu, and Donglin Wang. VLA-Adapter: An Effective Paradigm for Tiny-Scale Vision-Language-Action Model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 18638–18646, 2026. doi: 10.1609/aaai.v40i22.38931. URL https://ojs.aaai.org/index.php/AAAI/article/view/38931.

Mengxuan Wu, Xuanlei Zhao, Ziqiao Wang, Ruicheng Feng, Zhangyang Wang, and Kai Wang. Hywu (part i): An extensible functional neural memory framework and an instantiation in text-guided image editing. *arXiv preprint arXiv:2603.07236*, 2026.

Zheng Xiong, Siddhant Sharma, Kang Li, Risto Vuorio, and Shimon Whiteson. Efficient domain adaptation of robotic foundation models via hypernetwork-generated lora. In *Adaptive Foundation Models: Evolving AI for Personalized and Efficient Learning*, 2024.

Zheng Xiong, Kang Li, Zilin Wang, Matthew Jackson, Jakob Foerster, and Shimon Whiteson. HyperVLA: Efficient Inference in Vision-Language-Action Models via Hypernetworks. *arXiv preprint arXiv:2510.04898*, 2025. URL https://arxiv.org/abs/2510.04898.

Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-Task Reinforcement Learning with Soft Modularization. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/32cfdce9631d8c7906e8e9d6e68b514b-Abstract.html.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient Surgery for Multi-Task Learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020a. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/3fe78a8acf5fda99de95303940a2420c-Abstract.html.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. In *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 1094–1100. PMLR, 2020b. URL https://proceedings.mlr.press/v100/yu20a.html.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.016. URL https://www.roboticsproceedings.org/rss19/p016.html.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Tai Wang, Ya-Qin Zhang, Jingjing Liu, and Xianyuan Zhan. X-VLA: Soft-Prompted Transformer as Scalable Cross-Embodiment Vision-Language-Action Model. In *International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10007740.

A Atomic-Task Diagnostic Protocol and Complete Results

The diagnostic uses the 65 native atomic tasks in RoboCasa365 (Nasiriany et al., 2026), as described in Section 2. Each task identifier defines one group; the complete identifiers appear in Table A6. Task labels organize the diagnostic comparisons and are not inputs or supervision for MorphAct.

A.1 Parameter Scope and Sampling

All three measurements start from the same RoboCasa-pretrained $\pi_{0.5}$ checkpoint and its associated normalization statistics. The VLM diagnostic set contains 2,923,335,408 scalar parameters, and the action set contains 430,098,464. These sets include vision and language backbones, token embeddings, the action expert, and their associated projections and normalization parameters. They are the full diagnostic sets, not merely the projections targeted by the generated low-rank updates. Token-embedding gradients retain their sparse representation.

For each diagnostic subset and sampling round, the sampler constructs U_i and V_i from disjoint episode groups. For alignment (A) and contribution accounting (C), each batch contains 32 samples from eight episodes, with four sampled timesteps per episode. Cross-task comparisons use U_i, U_j with $i < j$; within-task controls use U_i, V_i. The 16 rounds yield $16\binom{65}{2} = 33{,}280$ cross-task pair-rounds and $16 \times 65 = 1{,}040$ within-task observations per module. These are repeated batch samples at one checkpoint, not independent training seeds.

Setting	A: alignment	B: one-step affinity	C: contributions
Batch size	32	16	32
Compared batches	U_i, U_j; U_i, V_i	Source U_i, target V_j	U_i, U_j; U_i, V_i
Parameter precision	BF16	FP32 from BF16 checkpoint	BF16
Sampling rounds	16	4	16
Cross-task observations	33,280 unordered pair-rounds	16,640 directed pair-rounds	33,280 unordered pair-rounds
Within-task observations	1,040	260	1,040
Measurement	Module gradient cosine	Full-parameter update, target loss	66 parameter groups

Table A1: **Diagnostic settings.** A/C resample batches over 16 rounds; B uses four complete source-to-target rounds at the same checkpoint.

A uses microbatches of four, FP32 accumulation, and BF16 gradient storage. C independently computes group statistics on the same sample manifests, accumulating eight microbatch contributions per batch. Its full-module quantities are reconstructed from its own dot products and norms, so A and C have separate numerical realizations. Negative-cosine frequencies count observations within individual rounds; the sign of a pair's 16-round mean is a different statistic. Confidence intervals and zero-norm exclusion counts are not reported.

A.2 Gradient Alignment and Directional Effects

For module $m \in \{v, a\}$, we compute gradients of the native action loss on each source batch and compare their alignment:

$$g_i^{(m)} = \nabla_{\theta_m} \mathcal{L}(U_i; \theta), \qquad c_{ij}^{(m)} = \frac{\langle g_i^{(m)}, g_j^{(m)} \rangle}{\|g_i^{(m)}\|_2 \|g_j^{(m)}\|_2}. \tag{A1}$$

Negative alignment indicates competing local descent directions. The conflict frequency is $p_m^- = \operatorname{mean} \mathbf{1}[c_{ij}^{(m)} < 0]$. Cross-task comparisons use (U_i, U_j) with $i < j$; within-task comparisons use independently sampled (U_i, V_i).

Table A2 reports both conflict frequency p_m^- and angular magnitude $b_m = \operatorname{mean} \max(0, -c^{(m)})$. The last column counts pairs whose mean cosine across the 16 rounds is negative. This statistic must not be substituted for the frequency of individual negative observations.

Module	Comparison	Mean cosine (10^{-3})	p_m^- (%)	b_m (10^{-3})	Negative mean pairs
VLM	Cross-task	1.05	47.80	7.00	890/2,080
VLM	Within-task	10.24	38.94	12.01	11/65
Action	Cross-task	3.00	48.28	26.76	918/2,080
Action	Within-task	25.11	42.69	34.50	19/65

Table A2: **Gradient alignment at the fixed checkpoint.** Within-task batches come from disjoint episode groups. Mean negative part includes zeros from nonnegative observations.

For one-step transfer, each source update starts from the common checkpoint. Following the affinity convention of Fifty et al. (2021), we define

$$\theta_i^+ = \theta - \eta \nabla_\theta \mathcal{L}(U_i; \theta), \qquad Z_{i \to j} = 1 - \frac{\mathcal{L}(V_j; \theta_i^+)}{\mathcal{L}(V_j; \theta)}. \tag{A2}$$

Positive $Z_{i \to j}$ means the update reduces target loss; negative values mean it increases target loss.

For a sufficiently small unpreconditioned update, $\mathcal{L}(V_j; \theta - \eta g_i) - \mathcal{L}(V_j; \theta) = -\eta \langle \nabla_\theta \mathcal{L}(V_j; \theta), g_i \rangle + O(\eta^2)$, where $g_i = \nabla_\theta \mathcal{L}(U_i; \theta)$. This first-order relation motivates the gradient diagnostic but is not a claim about all AdamW steps. The directional experiment instead measures an actual finite step directly.

B selects 16 samples per side at positions $4k$ and $4k + 1$, $k = 0, \ldots, 7$, from each original 32-sample batch. Source gradients accumulate in FP32 with microbatches of four; target evaluation uses a microbatch of 16. Each source update starts afresh from the common checkpoint and applies Equation (A2) with $\eta \approx 0.011322$, calibrated using the first three source identifiers. Parameters are converted from the BF16 checkpoint to FP32 for this experiment. Target observations, action targets, flow times, and noise are held fixed before and after each update. Source self-improvement is measured on U_i, not on the independently sampled diagonal target V_i.

Comparison	Loss up	Loss down	Unchanged	Mean Z (10^{-3}%)	Mean increase (10^{-3}%)
Cross-task	8,310	8,326	4	-0.47	44.36
Within-task	133	127	0	$+4.89$	96.62

Table A3: **One-step target effects across four sampling rounds.** Affinity and loss changes are relative to the target's initial loss. Mean increase is conditional on a loss-increase event. A positive Z means lower target loss.

The stored relative-loss-increase matrix H is converted to affinity only by $Z = -H$ (Fifty et al., 2021). Table A3 pools all four complete 65×65 evaluations. All 260 source updates reduce source-batch loss, by 2.38% on average (range 0.37–16.61%). Each source update harms 22–46 other target batches, with a median of 32/64; conversely, each target receives antagonistic updates from at least 20 other sources within its round. The maximum cross-task relative loss increase is 0.80%. These are local loss effects, not closed-loop success rates.

A.3 Parameter Groups and Contribution Accounting

The parameter partition has 47 VLM groups: 27 vision blocks, 18 language blocks, token embeddings, and one group outside Transformer blocks. The action set has 19 groups: 18 Transformer blocks and one outside-block group. Only actual scanned Transformer axes are split by depth. The action outside-block group includes final conditional-normalization mapping, action input/output projections, and the time MLP. The VLM counterpart includes image input/output mappings, position embeddings, and final normalization parameters. These groups remain distinct from Transformer layers.

For a fixed module and sampling round, parameter group ℓ contributes to the full-module alignment as

$$q_{ij}^{(\ell)} = \frac{\langle g_i^{(\ell)}, g_j^{(\ell)} \rangle}{\|g_i\|_2 \|g_j\|_2}, \qquad c_{ij} = \sum_{\ell} q_{ij}^{(\ell)} = P_{ij} - N_{ij},$$
$$P_{ij} = \sum_{\ell} \max(q_{ij}^{(\ell)}, 0), \quad N_{ij} = \sum_{\ell} \max(-q_{ij}^{(\ell)}, 0). \tag{A3}$$

The full-module denominator makes the group contributions additive. We quantify their cancellation by pooling absolute contribution mass over pair-rounds:

$$C_m = 1 - \frac{\sum_{r,i<j} \left| P_{rij}^{(m)} - N_{rij}^{(m)} \right|}{\sum_{r,i<j} \left(P_{rij}^{(m)} + N_{rij}^{(m)} \right)}. \tag{A4}$$

For each pair and round, Equation (A3) gives $c = P - N$. The cancelled mass is $P + N - |P - N|$ and is computed *before* averaging across pairs or rounds. Equation (A4) divides its pooled sum by total absolute contribution mass. Cancellation therefore measures opposing normalized contributions, not the proportion of conflicting parameters or a percentage loss degradation.

Module	Comparison	Observations	Mixed signs (%)	Cancellation (%)
VLM	Cross-task	33,280	99.45	36.13
VLM	Within-task	1,040	94.13	23.51
Action	Cross-task	33,280	95.93	30.90
Action	Within-task	1,040	90.10	22.70

Table A4: **Opposing parameter-group contributions.** Mixed-sign observations satisfy both $P > 0$ and $N > 0$. Cancellation pools absolute contribution mass within observations.

Groups are ranked by their share of negative cross-task contribution, $\sum_{r,i<j} \max(-q_{rij}^{(\ell)}, 0)$ divided by the sum over groups in the same module. The three largest action groups—outside Transformer blocks (39.02%), block 00 (23.74%), and block 11 (16.11%)—account for 78.88% before rounding. The largest VLM groups are outside Transformer blocks (20.40%), language block 00 (17.44%), and vision block 00 (7.83%). Table A5 lists all groups. The group-local cosine used for depth profiles has its own gradient norms in the denominator; it is distinct from the additive contribution q, which uses full-module norms.

Rank	VLM parameter group	Params (M)	Conflict (%)	Negative share (%)
1	Outside Transformer blocks	3.34	48.56	20.40
2	Language block 00	110.11	48.93	17.44
3	Vision block 00	15.24	48.15	7.83
4	Vision block 01	15.24	48.23	3.91
5	Language block 14	110.11	48.86	3.48
6	Language block 07	110.11	48.80	3.14
7	Vision block 02	15.24	48.08	3.00
8	Language block 09	110.11	48.73	2.98
9	Language block 06	110.11	49.26	2.95
10	Language block 01	110.11	48.50	2.76
11	Language block 04	110.11	49.65	2.58
12	Language block 03	110.11	49.25	2.28
13	Vision block 04	15.24	48.07	2.28
14	Vision block 03	15.24	47.94	2.00
15	Language block 05	110.11	49.50	1.96
16	Language block 11	110.11	48.79	1.89
17	Language block 08	110.11	48.81	1.67
18	Language block 10	110.11	48.65	1.63
19	Language block 02	110.11	49.63	1.36
20	Language block 13	110.11	48.59	1.35
21	Language block 12	110.11	48.80	1.28
22	Vision block 06	15.24	47.67	1.07
23	Token embedding	526.65	48.97	1.00
24	Vision block 05	15.24	47.60	0.91
25	Vision block 26	15.24	48.50	0.70
26	Vision block 08	15.24	47.62	0.68
27	Vision block 24	15.24	48.75	0.64
28	Vision block 07	15.24	47.82	0.62
29	Vision block 09	15.24	48.11	0.46
30	Vision block 12	15.24	47.94	0.46
31	Vision block 13	15.24	48.30	0.45
32	Vision block 14	15.24	48.40	0.43
33	Vision block 10	15.24	48.00	0.42

Table A5: **Complete parameter-group rankings.** Groups are ranked within each module by negative cross-task contribution share. Conflict frequencies use group-local cosines. The remaining 14 VLM groups and all 19 action groups continue on the next page.

Table A5 (continued): complete parameter-group rankings.

Rank	Parameter group	Params (M)	Conflict (%)	Negative share (%)
Vision–language module (continued)				
34	Vision block 11	15.24	48.04	0.40
35	Vision block 25	15.24	48.64	0.38
36	Vision block 23	15.24	49.00	0.37
37	Language block 17	110.11	49.52	0.35
38	Vision block 15	15.24	48.47	0.31
39	Language block 16	110.11	49.03	0.27
40	Vision block 17	15.24	48.55	0.27
41	Vision block 16	15.24	48.34	0.25
42	Vision block 22	15.24	49.18	0.25
43	Vision block 18	15.24	48.56	0.25
44	Vision block 21	15.24	48.58	0.24
45	Vision block 20	15.24	49.13	0.24
46	Vision block 19	15.24	48.31	0.22
47	Language block 15	110.11	49.64	0.19
Action module				
1	Outside Transformer blocks	5.32	48.22	39.02
2	Action block 00	23.60	49.29	23.74
3	Action block 11	23.60	49.53	16.11
4	Action block 17	23.60	48.53	4.48
5	Action block 01	23.60	48.84	4.42
6	Action block 16	23.60	48.45	2.30
7	Action block 02	23.60	48.46	1.97
8	Action block 14	23.60	48.44	1.50
9	Action block 08	23.60	49.12	0.93
10	Action block 07	23.60	48.16	0.78
11	Action block 15	23.60	48.87	0.71
12	Action block 09	23.60	48.19	0.70
13	Action block 04	23.60	48.59	0.64
14	Action block 12	23.60	48.22	0.64
15	Action block 13	23.60	48.28	0.58
16	Action block 10	23.60	47.94	0.53
17	Action block 03	23.60	48.13	0.47
18	Action block 06	23.60	48.51	0.33
19	Action block 05	23.60	47.76	0.15

ID	Original RoboCasa source-task identifier	ID	Original RoboCasa source-task identifier
1	AdjustToasterOvenTemperature	34	PickPlaceCounterToCabinet
2	AdjustWaterTemperature	35	PickPlaceCounterToDrawer
3	CheesyBread	36	PickPlaceCounterToMicrowave
4	CloseBlenderLid	37	PickPlaceCounterToOven
5	CloseCabinet	38	PickPlaceCounterToSink
6	CloseDishwasher	39	PickPlaceCounterToStandMixer
7	CloseDrawer	40	PickPlaceCounterToStove
8	CloseElectricKettleLid	41	PickPlaceCounterToToasterOven
9	CloseFridge	42	PickPlaceDrawerToCounter
10	CloseFridgeDrawer	43	PickPlaceFridgeDrawerToShelf
11	CloseMicrowave	44	PickPlaceFridgeShelfToDrawer
12	CloseOven	45	PickPlaceMicrowaveToCounter
13	CloseStandMixerHead	46	PickPlaceSinkToCounter
14	CloseToasterOvenDoor	47	PickPlaceStoveToCounter
15	CoffeeServeMug	48	PickPlaceToasterOvenToCounter
16	CoffeeSetupMug	49	PickPlaceToasterToCounter
17	LowerHeat	50	PreheatOven
18	MakeIcedCoffee	51	SlideDishwasherRack
19	NavigateKitchen	52	SlideOvenRack
20	OpenBlenderLid	53	SlideToasterOvenRack
21	OpenCabinet	54	StartCoffeeMachine
22	OpenDishwasher	55	TurnOffMicrowave
23	OpenDrawer	56	TurnOffSinkFaucet
24	OpenElectricKettleLid	57	TurnOffStove
25	OpenFridge	58	TurnOnBlender
26	OpenFridgeDrawer	59	TurnOnElectricKettle
27	OpenMicrowave	60	TurnOnMicrowave
28	OpenOven	61	TurnOnSinkFaucet
29	OpenStandMixerHead	62	TurnOnStove
30	OpenToasterOvenDoor	63	TurnOnToaster
31	PackDessert	64	TurnOnToasterOven
32	PickPlaceCabinetToCounter	65	TurnSinkSpout
33	PickPlaceCounterToBlender		

Table A6: **Original identifiers retained in the diagnostic manifests.** IDs follow the original alphabetical order. Each identifier denotes one native RoboCasa365 atomic task and one diagnostic group.

A.4 PARAMETER-GROUP VISUALIZATIONS

Figure 5 retains the five largest negative-contribution groups in each module and explicitly aggregates the rest. Figure 6 reports complete depth profiles for the vision, language, and action Transformers. Frequencies and mean negative parts use group-local cosine; contribution shares use the full-module-normalized decomposition. These distinct normalizations prevent a large group-local angular conflict from being mistaken for a large contribution to the module's total interference.

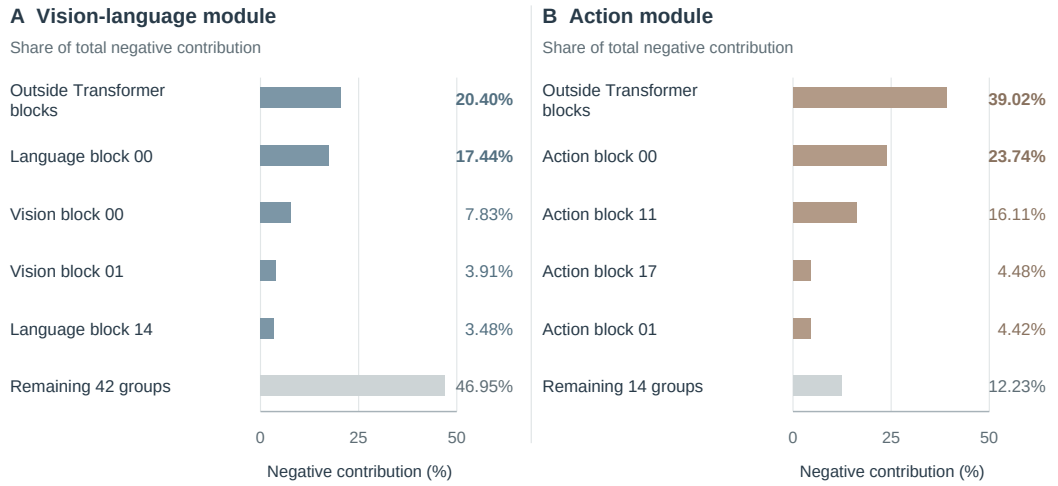


Figure 5: **Negative contributions are concentrated.** The five largest groups are shown individually for each module; all others are explicitly combined. Parameters outside Transformer blocks remain separately identified. Percentages are shares of pooled negative cross-task contribution, not fractions of model parameters.

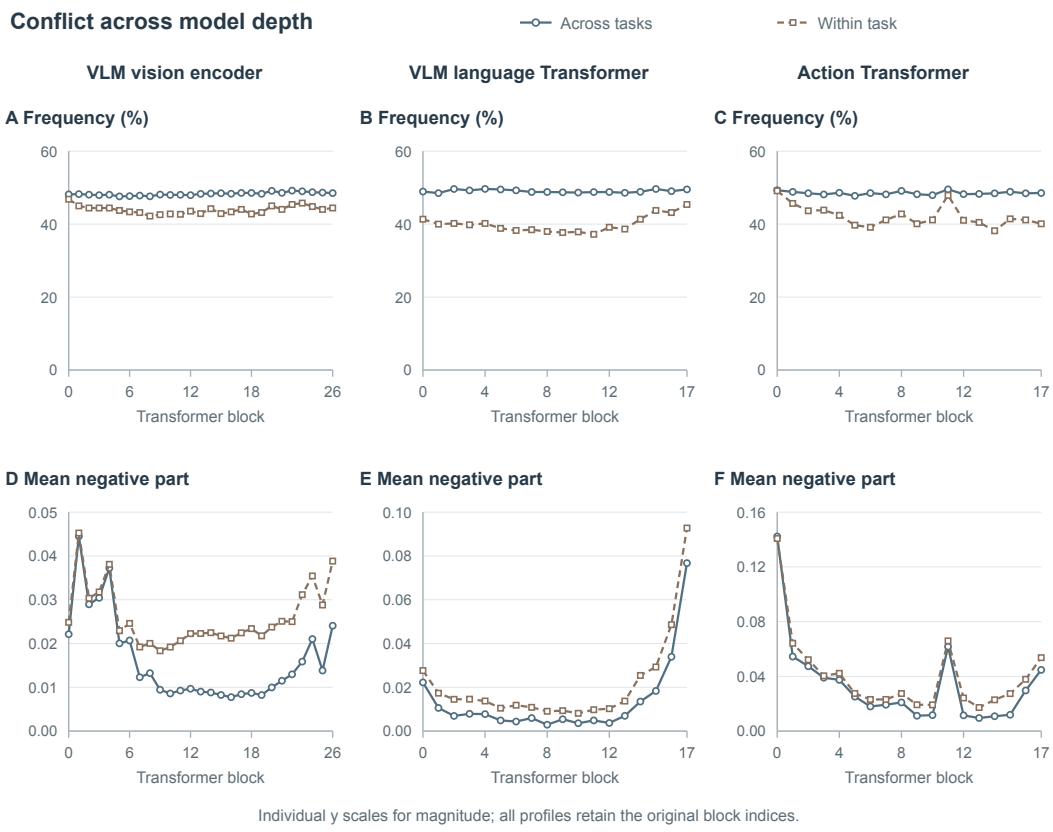


Figure 6: **Complete Transformer depth profiles.** Cross- and within-task comparisons report negative-cosine frequency and the mean negative part of group-local cosine. Block indices follow the actual 27-block vision, 18-block language, and 18-block action layouts.

B Parameter Geometry and Architecture Interfaces

For a query projection with H_q heads and head width d_h, the generated update is assembled from output-head blocks:

$$\Delta W_Q = \left[\frac{\alpha}{r}A_1B_1 \;\middle|\; \cdots \;\middle|\; \frac{\alpha}{r}A_{H_q}B_{H_q}\right], \qquad A_h \in \mathbb{R}^{d_{\text{in}}\times r}, \qquad B_h \in \mathbb{R}^{r\times d_h}. \tag{B1}$$

Key and value projections use their actual number of key–value heads. For the attention output projection, input-head contributions are summed:

$$\delta y_O = \frac{\alpha}{r}\sum_{h=1}^{H_q} u_h A_h B_h, \qquad A_h \in \mathbb{R}^{d_h\times r}, \qquad B_h \in \mathbb{R}^{r\times d_{\text{out}}}. \tag{B2}$$

Each factor product has rank at most r, whereas an assembled projection can reach the sum of block ranks, bounded by the matrix dimensions. A matrix-level and an output-head construction have scalar factor counts

$$N_{\text{matrix}} = r(d_{\text{in}} + d_{\text{out}}), \tag{B3}$$

and

$$N_{\text{heads}} = H_q r(d_{\text{in}} + d_h), \tag{B4}$$

respectively. These are per-projection, per-context output budgets, excluding generator weights. Equal block rank therefore does not imply equal generated or trainable capacity. The implementation validates layer counts, projection widths, head counts, factor shapes, and scales before executing residual injection.

The general interface in Equation (1) accommodates different base policies. Table A7 summarizes the integrations. X-VLA retains its fused QKV base projections while adding separate generated Q/K/V residuals. Its action generator additionally uses auxiliary-view visual features; this optional extension does not apply to the π instantiations.

Backbone	VLM-generator condition / targets	Action-generator condition / targets	Training prediction
π_0 / $\pi_{0.5}$	Multimodal prefix embeddings / selected VLM projections	Adapted final tokens and state / selected action-expert projections	Flow velocity
X-VLA	Visual/language embeddings / selected Florence encoder projections	Adapted features, auxiliary views, state / selected action-Transformer projections	Clean action

Table A7: **Architecture-specific interfaces.** Both integrations preserve the base policy's action-conditioning and loss definitions. Additional X-VLA view and domain interfaces are integration-specific, not prerequisites of contextual parameterization.

C JOINT TRAINING AND POLICY EXECUTION

Algorithms 1 and 2 summarize one training update and action-chunk prediction at one replanning context. Both follow the generation order in Equation (1): generate the VLM update, compute the adapted representation, and generate the action update. Generated updates are applied as low-rank residuals, without materializing dense weight updates.

Only $\Phi = (\phi_v, \phi_a, \rho_a, \psi)$ is optimized, where ψ denotes the projector parameters retained as trainable by the adaptation configuration. The remaining backbone parameters stay frozen. The condition constructor C_a uses adapted features and robot state; for X-VLA, it additionally includes native auxiliary-view features. Per-context operations in the training algorithm are evaluated over the demonstration batch.

Algorithm 1 MorphAct: One Joint Training Update

Input: Demonstration batch $\mathcal{B} = \{(x_t, Y_t)\}$, with $x_t = (o_t, \ell, s_t)$; current parameters θ, Φ; configured optimizer
Output: Updated trainable parameters Φ

Apply native preprocessing and construct embeddings $e_t = E(o_t, \ell)$ and validity masks.
$\Delta\theta_{v,t} \leftarrow G_v(e_t; \phi_v)$
$(h_t, C_t) \leftarrow F_v(e_t; \theta_v + \Delta\theta_{v,t})$
Construct $u_{a,t}$ using C_a from h_t, s_t, and any native auxiliary-view features.
$\Delta\theta_{a,t} \leftarrow G_a(u_{a,t}; \phi_a)$ ▷ Keep h_t and C_t differentiable
Sample the native objective's random variables ξ_t.
Construct the native training inputs from (Y_t, ξ_t).
Compute $\widehat{y}_t$ using the training inputs, C_t, s_t, and the action module with parameters $\theta_a + \Delta\theta_{a,t}$.
$\mathcal{L} \leftarrow \mathrm{mean}_{(x_t, Y_t) \in \mathcal{B}} \mathcal{L}_{\mathrm{native}}(\widehat{y}_t, Y_t; \xi_t)$
$g_\Phi \leftarrow \nabla_\Phi \mathcal{L}$ ▷ Through action conditioning and $h_t \to G_a$
if $\mathcal{L}$ and $\|g_\Phi\|_2$ are finite **then**
    Apply global gradient clipping to g_Φ.
    Update Φ using the configured optimizer and schedule.
else
    Skip the parameter update.
end if
return Φ

The factors produced by G_v and G_a are differentiable intermediate outputs, not separate optimization variables. The native action loss trains the VLM generator through both C_t and the dependence of $\Delta\theta_{a,t}$ on h_t. No operation labels or operation-specific adapter targets are used. The prediction target follows the backbone: flow velocity for π_0 and $\pi_{0.5}$, and clean actions for X-VLA.

Algorithm 2 MorphAct: Action-Chunk Prediction at One Context

Input: Current context $x_t = (o_t, \ell, s_t)$; learned parameters θ, Φ; native action sampler
Output: Predicted action chunk $\widehat{Y}_t$

Apply native preprocessing and construct embeddings $e_t = E(o_t, \ell)$ and validity masks.
$\Delta\theta_{v,t} \leftarrow G_v(e_t; \phi_v)$
$(h_t, C_t) \leftarrow F_v(e_t; \theta_v + \Delta\theta_{v,t})$
Construct $u_{a,t}$ using C_a from h_t, s_t, and any native auxiliary-view features.
$\Delta\theta_{a,t} \leftarrow G_a(u_{a,t}; \phi_a)$
Cache the generated updates and adapted condition C_t.
Initialize the backbone's native action sampler.
for each action-module evaluation required by the sampler **do**
    Compute the native prediction using the current sampler inputs, C_t, s_t, and $\theta_a + \Delta\theta_{a,t}$.
    Advance the sampler using its original update rule. ▷ No parameter regeneration
end for
Obtain $\widehat{Y}_t$ from the completed sampling procedure.
return $\widehat{Y}_t$

The surrounding control loop executes the first H_{exec} actions of $\widehat{Y}_t$ and acquires a new observation for replanning. Both generated updates and the adapted conditioning states are recomputed for that new

context. The generators do not receive noisy actions or internal solver time; these remain inputs to the native action-prediction procedure. The predicted horizon, executed-prefix length, and solver-step count follow the evaluation configuration.

Objective instantiations. With $z_\tau = \tau\epsilon + (1-\tau)Y_t$, π_0 and $\pi_{0.5}$ minimize velocity-prediction error,

$$\mathcal{L}_\pi = \mathbb{E}_{\tau,\epsilon}\left[\|v_\Phi(z_\tau, \tau, x_t) - (\epsilon - Y_t)\|_2^2\right], \tag{C1}$$

using the base implementation's reduction over action entries. The X-VLA integration instead predicts clean actions:

$$\mathcal{L}_{\text{X-VLA}} = \mathbb{E}_{\tau,\epsilon}\left[\ell_{\text{action-space}}\big(f_\Phi(z_\tau, \tau, x_t), Y_t\big)\right]. \tag{C2}$$

Its action-space loss retains the original treatment of position, rotation, and gripper components. These are instantiations of Equation (3), not extra objectives.

Initialization and residual scale. Both A/B readout weights are initialized from a zero-mean Gaussian with standard deviation $\sigma_0/\sqrt{d_g}$ and zero bias, where d_g is the readout input width. The generated residual $\delta y = (\alpha/r)uAB$ obeys $\|\delta y\|_2 \le (|\alpha|/r)\|u\|_2\|A\|_2\|B\|_2$, using spectral norms for the factors. Small factor scales thus favor small local residuals, without imposing an exactly zero initial policy update.

If an entire factor readout is initialized to zero, the generator trunk initially receives no gradient through that product: one factor derivative is multiplied by zero, while the zero readout has zero input Jacobian. Small, nonzero weights on both readouts avoid this structural blockage. They do not guarantee nonzero gradients for every parameter or bound accumulated policy error.

Precision and optimization. Generator parameters are stored in FP32; internal attention and feed-forward computation can use BF16. Factor readouts, low-rank multiplications, scaling, and residual accumulation are evaluated in FP32 before conversion to the base output dtype. The conversion can still quantize a sufficiently small residual. The training implementation uses one AdamW schedule with warmup and cosine decay, global gradient clipping, and checks that prevent updates on non-finite losses or gradient norms. The π implementation additionally checks trainable-gradient coverage. Non-reentrant activation checkpointing reduces activation memory.

Variable-length conditioning. Valid condition tokens are compacted using the supplied masks, grouped by sequence length, and restored to the original batch order after generation. These indexing operations retain feature gradients; detached masks are not detached representations. Grouping can require multiple generator invocations within a batch, so the reuse property is per observation and action chunk, not a fixed invocation count for every batch. Masks follow each integration's interface and do not imply that all padding is removed throughout the base VLA.

D Additional Experimental Details

D.1 Ablation Configurations

The variants in Table 3 examine conditioning, adaptation targets, and the representation passed between generators. Task-ID assigns a fixed generation condition to each task, while Constant uses conditions that remain fixed across tasks and observations. Both variants retain the policy's normal image, language, and robot-state inputs. VLM-only disables generated action-side updates; Action-only disables generated VLM updates while retaining visual inputs. Unadapted keeps both generators active but feeds the action generator the VLM representation computed before the generated VLM update. The action predictor continues to receive the adapted representation, so this variant specifically changes the condition used for action-side parameter generation.

D.2 Adaptation Budget and Runtime

For N_{solve} action-refinement steps, the approximate computation is $T_{\text{policy}} \approx T_E + T_{G_v} + T_{F_v} + T_{G_a} + N_{\text{solve}} T_{F_a}$. Generator costs include condition construction, and adapted-module costs include the generated residuals. Generation is paid per context rather than per solver step, although its end-to-end latency remains an empirical quantity.

Contextual parameterization separates learned generator weights from the updates instantiated for an observation. We report the parameter count of the two generators and, separately, the number of scalar A/B entries in an active low-rank update. Shared LoRA reuses learned factors across contexts, whereas MorphAct generates them from the current context; generated factors are outputs, not additional independently optimized parameters. Explicit factor counts also account for differences between head-aware and matrix-level layouts that a nominal rank alone does not capture.

Table A8 pairs these counts with end-to-end chunk latency, measured inference memory, and control rate. Timing covers observation preprocessing and complete action-chunk prediction, including parameter generation and residual computation where applicable. Methods are compared on matched hardware and execution settings, including solver steps, predicted action horizon, and executed-prefix length. Control rate is measured under the actual action-execution schedule, rather than inferred solely from the inverse of chunk-prediction latency.

Adaptation	Generator params (M)	Active factors (M)	Chunk latency (ms)	Memory (GiB)	Rate (Hz)
Backbone: π_0					
Shared LoRA	–	49.99	120.00	7.43	28.09
MorphAct	300.09	49.99	145.00	9.10	23.20
Backbone: $\pi_{0.5}$					
Shared LoRA	–	49.99	138.71	7.88	24.99
MorphAct	300.09	49.99	164.00	9.53	21.20

Table A8: **Adaptation capacity and execution cost.** Generator counts include the two generators only; "–" indicates the absence of a generator. Active factors count scalar A/B entries used by one context, not the number of matrices. Execution measurements use matched settings within each backbone. Control rates are measured under the action-execution schedule.

MorphAct reuses the generated updates throughout action-chunk prediction, keeping parameter generation outside the solver loop. This amortizes generation across refinement steps but retains the cost of computing and applying the updates. Relative to shared LoRA, chunk latency rises by 25.00 ms on π_0 and 25.29 ms on $\pi_{0.5}$, while inference memory rises by 1.67 and 1.65 GiB, respectively. Measured control rates are 23.20 and 21.20 Hz for MorphAct, compared with 28.09 and 24.99 Hz for shared LoRA. Read together with Table 1, these measurements characterize the manipulation-performance–execution-cost trade-off of contextual adaptation.