Never Too Late for Force: Accelerating VLA Post-Training with Reactive Force Injection

Yi Wang[12*], Wendi Chen[12*‡], Zimo Wen[1*]
Han Xue[1], Xueqi Li[23], Wenye Yu[12], Zhijie Chen[1], Hao Yang[1], Jun Lv[4]
Chuan Wen[1†], Cewu Lu[124†]
[1]Shanghai Jiao Tong University [2]Shanghai Innovation Institute
[3]Southern University of Science and Technology [4]Noematrix Ltd.
[*]Equal contribution [‡]Project lead [†]Corresponding authors
lift-policy.github.io

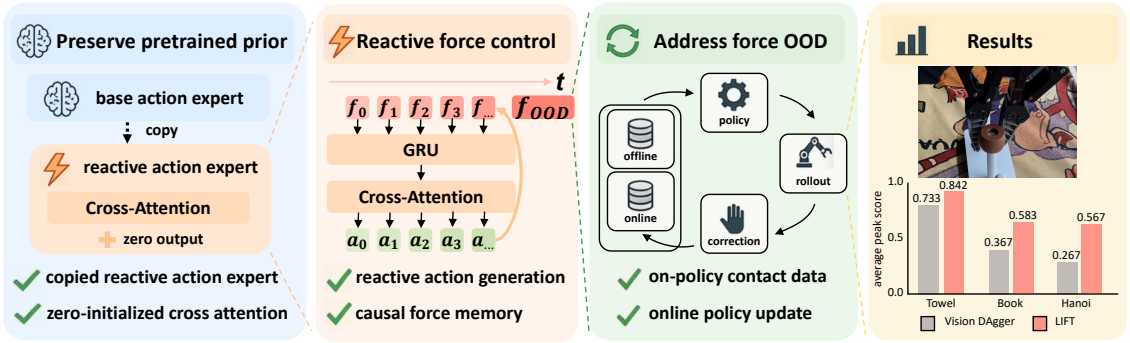

Figure 1: **LIFT overview.** (Left) LIFT copies the pretrained base action expert to initialize a reactive action expert, with zero-initialized force-injected cross attention preserving the original policy's output at initialization. (Center-left) Causal force memory encodes recent force history and injects it through cross attention for reactive within-chunk action updates. (Center-right) To address force distribution shift (force OOD), online DAgger collects on-policy contact corrections and updates the policy using mixed offline and online data. (Right) LIFT achieves higher peak and final performance than vision-only DAgger across the three evaluated manipulation tasks.

Abstract: Pretrained vision-language-action (VLA) policies provide strong language-conditioned manipulation knowledge, but they remain largely vision-driven and can struggle once manipulation enters contact states where the scene is occluded, depth is ambiguous, or small force errors push execution off the offline demonstration distribution. We present **LIFT**, ***L**ate Reactive **I**njection of **F**orce for VLA Post-**T**raining*, a force-aware post-training framework that adds contact reactivity to a pretrained VLA policy while preserving its output at initialization. LIFT grafts a reactive action expert beside the original action expert, initializes it from pretrained action weights, and injects recent 6D end-effector force through *causal force memory* and *zero-initialized cross attention*, enabling actions to be refreshed during execution. To address the policy-dependent distribution shift of contact feedback, LIFT further couples reactive force injection with an online DAgger loop that trains on a mixture of offline task-alignment data and human-corrected online rollouts. Across towel folding, book insertion, and Hanoi ring placement, LIFT learns faster and reaches higher peak and final performance than vision-only DAgger, while ablations show that reactive force memory and online corrective data are both important for robust contact-rich manipulation. Our code is publicly available at `https://github.com/y-wng/lift`.

Keywords: Force-Augmented Policies, Online Post-Training, Vision-Language-Action Models

10th Conference on Robot Learning (CoRL 2026), Austin TX, USA.

1 Introduction

Pretrained vision-language-action (VLA) policies [1, 2, 3, 4, 5, 6] have made it possible to transfer large-scale vision-language priors to robot manipulation. These models are strong at language-conditioned scene understanding and general action generation, but they still rely mainly on vision. Without force sensing, a policy can struggle when contact states are visually ambiguous due to occlusion, partial observation, or inaccurate depth perception. This limitation matters even when a task is not explicitly contact-rich, because execution still passes through contact states where physical feedback can reveal whether the robot has touched, pushed, inserted, or released an object. In those moments, force provides extra physical feedback and can also serve as a cheap memory of contact history. Recent force-aware manipulation systems [7, 8, 9, 10, 11, 12, 13] further support this view by showing that explicit contact feedback helps in both reactive control and policy learning. However, force and torque are harder to include during pretraining at scale because they are costly to collect, depend on robot platform and end-effector hardware, and vary across setups.

This makes force-aware post-training a natural next step. Instead of rebuilding the foundation model, we want to adapt an existing VLA to a specific robot platform and sensor stack. Doing so, however, raises three questions. First, because force is high-frequency, low-dimensional, and tightly coupled to contact dynamics, how should it be injected so the policy can react quickly to new contact events and retain a useful force memory? Second, adding a new force path can disturb the pretrained VLA prior, so how can we preserve it at initialization and during adaptation? Third, force distributions are highly policy-dependent and can shift sharply after small deviations from offline states, so how can post-training stay effective when offline force data do not cover most states of force?

We address these questions with **LIFT**, ***L**ate Reactive **I**njection of **F**orce for VLA Post-**T**raining*, a force-aware post-training framework for pretrained VLA models. As summarized in Figure 1, LIFT adds a reactive action expert beside the original action expert and injects *causal force memory* through a causal force encoder and cached vision-language prefix. This gives the policy a fast reactive path that can update actions in real time while treating force as a low-cost temporal context over recent contact history. To avoid damaging the pretrained model, LIFT copies the original action parameters into the reactive action expert, uses *shifted causal attention*, and adds *zero-initialized cross attention* so the network starts output-equivalent to the original VLA. To handle noisy and shifting force distributions, LIFT couples this architecture with online DAgger, so the model can keep collecting corrections from the states induced by its own policy and adapt to the resulting contact distribution.

We evaluate LIFT on three real-world manipulation tasks with different contact characteristics: towel folding, book insertion, and Hanoi ring placement. Across these tasks, LIFT improves faster than vision-only DAgger and reaches higher peak and final performance. Under the tested object, tablecloth, and lighting shifts, LIFT shows no clear degradation relative to its in-distribution behavior. Online post-training consistently outperforms offline-only adaptation on all three tasks.

Our contributions are listed below.

- We design reactive force injection with *causal force memory* and cached vision-language prefix so VLA can react to contact changes while using force as a low-cost temporal context.
- We preserve the pretrained VLA prior with initialization-equivalent parameter copying, *shifted causal attention*, and *zero-initialized cross attention*.
- We propose a force-aware online post-training framework that adapts pretrained VLA models to platform-specific force feedback and mitigates force distribution shift.

2 Problem Formulation

Starting from a vision-only VLA policy, our ultimate goal is to introduce force late in post-training without discarding its general manipulation knowledge. To achieve this, we use a two-stage training pipeline. In Stage 1, a handheld data-collection device provides an abundant vision-only task-

alignment dataset $D_v = \{(\ell, I, \mathbf{a})\}$, from which a vision-only policy $\pi^{\mathrm{V}}(\mathbf{a} \mid \ell, I)$ learns coarse manipulation knowledge. In Stage 2, we switch to a real robot equipped with a 6D end-effector force sensor and collect online corrective data $D_f^{(k)} = \{(\ell, I, \mathbf{F}, \mathbf{a}^*)\}$ (where $\mathbf{a}$ and $\mathbf{F}$ are chunks from t to $t + H - 1$). The post-training policy $\pi^{\mathrm{V+F}}(\mathbf{a} \mid \ell, I, \mathbf{F})$ is trained on the online sequence $\{D_f^{(k)}\}_k$ together with the static visual set D_v. Because the second-stage data are collected by online DAgger from the on-policy distribution $d_{\pi^{(k)}}$, the corrective set is effectively reduced to the states actually visited by the current policy, which helps mitigate covariate shift. Our hypothesis is that this force participation should speed up post-training rather than merely add another modality.

To realize this overall goal, we must solve three technical objectives:

- **Reactive force injection (O1).** Inject recent force memory into the policy so it can react to contact events instead of committing to an open-loop action chunk.
- **Preservation of general capabilities (O2).** Preserve the model's generalization ability while still improving its ability to adjust actions promptly during contact.
- **Simultaneous use of heterogeneous data (O3).** Jointly exploit abundant visual task-alignment data and scarce force-enabled online corrections so the policy benefits from both supervision sources throughout post-training.

3 Methods

Our methods are designed around the three objectives in Section 2. To satisfy O1, we add a late reactive force injection path that can causally decode actions within a chunk from recent force measurements while reusing the cached vision-language prefix. To satisfy O2, we initialize this new path so that the augmented model is output-equivalent to the original $\pi_{0.5}$ before post-training. To satisfy O3, we train the same model on both vision-only offline data and force-enabled online corrections through explicit force masking and balanced sampling.

3.1 Closing the loop with reactive force injection (O1)

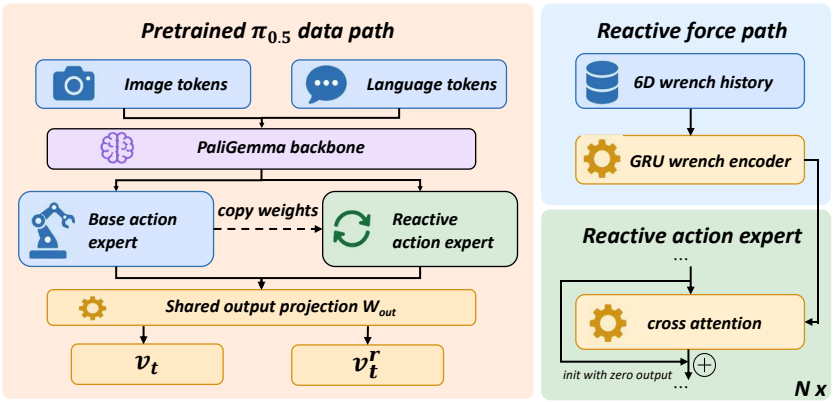

Figure 2: **LIFT architecture.** LIFT grafts a reactive action expert beside pretrained $\pi_{0.5}$ while keeping the original vision-language stack unchanged. Recent force is encoded as force memory and injected into the reactive branch through zero-initialized cross attention, enabling within-chunk action updates while preserving the pretrained prior at initialization.

Figure 2 summarizes the O1 mechanism. A pretrained $\pi_{0.5}$ action expert normally denoises a whole action chunk with full within-chunk attention, which is not suitable for updating individual actions from newly observed contact. LIFT first turns the action side into a causal reactive stream that decodes the chunk action by action, then injects latency-aligned force memory into that stream, and caches the slow vision-language context so the reactive path can be refreshed within each chunk.

O1.1 Reactive action expert. We instantiate a reactive action expert beside the original $\pi_{0.5}$ action expert and use it as the stream that will be updated during contact. For O1, the key requirement is that action decoding in this new expert is causal within each chunk. This makes the reactive stream suitable for refreshing the chunk action by action as new contact arrives. The exact shifted causal attention pattern and token-level formulation are deferred to O2.1 and highlighted in Figure 3.

O1.2 Causal force-injected cross attention. LIFT encodes recent 6D end-effector force into a causal force memory and injects it only into the reactive action expert through force-injected cross attention with a latency-aligned causal mask. This lets each reactive action use the newest force measurements available when inference finishes, so the controller can react to contact in real time without using outdated or unavailable force, with the full formulation deferred to Appendix A.

O1.3 Cached Vision-Language Slow Context. The final obstacle to closed-loop contact response is latency from the vision-language context. LIFT separates this slow context from the fast force-injected action update. At deployment, the runtime first computes the vision-language prefix once and stores its KV cache. Within the action chunk, it then repeatedly encodes the latest latency-aligned force history and reevaluates the action experts against the cached prefix. Because the expensive vision-language prefix is reused, each refresh can update the reactive action chunk with newly observed contact transients without waiting for a full vision-language forward pass.

3.2 Preserving the pretrained VLA prior at initialization (O2)

Reactive force injection gives LIFT the closed-loop path required by O1, but it also alters the pretrained VLA's action-side structure. Such structural change can make the model forget the original manipulation capabilities that make the pretrained prior useful. We therefore preserve the prior at initialization through an output-equivalent reactive action expert and zero-initialized cross attention.

O2.1 Make reactive action expert output-equivalent. Starting from pretrained $\pi_{0.5}$, we instantiate a second action-denoising expert with the same action-token interface as the original expert and copy the original action-expert weights into it. The original action expert is fully attentive within a chunk: each base action token a_i can attend to all action tokens $a_{0:H-1}$. LIFT keeps this base action stream unchanged, as shown by ① in Figure 3, then appends reactive tokens $r_{0:H-1}$ with the shifted causal attention introduced in O1.1 and illustrated in ③ of Figure 3. For each position i, the reactive token r_i sees the vision-language prefix, later base-action tokens $a_{i+1:H-1}$, and the causal reactive prefix $r_{0:i}$, while blocking base-action tokens $a_{0:i}$ and subsequent reactive tokens $r_{i+1:H-1}$. At initialization, copied weights and aligned positions make $r_{0:i}$ equivalent to base-action representations $a_{0:i}$, while the shifted context supplies $a_{i+1:H-1}$ from the base stream. Thus r_i receives the same context as a_i in the original fully-attentive action expert, making ① and ③ in Figure 3 initialization-equivalent and the reactive action expert output-equivalent to the original before force training.

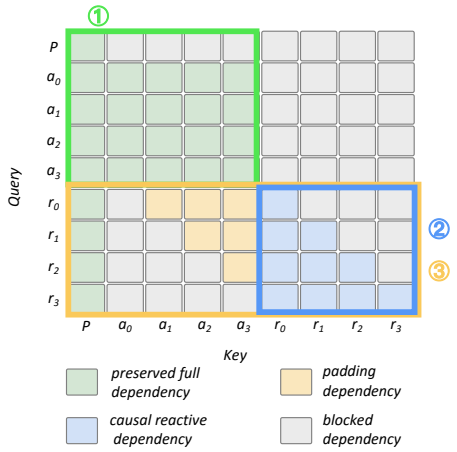

Figure 3: **Shifted causal attention with initialization equivalence.** Each reactive token attends to the vision-language prefix, later base-action tokens, and the causal reactive prefix. Combined with the copied pretrained action expert, LIFT preserves initialization equivalence while enabling causal reactive updates.

O2.2 Zero-initialize cross attention output. The zero-initialized cross attention block is added as a residual update to the reactive action representation. LIFT zero-initializes the output projection of this residual path, so the force-injected update is exactly zero at step 0 even though the force encoder and attention logits are present. With no effective force residual, the copied reactive action expert receives the same action interface and output projection as the original action expert. Thus force cannot change the pretrained action output until post-training learns a nonzero residual.

Together, O2.1 and O2.2 let LIFT add the causal action branch and force path needed for O1 while starting post-training from the same action output as the base model.

3.3 Training with heterogeneous visual and force data (O3)

O3 turns the architecture above into a post-training recipe that uses both available supervision sources. The offline set D_v provides broad vision-only task alignment from the handheld data-collection device, while the online set D_f provides force-enabled corrections on states visited by the deployed policy. LIFT trains both streams with one shared objective, but uses explicit force masking so samples without force data do not update the force encoder.

O3.1 Additive flow-matching objective. For both data sources, LIFT jointly trains the original and reactive action streams. We sample independent noises $\varepsilon, \varepsilon^{\mathrm{r}} \sim \mathcal{N}(0, I)$ and same-time interpolants $x_\tau, x_\tau^{\mathrm{r}}$, then minimize

$$\mathcal{L}(\theta) \;=\; \left\| v_\theta(x_\tau, \tau, o) - u \right\|_2^2 \;+\; \left\| v_\theta^{\mathrm{r}}(x_\tau^{\mathrm{r}}, \tau, o, m_{t:t+H}) - u^{\mathrm{r}} \right\|_2^2. \tag{1}$$

The two losses are computed in the same forward pass, their gradients are accumulated together, and the shared parameters are updated jointly. This keeps the base stream anchored to the pretrained action prior while the reactive stream learns force-injected corrections. At inference time, LIFT still computes both action streams because the reactive stream uses the base stream through shifted causal attention, but only the reactive action is sent to the robot controller.

O3.2 Selective force masking. Each sample enables the force path only when real force data exists. Vision-only batches use zero force placeholders for shape, then mask encoded force memory before cross attention, blocking gradients to the force encoder and force-injected attention. Online correction batches keep measured force memory active, updating the force pathway and reactive branch. Thus one model trains on both data sources without synthetic force labels.

O3.3 Equal sampling of heterogeneous data. We follow the RLPD symmetric sampling strategy [14] and train on a 1:1 mixture of offline task-alignment batches and online corrective batches. Each update therefore sees equal amounts of prior-preserving vision-only data and force-enabled on-policy corrections, allowing the model to benefit from the complementary strengths of both sources.

3.4 Pipeline overview and system implementation

As shown in Figure 4, the pipeline instantiates the three objectives in two stages. Stage 1 builds LIFT on pretrained $\pi_{0.5}$ and trains on the vision-only dataset D_v with force masked out, so the model aligns to the target tasks while preserving the pretrained prior. Stage 2 moves to the real robot, collects force-enabled corrective episodes into D_f, and continues post-training on a fixed 1:1 mixture of D_v and the asynchronously ingested online buffer D_f. Similar to recent work [15, 16] on online post-training infrastructure, the latest checkpoint is periodically synced back to the rollout client for redeployment, closing the loop between correction collection, post-training, and redeployment.

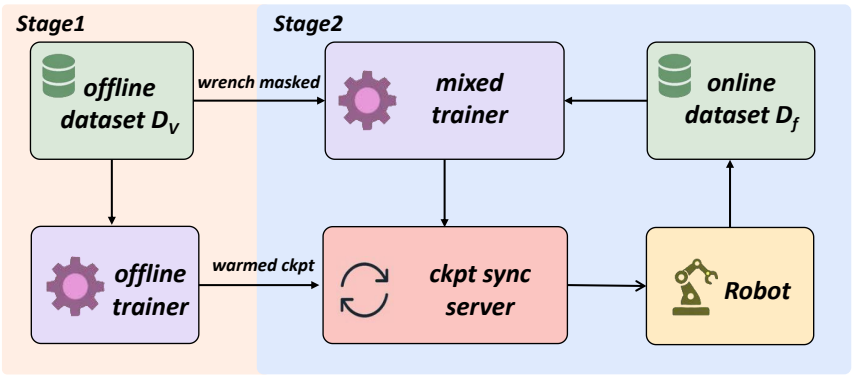

Figure 4: **LIFT training pipeline.** The two-stage system first aligns LIFT on vision-only demonstrations and then closes an online post-training loop between the robot and cloud trainer.

4 Experimental Results

4.1 Experimental setup

We evaluate towel folding, book insertion, and Hanoi ring placement on a Flexiv Rizon 4S with a 6D end-effector force sensor. Stage-one vision-only demonstrations use a handheld device; stage-two force-enabled corrections use Flexiv TDK [17]. Each checkpoint is evaluated in three groups of ten autonomous rollouts ($n = 30$) to compute 95% confidence intervals. Towel and book use graded task scores, whereas Hanoi uses binary success. Task protocols and failure case analysis are provided in Figure 9.

We compare six policies:

- $\pi_{0.5}$ **w/ Online DAgger**: the same online DAgger loop without force input.
- **LIFT w/o Reactive Force Injection**: stage-two force post-training with a single-frame force input rather than the reactive force memory.
- **LIFT w/ Offline DAgger**: the reactive model trained with a fixed offline force-correction buffer, without repeated on-policy aggregation or online updates.
- $\pi_{0.5}$ **w/ Offline Handheld Data**: a policy trained only on the larger offline handheld set.
- **Residual Policy**: a lightweight force-reactive Transformer that adds action corrections to a frozen, offline-trained $\pi_{0.5}$ policy.
- **LIFT**: the full reactive force-injected VLA trained with online DAgger.

4.2 Does force accelerate or improve post-training? (Q1)

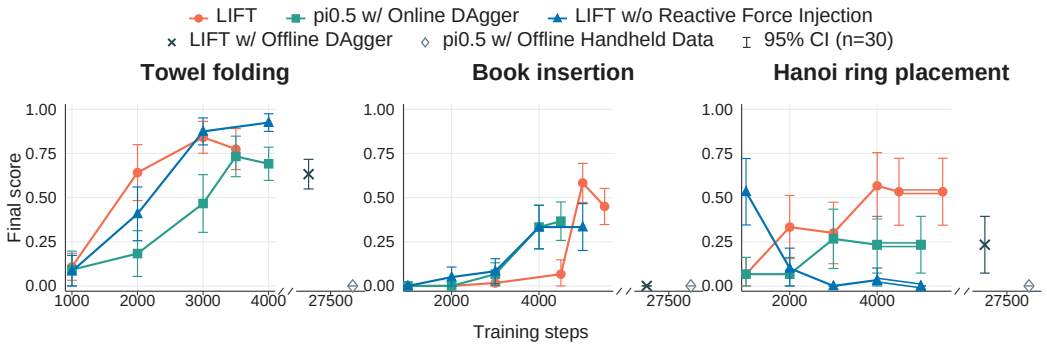

Figure 5: **Learning curves on three tasks as training steps increase.** Force-enabled post-training improves adaptation over vision-only online DAgger.

Figure 5 shows that force accelerates post-training and raises peak and final performance. For fair benchmarking, we compare policies at similar training steps (e.g., 1,000 and 2,000) within a comparable time budget. Each online experiment lasts approximately 2–3 hours and collects roughly 20–30 episodes.

On towel folding, both force-aware variants improve faster and reach higher scores than vision-only online DAgger. Because the towel is thin, a monocular wrist camera can misestimate depth and fail to tell whether the gripper has actually grasped the cloth. Force feedback exposes this state and allows recovery from an empty grasp during the first fold.

Book insertion highlights force feedback for constrained insertion. The book is bounded by neighboring books and the shelf, so side contact and insertion depth determine when to push, stop, or release. LIFT reaches higher scores because it learns these transitions, whereas vision-only control may continue pushing after the book has bottomed out or release before it is seated, causing the book to flip or fail to settle.

Hanoi ring placement shows the same advantage in precision placement. The small ring–pole clearance makes slight misalignment likely to cause a jam, while vision-only control can continue pushing downward and tilt the ring. LIFT uses contact-force direction and magnitude to identify offset or stuck states and adjust its motion, supporting faster learning and better performance.

4.3 Does reactivity matter? (Q2)

We compare LIFT with LIFT w/o Reactive Force Injection in Figure 5. The comparison asks whether force should enter as a reactive force memory rather than as a single-frame cue. Reactivity yields a clear benefit on book insertion and Hanoi ring placement, where the contact state evolves throughout execution.

On book insertion, LIFT achieves higher scores than the single-frame variant. A typical single-frame failure occurs after the book has already bottomed out in the slot: the policy keeps pushing downward against the shelf, even under large contact force, instead of releasing the gripper or backing out. This is a non-Markovian decision because the same current load can correspond either to ongoing insertion or to a completed insertion that should trigger release. A short force-memory window lets LIFT infer the contact phase from recent force evolution, stabilizes the force input, and enables timely release or withdrawal rather than continued pushing after the book has bottomed out.

On Hanoi ring placement, the advantage is more pronounced. A transient impact can trigger oscillation or drive the single-frame policy away from the placement region, producing unstable forward and backward motion or loss of localization. Because contact can occur around the whole ring, fast and precise correction is also required. Single-frame force is easily misled by instantaneous noise or out-of-distribution impact forces. LIFT averages this evidence over a short force-memory window and updates the action chunk quickly after the contact state changes, distinguishing transient impacts from persistent misalignment and enabling correction before placement fails.

Towel folding involves simpler contact transitions: a single force frame can already indicate whether the cloth is grasped. The additional benefit of reactivity is therefore smaller, and the single-frame variant can perform as well as or better than LIFT. Nevertheless, reactive force injection maintains better performance than the vision-only baseline, showing that LIFT is also useful when contact is relatively simple rather than highly contact-rich.

4.4 Does LIFT preserve original VLA generalization? (Q3)

Under the tested object, tablecloth, and lighting changes, Figure 6 shows no clear performance degradation relative to the in-distribution setting. The exact shifts, including the alternate desktop, blue towel, and purple lighting, are detailed in Appendix I; this preservation is consistent with the copied action expert, shifted causal attention, and zero-initialized cross attention.

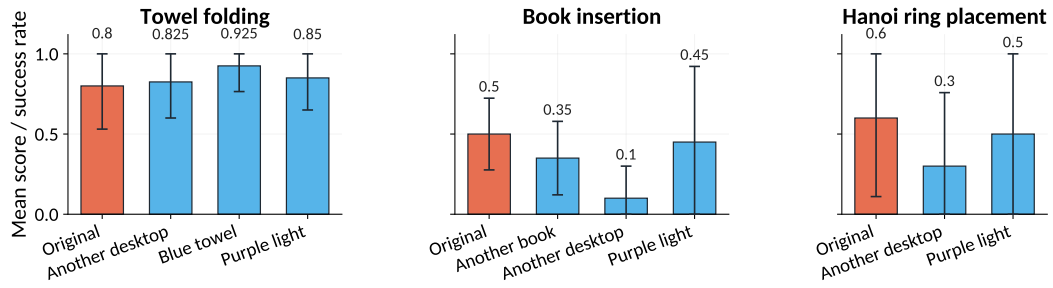

Figure 6: **Generalization of the final LIFT checkpoint.** LIFT shows no clear degradation under the tested object, tablecloth, and lighting changes. These tests retain ten rollouts per condition.

4.5 Does online data matter for reactive force injection? (Q4)

Offline-only comparisons are plotted as separate markers in Figure 5. LIFT w/ Offline DAgger, which uses a fixed force-correction buffer, underperforms on all three tasks and scores zero on book insertion. Offline data miss abrupt force-pattern changes during execution, while online DAgger exposes the policy to failure states induced by its own behavior and closes this distribution gap.

4.6 Why use LIFT rather than a lightweight residual policy? (Q5)

The lightweight residual policy achieves substantially lower scores than LIFT (Figure 7). The weak base requires large corrections, while sparse residual targets make intervention timing difficult to learn. Details and prior-method comparisons are in Appendix F.

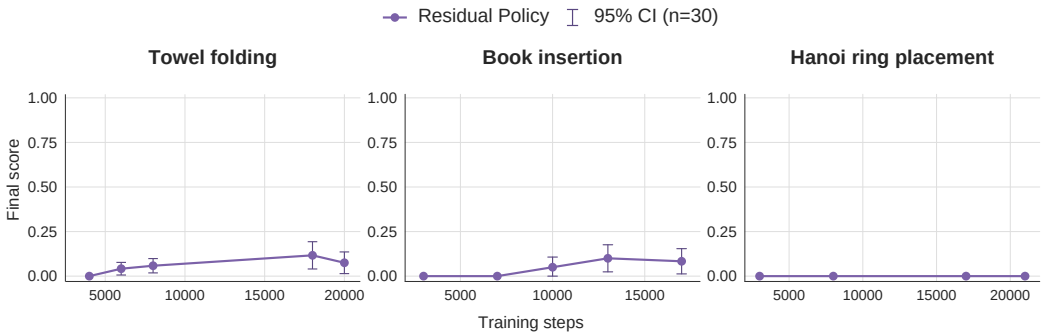

Figure 7: **Lightweight residual baseline on three tasks.** The lightweight residual module underperforms LIFT in Figure 5 under the tested base-policy and intervention protocol.

4.7 Why retain offline data during online post-training? (Q6)

Offline data broaden behavioral coverage and help preserve the pretrained prior. On towel folding, 0:1 performs substantially worse than 1:1 and 1:2, with 1:1 improving faster (Figure 8). All main experiments use fixed 1:1 offline:online sampling.

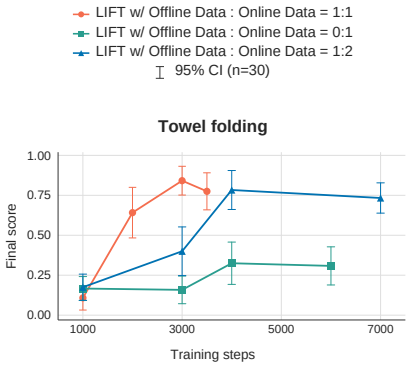

Figure 8: **Offline data ratio ablation.** Mixed training outperforms online-only training; the main protocol uses fixed 1:1 sampling.

5 Related Work

Force- and torque-aware VLAs. ForceVLA [18] fuses 6-axis force and vision-language features through modality- and phase-aware expert routing during action decoding. TA-VLA [19] compares torque integration strategies, favoring decoder-side adapters, a history-summary token, and auxiliary torque prediction. ForceVLA2 [20] combines force prompts and a cross-scale MoE for hybrid force–position control. FAVLA [21] uses predicted force variation to schedule a fast expert, injecting recent force sequences into multiple expert layers. LIFT instead emphasizes late VLA post-training: copied weights and zero-initialized force cross attention preserve initial outputs, while latency-aligned masking governs force access under inference delay. Our reactive expert refreshes full position-based actions under cached visual context, without desired-force targets.

Reactive visual-force policies. RDP [10] predicts latent actions with slow visual diffusion and decodes them with fast tactile feedback. ImplicitRDP [11] replaces this hierarchy with one visual-force diffusion network, using GRU force memory, causal attention, and consistent inference for within-chunk feedback. These precedents are neither simply open-loop nor additive-residual policies. LIFT's shifted causal attention additionally retains the pretrained action context at initialization while making the reactive stream causal, enabling feedback under a pretrained prior rather than proposing fast feedback alone.

Online corrections and residual learning. DAgger [22] addresses learner-induced distribution shift. CR-DAgger [23] collects compliant delta corrections without interrupting the base policy; its lightweight residual adjusts position and predicts force commands. It reuses a frozen image encoder and a temporal convolutional force encoder, and includes zero-residual supervision on non-correction states. Residual capacity is therefore only one part of its design. Our takeover interface and position commands differ, so our residual baseline is not a CR-DAgger reproduction. LIFT instead learns full actions from offline demonstrations and online corrections, complementing online training infrastructure [15, 16]. Further comparisons are provided in Appendix.

6 Conclusion and Limitations

LIFT adds force-aware post-training to pretrained VLA policies through late reactive force injection, prior-preserving initialization, and online corrective training. Across three tasks, it learns faster and reaches higher peak and final performance than vision-only DAgger while preserving the pretrained prior. This suggests that post-training can add a physically grounded modality to a pretrained VLA without disturbing its existing capabilities.

Limitations and future directions. Human corrections limit data throughput, and our evaluation covers one arm; future work should reduce correction load and test diverse robot arms, force sensors, and end-effectors.

Acknowledgments

We thank the reviewers for their constructive comments. We also thank Yuan Fang, Fangyuan Zhou, and Yanwen Zou for helpful discussions. We are especially grateful to Shirun Tang for invaluable help and support.

References

[1] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

[2] O. M. Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[3] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[4] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[5] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

[6] G. A. Team. Gen-1: Scaling embodied foundation models to mastery. *Generalist AI Blog*, 2026. https://generalistai.com/blog/apr-02-2026-GEN-1.

[7] Z. He, H. Fang, J. Chen, H.-S. Fang, and C. Lu. Foar: Force-aware reactive policy for contact-rich robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.

[8] W. Liu, J. Wang, Y. Wang, W. Wang, and C. Lu. Forcemimic: Force-centric imitation learning with force-motion capture system for contact-rich manipulation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1105–1112. IEEE, 2025.

[9] Y. Hou, Z. Liu, C. Chi, E. Cousineau, N. Kuppuswamy, S. Feng, B. Burchfiel, and S. Song. Adaptive compliance policy: Learning approximate compliance for diffusion guided control. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4829–4836. IEEE, 2025.

[10] H. Xue, J. Ren, W. Chen, G. Zhang, Y. Fang, G. Gu, H. Xu, and C. Lu. Reactive diffusion policy: Slow-fast visual-tactile policy learning for contact-rich manipulation. *arXiv preprint arXiv:2503.02881*, 2025.

[11] W. Chen, H. Xue, Y. Wang, F. Zhou, J. Lv, Y. Jin, S. Tang, C. Wen, and C. Lu. Implicitrdp: An end-to-end visual-force diffusion policy with structural slow-fast learning. *arXiv preprint arXiv:2512.10946*, 2025.

[12] X. Li, Y. Wang, W. Chen, et al. Vtam: Visual-tactile action model for contact-rich manipulation. arXiv preprint, 2025.

[13] Y. Wang, W. Chen, H. Xue, et al. Ftp-1: Foundation tactile policy for contact-rich manipulation. arXiv preprint, 2025.

[14] P. J. Ball, L. Smith, I. Kostrikov, and S. Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pages 1577–1594. PMLR, 2023.

[15] M. Pan, S. Feng, Q. Zhang, X. Li, J. Song, C. Qu, Y. Wang, C. Li, Z. Xiong, Z. Chen, et al. Sop: A scalable online post-training system for vision-language-action models. *arXiv preprint arXiv:2601.03044*, 2026.

[16] J. Fang, W. Chen, H. Xue, F. Zhou, T. Le, Y. Wang, Y. Zhang, J. Lv, C. Wen, and C. Lu. Robopocket: Improve robot policies instantly with your phone. *arXiv preprint arXiv:2603.05504*, 2026.

[17] Flexiv Robotics. Flexiv TDK: Teleoperation development kit for flexiv robots. https://github.com/flexivrobotics/flexiv_tdk, 2026. GitHub repository.

[18] J. Yu, H. Liu, Q. Yu, J. Ren, C. Hao, H. Ding, G. Huang, G. Huang, Y. Song, P. Cai, W. Zhang, and C. Lu. ForceVLA: Enhancing VLA models with a force-aware MoE for contact-rich manipulation. In *Advances in Neural Information Processing Systems*, volume 38, pages 93409–93439, 2025.

[19] Z. Zhang, H. Xu, Z. Yang, C. Yue, Z. Lin, H.-a. Gao, Z. Wang, and H. Zhao. Ta-vla: Elucidating the design space of torque-aware vision-language-action models. *arXiv preprint arXiv:2509.07962*, 2025.

[20] Y. Li, H. Jiang, J. Xia, H. Zhang, J. Du, Y. Zhou, J. Zeng, C. Hao, J. Ren, Q. Yu, et al. Forcevla2: Unleashing hybrid force-position control with force awareness for contact-rich manipulation. *arXiv preprint arXiv:2603.15169*, 2026.

[21] Y. Li, P. Tang, W. Zhang, C. Zhu, Y. Duan, W. Shi, X. Zhang, Z. Yang, J. Ji, and Y. Zhang. Favla: A force-adaptive fast–slow vla model for contact-rich robotic manipulation. arXiv preprint arXiv:2602.23648, 2026.

[22] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[23] X. Xu, Y. Hou, Z. Liu, and S. Song. Compliant residual DAgger: Improving real-world contact-rich manipulation with human corrections. In *Advances in Neural Information Processing Systems*, volume 38, pages 139559–139581, 2025.

A Latency-aligned causal mask details

After the reactive action stream is made causal, LIFT injects force so each action can condition on the newest admissible force observations. The recent 6D end-effector force chunk $F_{t:t+H} \in \mathbb{R}^{H\times 6}$ is first encoded by a single-layer **causal** GRU followed by a linear projection,

$$m_{t:t+H} \;=\; \mathrm{Linear}_{d_m}\big(\mathrm{GRU}_{d_h}(F_{t:t+H})\big) \;\in\; \mathbb{R}^{H\times d_m}, \tag{2}$$

where $d_h = d_m/2$ and d_m matches the action-expert width, and we denote the resulting sequence as the *force memory* $m_{t:t+H}$.

Only the reactive action tokens attend to this force memory. Concretely, each reactive query q_i attends over the full force-memory sequence through force-injected cross attention, while a latency-aligned causal mask removes force tokens that should not yet be available:

$$\begin{aligned}
\mathrm{Cross}(q_i, m_{0:H-1}) &= \sum_{j=0}^{H-1} \alpha_{ij}\, m_j W_V, \\
\alpha_{ij} &= \frac{\exp(s_{ij} + b_{ij}^{(L)})}{\sum_{k=0}^{H-1} \exp(s_{ik} + b_{ik}^{(L)})}, \qquad s_{ij} = \frac{(q_i W_Q)(m_j W_K)^\top}{\sqrt{d_m}}, \\
b_{ij}^{(L)} &= \begin{cases} 0, & j \le i - L, \\ -\infty, & j > i - L. \end{cases}
\end{aligned} \tag{3}$$

Here L is a latency-alignment offset chosen to match the network inference delay. The mask is causal in the force stream: it prevents the reactive action expert from using force measurements that should not yet be available and shifts the available force window to the action time after inference completes. The shift ensures that after a nonzero delay, the model still outputs the action to execute at completion time rather than an outdated action.

B Pseudocode: cached reactive inference

For completeness, Algorithm 1 restates the cached reactive rollout loop referenced in Section 3.

Algorithm 1 Cached reactive inference with latency-aligned force injection.

Require: policy π_θ, language instruction ℓ, image stream, force stream, execution horizon H, diffusion noise level N, latency offset L

$t \leftarrow 0$
while task not done **do**
  $I_t \leftarrow \mathrm{GetImage}()$
  $K_{\mathrm{VL}}, V_{\mathrm{VL}} \leftarrow \mathrm{VLA}_{\mathrm{prefix}}(\ell, I_t)$ ▷ cache slow vision-language context
  $x_N, x_N^{\mathrm{r}} \sim \mathcal{N}(0, \mathbf{I})$ ▷ initialize base and reactive action noise
  cache the latency-aligned causal mask $b^{(L)}$ with $b_{ij}^{(L)} = -\infty$ for $j > i - L$
  $\mathcal{F} \leftarrow [\,]$ ▷ clear force memory history for this chunk
  for $i \leftarrow 0$ **to** $H - 1$ **do**
    $f_{t+i} \leftarrow \mathrm{GetForceFrame}()$
    append f_{t+i} to $\mathcal{F}$
    $m_{t:t+i} \leftarrow \mathrm{ForceEncoder}(\mathcal{F})$ using (2)
    $\hat{a}_{t+i} \leftarrow \mathrm{DenoiseAction}(\pi_\theta, K_{\mathrm{VL}}, V_{\mathrm{VL}}, x_N, x_N^{\mathrm{r}}, m_{t:t+i}, b^{(L)})$
    $\mathrm{Execute}(\hat{a}_{t+i})$
    if task done **then**
      break
    end if
  end for
  $t \leftarrow t + H$ ▷ refresh slow context, noise, and force history
end while

C Hardware and sensor setup

Robot. A single Flexiv Rizon 4S 7-DoF cobot equipped with a Robotiq 2F-85 parallel-jaw gripper executes all rollouts. The arm operates in position-control mode at 10 Hz, and trajectory targets from the policy are tracked by the robot's internal impedance controller.

Sensing. A built-in 6D end-effector force/torque (F/T) sensor provides raw wrench measurements $f_x, f_y, f_z, \tau_x, \tau_y, \tau_z$ at more than 1000 Hz. We downsample this stream to 10 Hz and time-synchronize it with the action stream before using it as policy input. A wrist-mounted iPhone provides RGB images at 10 Hz. We use a single-camera setup for all tasks reported here.

Teleoperation. Stage-two force-feedback demonstrations are collected through Flexiv TDK [17]. TDK uses a bilateral master-follower setup with two Flexiv Rizon 4S arms: the operator moves the master arm, while the follower arm executes the task. Contact forces measured on the follower arm are fed back to the master arm, allowing the operator to feel force signals and adjust the motion.

D Dataset details

Stage-one offline data (stage-one handheld). The stage-one offline dataset D_v is collected with a handheld gripper with iPhone (no force sensor) at 10 Hz. We use the iPhone main camera for RGB recording and call the ARKit SLAM interface to estimate the camera pose. Each demonstration is recorded and represented in the iPhone camera frame. We remove short static segments from the beginning and end of each episode to avoid stuck frames.

Stage-two online data (stage-two online correction data). Collected with the Flexiv TDK setup. During collection, we set the robot TCP to the iPhone camera frame. Each timestep stores the synchronized 6D wrench alongside RGB observations. We record robot commands rather than robot states to provide more accurate control supervision. Only human-intervention correction data are added to the stage-two dataset D_f.

Action representation. All actions are relative actions. The action vector is padded to the shared output dimension required by the policy head.

E Training hyperparameters

Table 1: **Architecture hyperparameters.**

Parameter	Value
Backbone	$\pi_{0.5}$ with PaliGemma vision-language tower and action expert
Action chunk horizon	10
Padded action dimension	32
Force input dimension	6D wrench
Force encoder	Single-layer causal GRU followed by a linear projection
Force encoder hidden width	512
Force encoder output width	$d_m = 1024$
Latency-alignment offset	3
Reactive expert initialization	Copied from the base action expert
Cross-attention initialization	Zero-initialized output projection

F Residual baseline implementation

Our custom residual baseline is inspired by ImplicitRDP [11] and CR-DAgger [23], rather than a reproduction of either system. It learns force-conditioned corrections to a frozen policy, whereas LIFT adapts a full action expert. The comparison changes model capacity, initialization, and supervision together.

Table 2: **Optimization hyperparameters.**

Parameter	Value
Batch size	32
Total training steps	30,000
Learning-rate schedule	10,000-step warmup, cosine schedule
Peak and final learning rate	5×10^{-5}; decay horizon 1,000,000
Optimizer	AdamW
AdamW β_1	0.9
AdamW β_2	0.95
AdamW ε	10^{-8}
Weight decay	10^{-10}
Gradient clipping norm	1.0

Policy and feedback. The residual baseline freezes the offline-trained $\pi_{0.5}$ base policy and its image encoder, and trains only the residual module. The approximately 28M-parameter module uses a six-layer Transformer with width 512, four attention heads, MLP width 2048, and training dropout 0.1, together with a force GRU. Visual features are cached at 1 Hz; force-conditioned position-action corrections are generated at 10 Hz. Its action queries use causal self-attention and attend to valid visual tokens and causally available force tokens. Predictions are deterministic at inference, without diffusion sampling in the residual decoder. The output is added to the base action after reversing the configured residual scaling.

Targets and optimization. Training uses the fixed 1:1 offline–online mixture. Offline samples receive zero residual targets. For online samples, the target is the expert action minus the base-policy action at marked intervention timesteps, divided by the per-dimension residual scale; non-intervention targets are zero. The loss is mean squared error on the residual action dimensions. This differs from LIFT's full-action flow-matching supervision on both demonstrations and corrections. The comparison consequently changes model capacity, initialization, and target representation together; it is not a parameter-matched ablation. The weak base policy and sparse correction targets motivate the interpretation in Section 4.6, but their individual contributions are not isolated experimentally.

G Task protocol and failure case analysis

Each rollout lasts at most 2 min. For towel folding, the robot starts 5 cm directly above the lowest towel corner, with the towel reset before each rollout. For Hanoi, the robot starts holding the ring above the base of a 10 mm-diameter pole. The qualitative examples are shown in Figure 9.

Towel folding. The towel score is graded by task stage: grasping the first corner gives 0.25, completing the first fold gives 0.5, grasping the second corner gives 0.75, and completing the second fold gives 1.0. The score is the highest completed stage rather than an additive sum. Common failures include an empty grasp, grasping the wrong second corner, and an incorrect folding motion.

Book insertion. The robot receives 0.5 for grasping and lifting the book and inserting it without collision. It receives 1.0 after closing the gripper, pushing the spine until the book is fully inserted, and disengaging from contact. Common failures include side collision or early release, over-pushing or missing the spine, incomplete insertion, and failure to disengage.

Hanoi ring placement. The Hanoi task is binary: a ring fully seated on the pole base receives 1, and an unsuccessful placement receives 0. Common failures include angled insertion and jamming, as well as impact-induced oscillation or drift.

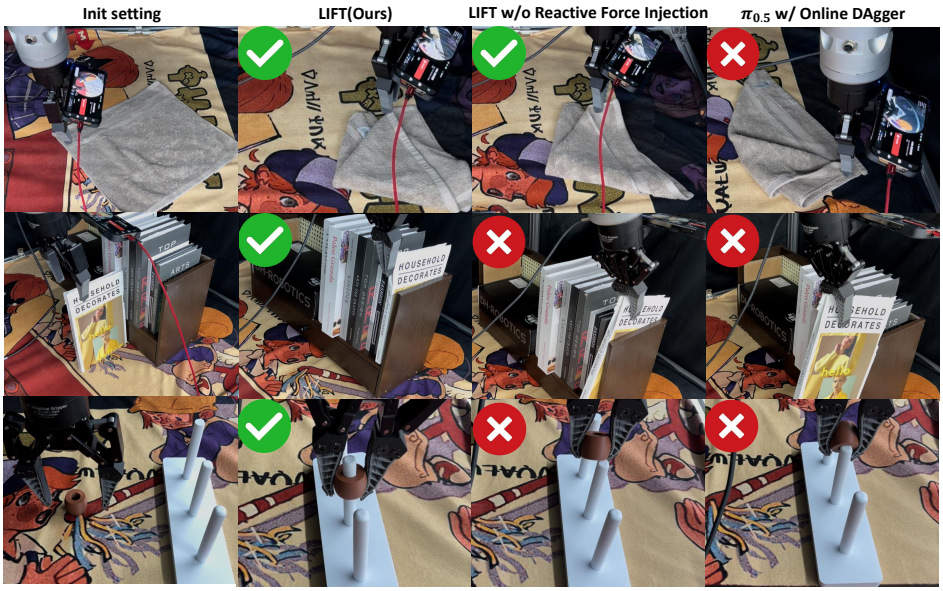

Figure 9: **Task protocol and failure case analysis.** Qualitative examples of towel folding, book insertion, and Hanoi ring placement, with success and failure marked by checks and crosses.

H Online DAgger protocol

Data sources. Two datasets are mixed at training time: an offline visual dataset $\mathcal{D}_v$ (stage-one handheld), and a growing online dataset $\mathcal{D}_f$ (stage-two online correction data) that the collection pipeline appends as each new rollout terminates. The online data are force-enabled.

Fixed data mixing. The main experiments use a fixed 1:1 offline:online sampling ratio. Each training batch therefore contains equal amounts of offline task-alignment data and online force-enabled correction data. The ratio ablation is reported separately on towel folding; its results do not change the main protocol.

Synchronization. The trainer checks the data-cloud endpoint for newly completed episodes at every training step. Updated checkpoints are pushed to the inference server every 100 training steps, which refreshes the policy deployed on the robot.

Human intervention. During each rollout, an operator monitors the robot and triggers a reset whenever (i) the policy stalls in a non-progressing configuration, (ii) the force trace exceeds a hardware safety threshold, or (iii) the episode exceeds the task's max length.

I Generalization condition details

Figure 10: **Generalization task settings.** The three panels show the towel folding, book insertion, and Hanoi ring placement setups used for shifted-condition evaluation.

Towel folding.

- *Blue towel:* replace the original towel with a blue towel of the same style.
- *Another tablecloth:* replace the rough cartoon-style brown tablecloth with a slightly smoother yellow tablecloth with an oil-painting texture.
- *Purple light:* add a purple side light while keeping the task geometry unchanged.

Book insertion.

- *Gray book:* replace the original book with a new gray book.
- *Another tablecloth:* replace the rough cartoon-style brown tablecloth with the same smoother yellow oil-painting-texture tablecloth used in the towel-folding shift.
- *Purple light:* add the same purple side light.

Hanoi ring placement.

- *Another tablecloth:* replace the rough cartoon-style brown tablecloth with the smoother yellow oil-painting-texture tablecloth.
- *Purple light:* add the same purple side light.

For every shift the robot start pose and target object location are kept identical to the in-distribution evaluation; only the listed factor changes.

J Compute and reproducibility

Compute. Each training run uses a single 8-GPU node equipped with NVIDIA A800 GPUs.

Code release. Our codebase is built on `openpi` and is publicly available at `https://github.com/y-wng/lift`.