# VLA-ULAP: INTERLEAVING CLOUD VLA CALLS WITH ULTRA-LIGHTWEIGHT LOCAL ACTION PREDICTION AT THE EDGE

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Billion-parameter vision–language–action (VLA) policies run either onboard, consuming substantial power, or on remote servers, adding communication latency. To address these drawbacks and better balance latency and onboard energy consumption, we propose VLA-ULAP. It partitions inference across decision times, interleaving remote VLA calls with predictions from an Ultra-Lightweight Local Action Predictor (ULAP). A single ULAP has $\sim$7.4M parameters including the frozen vision encoder. It combines current views, proprioception, and executed action history to predict chunks in one pass. Trained independently, it requires no VLA hidden states, online verification, or server round trips. On Jetson Orin Nano, ULAP takes 20.7 ms and 0.122 J of idle-subtracted energy per inference, versus 289.3 ms and 40.46 J for GR00T on RTX A6000. Across four simulated base-policy/benchmark pairs, VLA-ULAP removes 45.3–77.3% of VLA calls while retaining 95.0–98.5% of baseline success rates at selected operating points. On VLA-JEPA, VLA-ULAP also surpasses local acceleration alternatives, using an estimated 49.4% less inference time and 51.5% less GPU energy per successful episode than ACT, and 77.5% less time and 80.7% less energy than SP-VLA at higher success rates. In physical SO-101 trials, it similarly removes 70.3–71.9% of VLA calls without observed success-rate loss at seen or held-out placements. Measured device costs imply 64.6–66.4% less inference time and 70.1–71.7% less idle-subtracted energy per successful episode at these call counts. Beyond these savings, faster responses help VLA-ULAP exceed $\pi_{0.5}$'s success rate by 11.0 and 15.5 percentage points (pp) on two tasks in latency-aware LIBERO-Safety simulation while approximately halving VLA calls.

## 1 INTRODUCTION

Vision–language–action (VLA) policies leverage visual–language pretraining and robot experience to follow instructions and generalize across tasks and environments (Kim et al., 2025; Bjorck et al., 2025; Black et al., 2025). Deployment commonly uses onboard or remote GPUs. Onboard inference avoids network delay, but trades power for responsiveness: Yang et al. (2026) report naive $\pi_{0.5}$ latencies of 76.4 ms on RTX 4090 and 457.9 ms on Jetson Thor, whose listed maximum power ratings are 450 W and 130 W. The former's power budget challenges battery-powered operation; the latter's latency spans about 14 control steps at 30 Hz, delaying responses to scene changes. Remote inference offers powerful GPUs without their onboard battery burden, but adds communication delay and jitter (Peng et al., 2026). For example, Huang et al. (2025) measure Wi-Fi image-transfer costs of tens of ms and hundreds of mJ per frame. Neither placement alone resolves this latency–onboard-energy trade-off, raising the question: **Can hybrid inference achieve a better balance between latency and onboard energy consumption without sacrificing task success?**

Existing methods accelerate individual policies but leave hybrid requirements unmet. Quantization, token pruning, and adaptive depth reduce work within a VLA call (Zhang et al., 2026; Liu et al., 2026b; Yue et al., 2024), yet lower average cost does not ensure that peak compute and memory demands fit a low-power edge device. Adaptive execution horizons use more of an existing action chunk (Feng et al., 2026; Liang et al., 2026), but leave the VLA's per-call computation unchanged and cannot extend execution beyond its generated horizon. They therefore neither make the VLA



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.

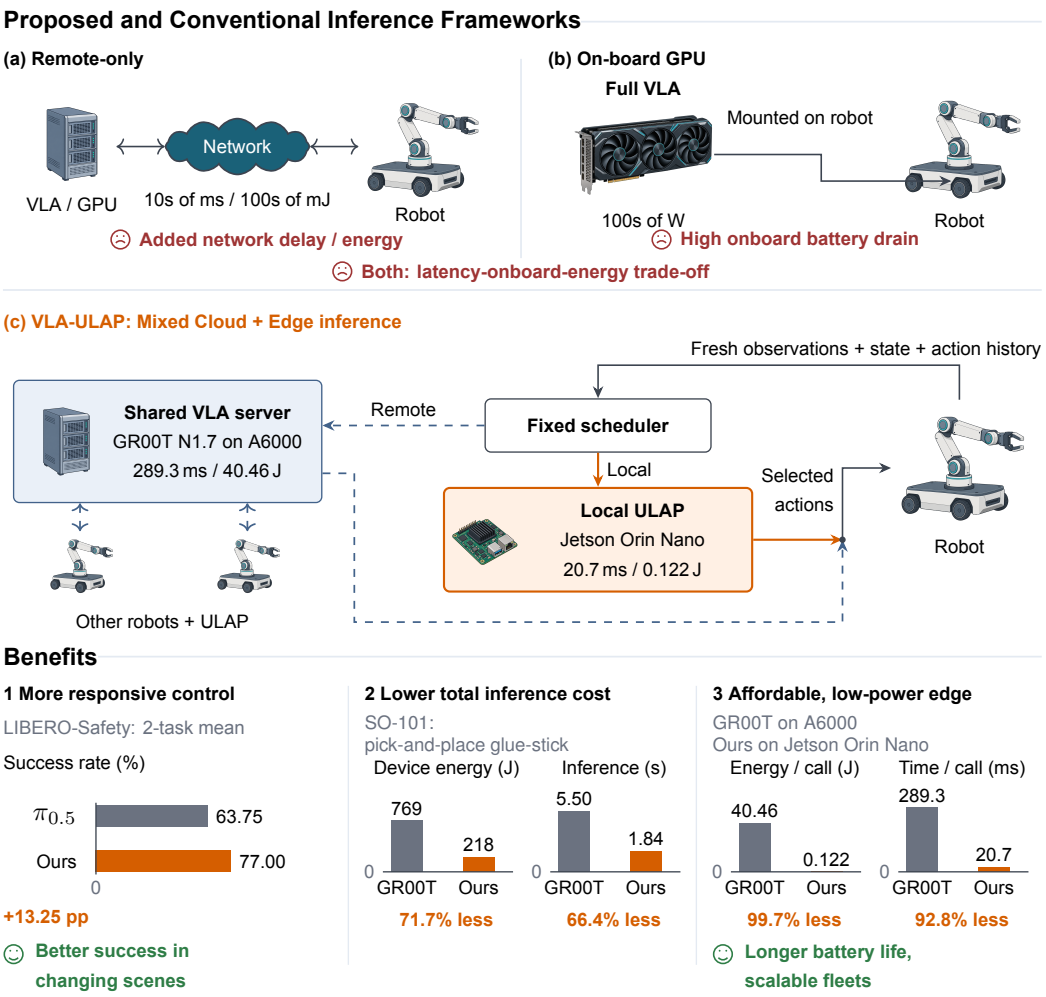



Figure 1: **Overview and benefits of the proposed VLA-ULAP framework.** (a) Remote-only inference adds communication delay and energy cost, while (b) full onboard VLA inference burdens the battery. (c) VLA-ULAP schedules remote calls alongside local predictions. Bottom panels show three benefits: (1) more responsive control with higher dynamic-task success rates; (2) lower estimated GPU-plus-edge energy and inference time per successful episode; and (3) fast, low-energy inference on inexpensive edge devices for longer battery life and scalable fleets.

fit a low-power edge device nor remove the horizon-imposed limit on call reduction. Lightweight policies such as ACT (Zhao et al., 2023) generate actions locally, but without large-scale pretraining, standalone deployment can fall short in task success and generalization. Model-internal partitioning instead couples a VLM with a fast action policy, as in HiRT (Zhang et al., 2025). The fast policy uses VLM features, so a remote VLM still requires communication to refresh them.

To achieve a better latency–onboard-energy trade-off through hybrid inference, we propose **VLA-ULAP**, a hybrid framework that interleaves occasional remote VLA calls with inexpensive action predictions from an **Ultra-Lightweight Local Action Predictor (ULAP)** on the robot (Figure 1). Local decisions respond to fresh observations without a server round trip, while periodic VLA calls help preserve task success by retaining access to the base policy. We partition inference across decision times rather than within the model (e.g., a remote VLM and a local action head), avoiding transfers of large intermediate features. This framework offers three benefits: (1) faster responses with fewer round trips, (2) lower total inference time and energy, and (3) affordable, low-power edge inference. ULAP avoids billion-parameter VLA memory demands on edge devices, allowing inexpensive, low-power hardware such as Jetson Orin Nano to replace consumer-grade GPUs. This supports longer battery life and, together with fewer remote calls, larger robot fleets per server.

To fully realize this framework's benefits, the local path must use fresh observations, be lightweight, and remain independent of remote features or verification. ULAP meets these requirements through the following design choices. It combines current camera views and proprioception with the preceding executed action sequence, which supplies motion context that appearance alone may not resolve. A frozen Theia-Tiny encoder (Shang et al., 2025), shallow multimodal fusion, and an action head that generates the entire chunk in one pass without iterative denoising keep prediction inexpensive and low-latency; the model has 1.86–2.12M trainable parameters depending on the benchmark's action chunk length, action dimensions, and task conditioning, plus 5.52M frozen visual parameters (Figure 2). Unlike HiRT (Zhang et al., 2025), ULAP requires no intermediate features from the remote model. The base VLA stays frozen. Only ULAP is trained on demonstrations or successful base-policy trajectories; for a few hundred episodes, this takes a few minutes on one A6000 GPU. Because ULAP uses no VLA internal states, it requires no joint training and does not depend on the base VLA's internal architecture. At inference time, we interleave VLA calls and fresh local predictions at chunk boundaries using fixed scheduling.

Our contributions are:

1. A split inference framework that deploys local action prediction on a low-power edge device and reduces expensive VLA calls on remote servers. This improves the latency–onboard-energy trade-off while preserving task success.
2. An ultra-lightweight local predictor that uses fresh vision, state, and action history to generate action chunks in one pass on a low-power edge device, without changing the base VLA or requiring its intermediate representations.
3. Strong success retention across diverse robot-policy architectures, including VLA and world-model-based policies. Our evaluation covers manipulation on LIBERO (Liu et al., 2023), generalization to held-out objects in RoboCasa's varied kitchen environments (Nasiriany et al., 2024), and long-horizon, multi-task execution on CALVIN (Mees et al., 2022). SO-101 trials (Knight et al., 2024) and Jetson measurements additionally test real-robot performance and edge inference. Latency-aware LIBERO-Safety evaluation (Cui et al., 2026) further tests whether fast local updates can improve success, not merely reduce cost.

At selected simulation settings, 45.3–77.3% fewer VLA calls retain 95.0–98.5% of baseline success rate. In physical trials, mean VLA calls per successful episode fall by 70.3–71.9% without observed success-rate loss at ID or OOD placements. On two dynamic tasks, our proposed VLA-ULAP improves success over $\pi_{0.5}$ (Black et al., 2025) by 11.0 and 15.5 percentage points (pp). Section 2 reviews related work, Section 3 describes our method, and Section 4 presents the evaluation.

## 2 Related Work

VLA inference acceleration methods can be broadly grouped into five families.

**Reduce computation within a VLA call.** Quantization (Zhang et al., 2026), token pruning or reuse (Liu et al., 2026b; Xu et al., 2025), and adaptive depth or intermediate-state reuse (Yue et al., 2024; Liu et al., 2026a; Oi et al., 2026) lower bit widths or skip internal computation, yet low-latency inference on Jetson Orin Nano-class devices can remain challenging. They can complement VLA-ULAP's call reduction by reducing the computational burden of the remaining VLA calls.

**Adaptive execution chunk length.** Action chunking amortizes one prediction over several control steps (Zhao et al., 2023). AutoHorizon, AAC, and DVAC adapt the executed prefix using attention, entropy, or denoising variance (Wang et al., 2026; Liang et al., 2026; Feng et al., 2026). Their selected actions remain within a generated chunk, so call reduction is bounded by the base policy's action chunk length.

**Action reuse and extrapolation.** FlashVLA repeats a recent action (Tan et al., 2025), whereas SP-VLA extrapolates an action-history buffer (Li et al., 2026). They can act beyond the base policy's action chunk length, but do not predict from current observations during bypass, so aggressive reuse or extrapolation can miss scene changes and sharply degrade task success. ULAP can also act

beyond the VLA's action chunk length, but predicts a new chunk conditioned on current observations and robot state.

**Retrieve executable actions.** RT-Cache (Kwon et al., 2025) uses frozen DINOv2 (Oquab et al., 2024) and SigLIP (Zhai et al., 2023) features to retrieve trajectory snippets. ALT (He et al., 2026) learns a query from multiple views and end-effector pose. Retrieving actions can avoid expensive VLA inference, but their reported zero-shot and OOD evaluations indicate limited generalization beyond memory coverage, including changes in object placement.

**Learn a cheaper action policy.** HiRT conditions a fast policy on VLM-generated latents (Zhang et al., 2025). Realtime-VLA FLASH couples an approximately 110M-parameter draft model to the main image encoder and Action Expert verifier (Niu et al., 2026). ULAP requires neither transferred large-model features nor online verification. We also compare an adapted ACT baseline (Zhao et al., 2023) to test a standard lightweight action-chunk predictor in the same hybrid framework. Appendix A expands the taxonomy.

## 3 METHOD: VLA-ULAP

Building on the discussions in Section 1, ULAP must use current observations, remain lightweight, and operate without VLA hidden states or online verification. The architecture in Figure 2 implements these three requirements as follows.

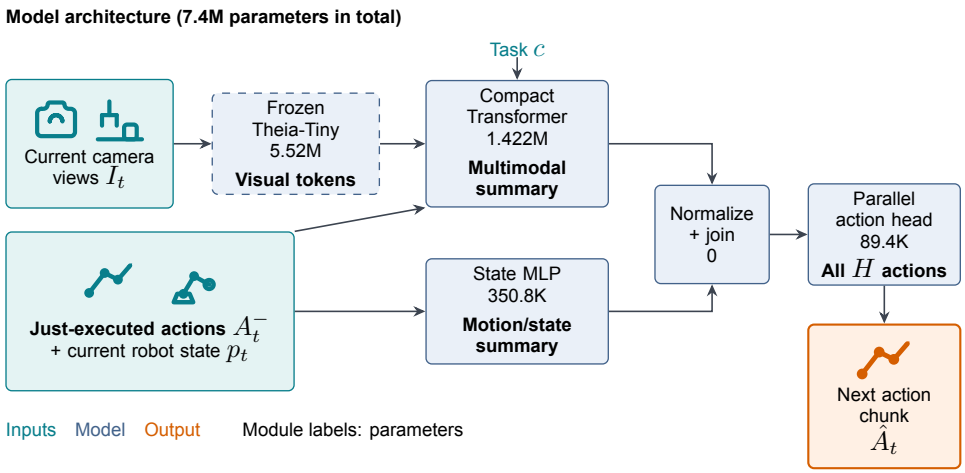



Figure 2: **ULAP model architecture.** The Transformer summarizes vision, task, and motion context; the state branch preserves nonvisual information. The action head uses their concatenated outputs to predict a chunk in one pass. Counts are for five-step LIBERO-Safety.

**Predict chunks from local vision, state, and action history.** At a decision boundary $t$, let $I_t$ denote the current camera views, $p_t$ the robot state, and $A_t^- = (a_{t-H}, \ldots, a_{t-1})$ the preceding $H$ executed actions. ULAP predicts

$$\widehat{A}_t = g_\theta(I_t, p_t, A_t^-, c),$$

where $c$ denotes task conditioning and $\widehat{A}_t$ contains the next $H$ actions. Similar views can occur while approaching, grasping, or withdrawing from an object. To help distinguish these phases, ULAP combines current vision and proprioception with the preceding actions actually sent to the robot, regardless of whether the VLA or ULAP generated them. The Transformer fuses $7 \times 7$-pooled Theia tokens (Appendix C.3), projected history/proprioception tokens, and an aggregation token augmented with a learned task embedding. A parallel state MLP maps proprioception and flattened action history directly to a nonvisual summary. Both summaries are independently normalized and concatenated with equal scaling. The action head (Figure 2) applies a shared two-layer MLP to fused features concatenated with learned temporal-position tokens, predicting all $H$ actions in parallel.

GR00T input ablations support retaining both the state MLP and action history for a better success–call-reduction trade-off at most tested $\geq$90%-retention settings (Appendix C.1).

**Lightweight: 1.86–2.12M trainable parameters.** ULAP combines a **5.52M-parameter frozen Theia-Tiny** encoder (Shang et al., 2025) with a **1.86–2.12M-parameter trainable predictor**, depending on action chunk length, action dimensions, and task conditioning. Including frozen vision, it is roughly 1/400 of the 3B-parameter GR00T N1.7 (NVIDIA, 2026). Rather than running a large vision–language backbone locally, ULAP uses compact visual features, a shallow Transformer, and a small state MLP. Its MLP action head generates the entire action chunk in a single forward pass without iterative denoising, keeping prediction latency low. Task embeddings or precomputed instruction features avoid an online language-model pass. GR00T ablations find no significant success gain from Theia-Small, supporting Tiny as the compact default (Appendix C.2).

**Independent: direct prediction without VLA verification.** All inputs to ULAP are available on the robot. Without VLA hidden states or online verification, ULAP runs locally without feature-transfer or verification round trips. The unchanged VLA is invoked only at scheduled decisions.

**Training.** Only the predictor is trained, while the VLA and Theia remain frozen. We supervise the next action chunk with demonstrations or successful base-policy rollouts using Smooth-L1 loss on training-set-standardized actions.

**Scheduling and closed-loop execution.** At deployment, the first decision invokes the VLA to initialize executed history. We then use a fixed accumulator-based scheduler to spread local decisions evenly among VLA calls. The requested local fraction $\rho$ is the target proportion of chunk decisions assigned to ULAP rather than the VLA. With equal execution horizons, the accumulator starts at $\rho$ to account for the initial VLA chunk. At each subsequent boundary, it adds $\rho$; if it reaches one, ULAP is selected and one is subtracted, otherwise the VLA is called. This distributes VLA refreshes as regularly as the requested fraction permits, avoiding unnecessary clusters of consecutive local predictions. We also tested several adaptive scheduling signals, but found no clear, consistent improvement over fixed scheduling and therefore adopt the fixed scheduler as our default (Appendix D). Appendix E uses SmolVLA (Shukor et al., 2025) as a case study to show that VLA-ULAP retains higher success than simply executing more of each VLA-generated chunk at comparable VLA-call reduction.

**Lightweight ensemble.** For accuracy, we equally average denormalized, clipped chunks from two ULAP predictors (four for CALVIN), trained with different episode-level training/validation partitions. Frozen Theia features are computed once and shared; only lightweight predictor passes and averaging add computation. The VLA and fixed schedule remain unchanged.

# 4 Results

We test two linked benefits: retaining success with less VLA computation, and improving success when delayed actions encounter a changing scene. Section 4.1 tests four base-policy/benchmark pairs, including long-horizon task chains. Section 4.2 compares success, inference speedup, and energy efficiency against ACT-based local prediction, SP-VLA action extrapolation, and standalone RT-Cache retrieval. Section 4.3 tests whether VLA-ULAP remains effective on a real robot, while Section 4.4 separately profiles inference on Jetson Orin Nano and A6000. Section 4.5 returns to simulation to test whether lower latency also improves success in dynamic environments.

## 4.1 Simulation across base policies and benchmarks

**LIBERO $\times$ GR00T N1.7.** We first pair GR00T N1.7 (NVIDIA, 2026) with ULAP using fixed scheduling and 16-step chunks on all 40 tasks in LIBERO, a simulated robot-manipulation benchmark (Liu et al., 2023). Training and evaluation initial states are disjoint (Appendix B.1). In Figure 3, single-predictor VLA-ULAP achieves **88.50%** success (**95.0%** retention) with **50.7%** fewer VLA calls. The ensemble achieves **88.50%** success (**95.0%** retention) with **50.9%** fewer calls. Ensembling adds up to **2.75 pp** at high local fractions, with a mean gain of **0.99 pp** across all nine tested fractions.

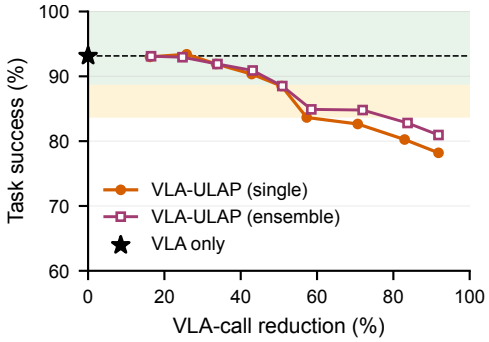



Figure 3: **LIBERO × GR00T N1.7.** Call reduction is relative to VLA only (star/dashed line). Green/amber bands show ≥95%/90–95% success-rate retention.

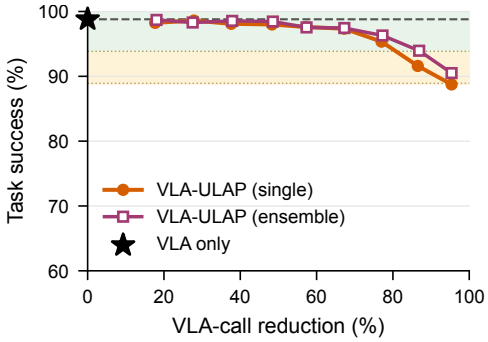



Figure 4: **LIBERO × VLA-JEPA.** Markers and bands follow Figure 3.

**LIBERO × VLA-JEPA.** To test another architecture, we pair VLA-JEPA (Sun et al., 2026) with ULAP using fixed scheduling and seven-step chunks on all 40 tasks, using disjoint training and evaluation initial states as in GR00T (Appendix B.2). In Figure 4, the single predictor and ensemble achieve **95.35%** and **96.30%** success (**96.5%** and **97.5%** retention) with **77.0%** and **77.3%** fewer VLA calls, respectively. Ensembling adds up to **2.35 pp** at high local fractions, with a mean gain of **0.68 pp** across all nine settings.

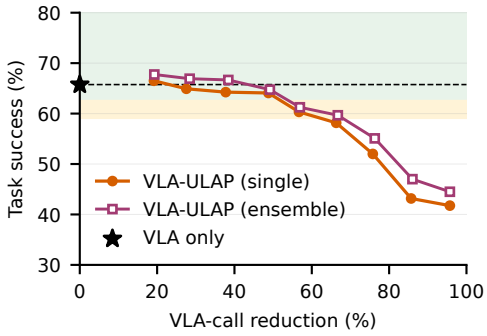



Figure 5: **RoboCasa × Cosmos-Policy.** Bands follow Figure 3.

**RoboCasa × Cosmos-Policy.** We next test more varied kitchen layouts and object placements in RoboCasa (Nasiriany et al., 2024). With Cosmos-Policy (Kim et al., 2026), we train a shared ULAP on 24 tasks using object split A and evaluate on split B, testing generalization to held-out objects (Appendix B.3).

Single-predictor and ensemble VLA-ULAP retain **97.5%** and **98.5%** of the baseline success rate (**64.08%** and **64.75%**), removing **48.8%** and **49.2%** of calls, respectively (Figure 5). Ensemble gains span **0.67–3.83 pp** across nine local fractions.

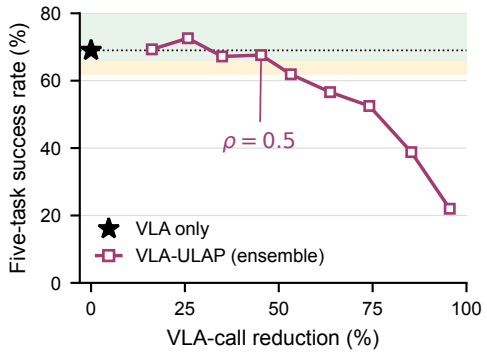



Figure 6: **CALVIN × X-VLA.** Five-task success rate. Bands follow Figure 3.

**CALVIN × X-VLA.** Additionally, we use CALVIN (Mees et al., 2022) to test long-horizon, multi-task execution through five instructions without resets. With X-VLA (Zheng et al., 2026) and a four-model ULAP ensemble at $\rho = 0.5$, VLA-ULAP removes **45.3%** of calls while retaining **98.0%** of baseline success rate (67.6% versus 69.0%; Figure 6). Mean completed tasks are maintained (4.176 versus 4.138), supporting effective bypass across long task chains. Appendix B.4 provides depth-wise results and settings.

### 4.2 Comparison with prior work

Having tested VLA-ULAP across models and benchmarks, we compare three alternatives. **ACT** (Zhao et al., 2023), a vision-based action-chunk imitation policy, tests a standard learned local path. **SP-VLA** (Li et al., 2026) tests action-history extrapolation using its scheduling component without token pruning. Both can supply actions beyond the VLA's last generated chunk; VLA-ULAP, ACT, and SP-VLA share the same VLA-JEPA base policy. **RT-Cache** (Kwon et al., 2025) instead tests standalone image-based action retrieval, using the same training rollout corpus without VLA calls. All methods are evaluated on the same LIBERO initial states.

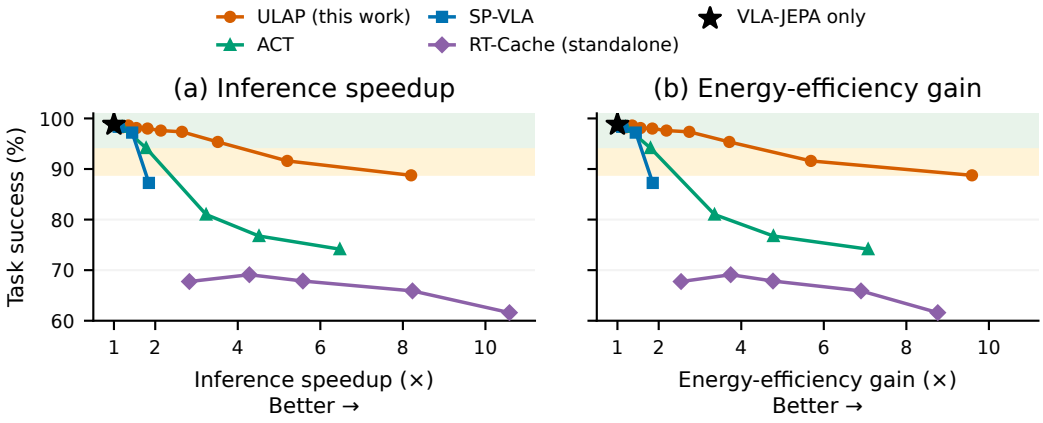


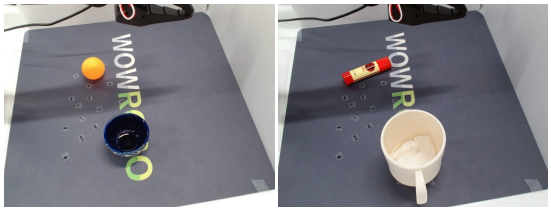

Figure 7: **Comparison with prior methods on LIBERO: success and inference efficiency.** Gains are VLA-JEPA-only cost divided by method cost, averaging successful episodes and using measured RTX A6000 branch costs. Larger gains lie to the right; VLA-JEPA alone is 1×. RT-Cache runs standalone with chunk lengths 4/6/8/12/16. Bands follow Figure 3; settings are in Appendix B.2.

Figure 7 compares success with inference speedup and energy-efficiency gain over VLA-JEPA, using mean costs per successful episode. VLA-ULAP achieves a better observed frontier than all three alternatives—ACT, SP-VLA, and RT-Cache. Compared with ACT at 94.15% success, VLA-ULAP achieves **95.35%** with **49.4% less inference time and 51.5% less GPU energy**. Outperforming ACT's 51.6M-parameter model supports ULAP's compact design. Compared with the most aggressive SP-VLA setting at 87.25% success, VLA-ULAP achieves 88.75% with **77.5% less time and 80.7% less energy**. SP-VLA's sharp decline with longer observation-free extrapolation highlights the value of using fresh observations. Compared with standalone RT-Cache's peak success of **69.10%** (chunk length 6), VLA-ULAP achieves **91.60%** with **17.7% less time and 34.2% less energy**. RT-Cache's low success on held-out placements suggests limited generalization beyond cache coverage.

### 4.3 Real robot × GR00T N1.7

Moving from simulation to a real robot, we evaluate two pick-and-place tasks on an SO-101 arm (Knight et al., 2024). The robot places a ping-pong ball into a bowl in one task and a glue stick into a cup in the other. Each task uses 100 human demonstration episodes to fine-tune GR00T; ULAP uses 80 for training, 10 for validation, and 10 for testing. Given the ensemble gains in simulation, we use ensemble ULAP with modest latency and energy overhead (Appendix B.5, Table 4). Local ULAP and cloud VLA both execute 16-step chunks, with requested local fraction 0.75. Each policy is evaluated on 75 trials, comprising 50 at training placements (ID) and 25 at held-out placements (OOD).

**Performance is retained at both in-distribution and held-out placements.** VLA-ULAP matches GR00T-only's OOD success rates of **88.0%** (22/25) on Ping-pong and **100.0%** (25/25) on Glue-stick, while achieving **96.0%** (48/50) at ID placements on both tasks.



Ping-pong: ball → blue bowl

Glue-stick: glue stick → white cup

Figure 8: **Physical pick-and-place tasks.** Initial external-camera views from evaluated trials.

Mean VLA calls per successful episode fall by **70.3%** on Ping-pong and **71.9%** on Glue-stick (Figure 9), with no observed success-rate decrease at either ID or OOD placements.

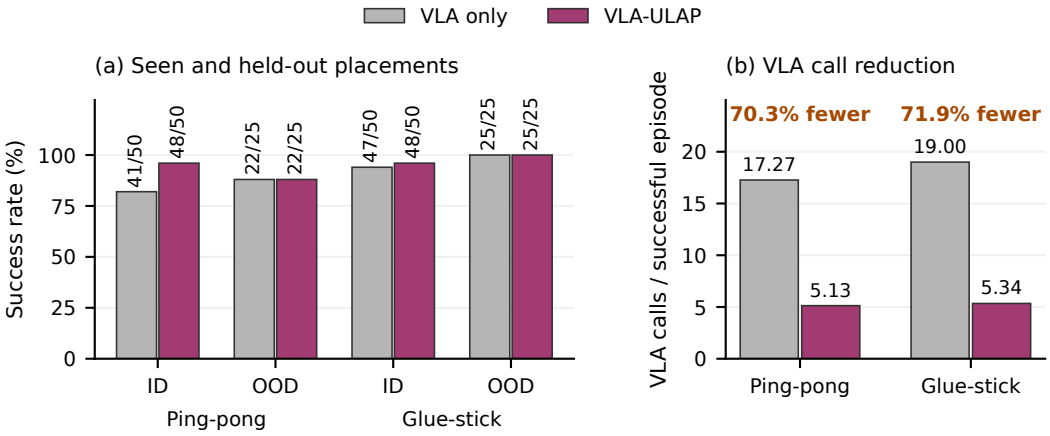



Figure 9: **Real-robot evaluation on SO-101 with ensemble ULAP.** Left: ID/OOD success rates, with successful/total trial counts. Right: mean VLA calls per successful episode and reduction from GR00T-only. Settings: Appendix B.5.

### 4.4 A LIGHTWEIGHT LOCAL FAST PATH

Physical rollouts used Mac MPS for ULAP; separate profiling measures its edge-device cost. On Jetson Orin Nano, ensemble ULAP takes **20.72 ms** and **0.122 J** of idle-subtracted energy per inference, versus **289.27 ms** and **40.46 J** for GR00T on A6000. This saves **92.8%** of inference time and **99.7%** of inference energy per local decision. Combining these measured costs with successful physical episodes' call counts, we estimate inference time reductions of **64.6%** on Ping-pong and **66.4%** on Glue-stick, with idle-subtracted GPU-plus-edge energy reductions of **70.1%** and **71.7%**, respectively (Figure 10). Thus, even on an inexpensive edge device, the local path delivers fast inference while substantially reducing inference energy consumption.

### 4.5 LIBERO-SAFETY × $\pi_{0.5}$: BEYOND FASTER EXECUTION

Finally, we ask whether the fast path offers more than computational savings: when the scene changes during inference, faster local updates may also improve task success. We test this benefit on two latency-sensitive tasks from LIBERO-Safety's dynamic obstacle-avoidance suite (Cui et al., 2026): placing both moka pots on a stove, and placing a mug in a microwave and closing its door. We model three control-step delays for the VLA and zero additional steps for local prediction, reflecting their measured latency scales (Appendix B.6). The simulator continues stepping during the modeled VLA delay. We compare $\pi_{0.5}$-only execution (Black et al., 2025) with VLA-ULAP replacing half of the scheduled decisions, using the same five-step execution interval and 200 episodes per task and policy (Appendix B.6).

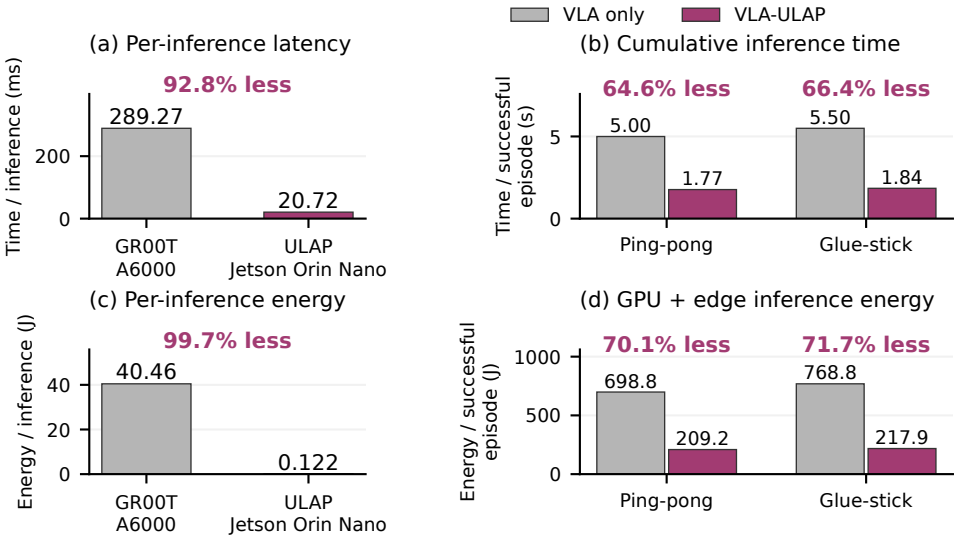



Figure 10: **Inference latency and energy for real-robot evaluation.** Left: measured costs of GR00T on A6000 and ensemble ULAP on Jetson Orin Nano. Right: estimated cumulative inference time and energy per successful episode. Energy sums idle-subtracted A6000 board and Jetson module costs; details are in Appendix B.5.

Table 1: **Evaluation on two LIBERO-Safety tasks under latency-aware execution.** Each condition uses 50 initial states with four repetitions. Gain is the absolute success-rate increase in pp.

| Task | $\pi_{0.5}$ only | VLA-ULAP | Gain |
|---|---|---|---|
| Moka pots → stove | 77.5% (155/200) | **88.5%** (177/200) | **+11.0 pp** |
| Mug → microwave | 50.0% (100/200) | **65.5%** (131/200) | **+15.5 pp** |

VLA-ULAP raises success by **11.0** and **15.5 pp**, while reducing total VLA calls by **51.8%** and **52.5%**, respectively (Table 1). Thus, the fast local path does not merely trade accuracy for cheaper inference: on these latency-sensitive tasks, it achieves both fewer heavy-policy calls and more successful executions.

## 5 Conclusion

VLA-ULAP combines remote VLA calls with ultra-lightweight local prediction to reduce communication delay and onboard energy demand. ULAP generates fresh chunks from current observations, proprioception, and executed action history, while periodic VLA calls help preserve success. Independence from VLA hidden states and online verification enables low-power edge deployment without joint training or costly transfers of large intermediate features during inference.

Across four simulation settings, selected points remove 45.3–77.3% of VLA calls while retaining 95.0–98.5% of baseline success rate. On VLA-JEPA, VLA-ULAP achieves a stronger success–efficiency trade-off than three representative approaches to reducing VLA calls. Physical SO-101 tests reduce VLA calls without observed success-rate loss under placement shifts, and Jetson Orin Nano ensemble measurements show the feasibility of fast, low-energy local inference. Beyond these savings, latency-aware LIBERO-Safety results show that faster updates can improve dynamic-task success.

Together, these results support decision-time partitioning that retains VLA capability while reducing remote requests. Low-power, low-latency local prediction enables responsive control and longer battery life, while freeing shared-GPU capacity for concurrent operation of larger fleets.

## AI USE STATEMENT

Generative AI tools assisted with developing the conceptual framing, refining hypotheses, providing feedback on method and experimental design, interpreting reported results, and translating author notes into English manuscript text. They also assisted with implementing ULAP and its training and evaluation code, as well as organizing and reformatting evaluation logs and implementing scripts for data auditing, aggregation, cost calculations, and figure generation. AI agents were also used to execute experimental commands, manage experiments, and diagnose and fix errors. Additional uses included literature discovery and synthesis, manuscript drafting, restructuring and editing, reference formatting, and creation and revision of scientific figures and conceptual illustrations. No generative-AI-generated synthetic data were used for training or evaluation. Generative AI was not used to formulate mathematical or theoretical claims or to write or verify proofs.

The authors reviewed the AI-assisted text, citations, result interpretations, and figures, and rigorously checked and tested the implementation code, including training and evaluation code. Numerical reporting is checked against recorded evaluation artifacts, and cited claims are checked against primary sources documented in a source ledger. Analysis scripts include consistency checks on aggregate counts and derived metrics, and rendered figures and manuscript layouts are inspected. The authors retain responsibility for the final text, methods, results, code, figures, and all AI-assisted claims and artifacts.

## ETHICS STATEMENT

Learned robot policies can produce unsafe actions when observations are unfamiliar or inaccurate. Our physical evaluation concerns bounded tabletop pick-and-place, retaining the existing joint-command clamps and episode timeout (Appendix B.5). Neither success at held-out placements nor the LIBERO-Safety simulation establishes safe human–robot collaboration. ULAP is not a safety controller; deployment near people requires independent safeguards and risk assessment. Reported energy savings concern inference devices, not the robot's full operational or life-cycle footprint.

## REPRODUCIBILITY STATEMENT

Section 3 specifies the predictor inputs, architecture, training objective, and scheduling. Appendices B and D give benchmark-specific training and evaluation settings, seeds, action horizons, and metric definitions. Appendix B.6 describes injected-delay execution; Appendix B.5 states the hardware measurement boundaries and episode-cost reconstruction. Per-task plots and success counts accompany these settings so that aggregate claims can be checked without conflating protocols.

## REFERENCES

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025. URL https://arxiv.org/abs/2503.14734.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert 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. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025. URL https://proceedings.mlr.press/v305/black25a.html.

Feng Chen, Xianghui Wang, Yuxuan Chen, Boying Li, Yefei He, Zeyu Zhang, and Yicheng Wu. Dynamic execution commitment of vision-language-action models. *arXiv preprint arXiv:2605.11567*, 2026. URL https://arxiv.org/abs/2605.11567.

Rongxu Cui, Zongzheng Zhang, Jingrui Pang, Haohan Chi, Jinbang Guo, Saining Zhang, Shaoxuan Xie, Xin Jin, Yao Mu, Jiaolong Yang, Guocai Yao, Xianyuan Zhan, Ya-Qin Zhang, and Hao Zhao. LIBERO-Safety: A comprehensive benchmark for physical and semantic safety in vision-language-action models. In *Computer Vision – ECCV 2026*, pp. 512–530, 2026. doi: 10.1007/978-3-032-37627-5_28. URL https://link.springer.com/chapter/10.1007/978-3-032-37627-5_28.

Xiangdong Feng, Yuxuan Cheng, Chen Shi, Boyao Han, Yuxuan Yan, Yitong Hong, Zhuotao Tian, and Li Jiang. Denoising tells when to replan: Denoising-variance adaptive chunking for flow-based robot policies. *arXiv preprint arXiv:2606.03847*, 2026. URL https://arxiv.org/abs/2606.03847.

Chengyang He, Xu Liu, Gadiel Sznaier Camps, Joseph Bruno, Guillaume Sartoretti, and Mac Schwager. Demystifying robot diffusion policies: Action memorization and a simple lookup table alternative. In *International Conference on Learning Representations*, volume 2026, pp. 10890–10917, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/122ea6470232ee5e79a2649243348005-Abstract-Conference.html.

Yiyang Huang, Yuhui Hao, Bo Yu, Feng Yan, Yuxin Yang, Feng Min, Yinhe Han, Lin Ma, Shaoshan Liu, Qiang Liu, and Yiming Gan. DaDu-Corki: Algorithm-architecture co-design for embodied AI-powered robotic manipulation. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025. doi: 10.1145/3695053.3731099. URL https://doi.org/10.1145/3695053.3731099.

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, 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, and Jinwei Gu. Cosmos Policy: Fine-tuning video models for visuomotor control and planning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/748becc400a57c0e31cfe6a2e7951467-Abstract-Conference.html.

Rob Knight, Pepijn Kooijmans, Remi Cadene, Simon Alibert, Michel Aractingi, Dana Aubakirova, Adil Zouitine, Russi Martino, Steven Palma, Caroline Pascal, and Thomas Wolf. Standard open SO-100 & SO-101 arms. Open-source hardware repository, 2024. URL https://github.com/TheRobotStudio/SO-ARM100.

Owen Kwon, Abraham George, Alison Bartsch, and Amir Barati Farimani. RT-Cache: Training-free retrieval for real-time manipulation. In *2025 IEEE-RAS 24th International Conference on Humanoid Robots (Humanoids)*, pp. 1131–1138, 2025. doi: 10.1109/HUMANOIDS65713.2025.11203198. URL https://ieeexplore.ieee.org/document/11203198.

Ye Li, Yuan Meng, Zewen Sun, Kangye Ji, Chen Tang, Jiajun Fan, Xinzhu Ma, Shutao Xia, Zhi Wang, and Wenwu Zhu. SP-VLA: A joint model scheduling and token pruning approach for VLA model acceleration. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/4072543747a14bbed76284cf2c04b9e9-Abstract-Conference.html.

Yuanchang Liang, Xiaobo Wang, Kai Wang, Shuo Wang, Xiaojiang Peng, Haoyu Chen, David Kim Huat Chua, and Prahlad Vadakkepat. Adaptive action chunking at inference-time for vision-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20802–20811, 2026. URL https:

//openaccess.thecvf.com/content/CVPR2026/html/Liang_Adaptive_Action_Chunking_at_Inference-time_for_Vision-Language-Action_Models_CVPR_2026_paper.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 36: Datasets and Benchmarks Track*, pp. 44776–44791, 2023. doi: 10.52202/075280-1939. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/8c3c666820ea055a77726d66fc7d447f-Abstract-Datasets_and_Benchmarks.html.

Yudong Liu, Yuan Li, Zijia Tang, Yuxi Zheng, Yueqian Lin, Qinsi Wang, Yi Li, Shuangjun Liu, Shuai Zhang, Taotao Jing, Dashan Gao, Ning Bi, Jingwei Sun, Yiran Chen, and Hai Li. Latent bridge: Feature delta prediction for efficient dual-system vision-language-action model inference. *arXiv preprint arXiv:2605.02739*, 2026a. URL https://arxiv.org/abs/2605.02739.

Ziyan Liu, Yeqiu Chen, Yiming Zhang, Hongyi Cai, Tao Lin, Runquan Gui, Shuo Yang, Zheng Liu, and Bo Zhao. Bridging the semantic-action gap in visual token pruning for efficient VLA inference. *arXiv preprint arXiv:2511.16449*, 2026b. URL https://arxiv.org/abs/2511.16449.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022. URL https://arxiv.org/abs/2112.03227.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-scale simulation of household tasks for generalist robots. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.050. URL https://www.roboticsproceedings.org/rss20/p050.html.

Jiahui Niu, Kefan Gu, Yucheng Zhao, Shengwen Liang, Tiancai Wang, Xing Hu, Ying Wang, and Huawei Li. Realtime-VLA FLASH: Speculative inference framework for diffusion-based VLAs. *arXiv preprint arXiv:2605.13778*, 2026. URL https://arxiv.org/abs/2605.13778.

NVIDIA. NVIDIA RTX A6000 datasheet. Official product datasheet, May 2022. URL https://www.nvidia.com/content/dam/en-zz/Solutions/products/workstations/nvidia-rtx-a6000-datasheet.pdf.

NVIDIA. GR00T-N1.7-3B model card. Hugging Face, 2026. URL https://huggingface.co/nvidia/GR00T-N1.7-3B.

Ryuji Oi, Hikari Otsuka, Kosuke Matsushima, Yuki Ichikawa, Masato Motomura, Tatsuya Kaneko, and Daichi Fujiki. ActionCache: Training-free acceleration for vision-language-action models with action caching and refinement. *arXiv preprint arXiv:2607.06370*, 2026. URL https://arxiv.org/abs/2607.06370.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=a68SUt6zFt.

Yi Pan, Miao Pan, Qi Lu, Jiaming Huang, Man Zhang, Siteng Huang, Xin Li, Jie Zhang, Yongliang Shen, Xuhong Zhang, and Wenqi Zhang. VLA-Corrector: Lightweight detect-and-correct inference for adaptive action horizon. *arXiv preprint arXiv:2607.01804*, 2026. URL https://arxiv.org/abs/2607.01804.

Daojie Peng, Fulong Ma, Bingtao Wang, Sheng Wang, and Jun Ma. Latency-tolerant cloud-edge collaborative vision-language-action models via emergent representational specialization. *arXiv preprint arXiv:2608.00569*, 2026. URL https://arxiv.org/abs/2608.00569.

Jinghuan Shang, Karl Schmeckpeper, Brandon B. May, Maria Vittoria Minniti, Tarik Kelestemur, David Watkins, and Laura Herlant. Theia: Distilling diverse vision foundation models for robot learning. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 724–748. PMLR, 2025. URL https://proceedings.mlr.press/v270/shang25a.html.

Suhas Hariharapura Sheshadri, Leela Subramaniam Karumbunathan, and Dustin Franklin. NVIDIA Jetson Orin Nano Developer Kit gets a "super" boost. NVIDIA Technical Blog, December 17, 2024.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. SmolVLA: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025. URL https://arxiv.org/abs/2506.01844.

Jingwen Sun, Wenyao Zhang, Zekun Qi, Shaojie Ren, Zezhi Liu, Hanxin Zhu, Guangzhong Sun, Xin Jin, and Zhibo Chen. VLA-JEPA: Enhancing vision-language-action model with latent world model. In *Computer Vision – ECCV 2026*, pp. 478–497, 2026. doi: 10.1007/978-3-032-37196-6_26. URL https://link.springer.com/chapter/10.1007/978-3-032-37196-6_26.

Xudong Tan, Yaoxin Yang, Peng Ye, Jialin Zheng, Bizhe Bai, Xinyi Wang, Jia Hao, and Tao Chen. Think twice, act once: Token-aware compression and action reuse for efficient inference in vision-language-action models. *arXiv preprint arXiv:2505.21200*, 2025. URL https://arxiv.org/abs/2505.21200.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Ramana Rao Kompella, and Gaowen Liu. VLA knows its limits: Adaptive execution horizons for robot policies. In *Computer Vision – ECCV 2026*, pp. 607–623, 2026. doi: 10.1007/978-3-032-37196-6_33. URL https://link.springer.com/chapter/10.1007/978-3-032-37196-6_33.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. VLA-Cache: Efficient vision-language-action manipulation via adaptive token caching. In *Advances in Neural Information Processing Systems 38*, pp. 182377–182402, 2025. doi: 10.52202/085713-5484. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/f062da1973ac9ac61fc6d44dd7fa309f-Abstract-Conference.html.

Zebin Yang, Qi Wang, Yunhe Wang, Xiurui Guo, Bo Yu, Shaoshan Liu, Jiafeng Xu, Hao Dong, and Meng Li. Jetson-PI: Towards onboard real-time robot control via foresight-aligned asynchronous inference. *arXiv preprint arXiv:2607.12659*, 2026. URL https://arxiv.org/abs/2607.12659.

Feng Ye, Yiming Zhao, Yong Yu, Hongxu Zhou, Yong Pan, Yuan Xue, Peng Jia, and Chuanmin Jia. Rethink before you execute: Adaptive execution for world action models. *arXiv preprint arXiv:2608.09492*, 2026. URL https://arxiv.org/abs/2608.09492.

Yang Yue, Yulin Wang, Bingyi Kang, Yizeng Han, Shenzhi Wang, Shiji Song, Jiashi Feng, and Gao Huang. DeeR-VLA: Dynamic inference of multimodal large language models for efficient robot execution. In *Advances in Neural Information Processing Systems 37*, pp. 56619–56643, 2024. doi: 10.52202/079017-1803. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/67b0e7c7c2a5780aeefe3b79caac106e-Abstract-Conference.html.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11975–11986, 2023. URL https://openaccess.thecvf.com/content/ICCV2023/html/Zhai_Sigmoid_Loss_for_Language_Image_Pre-Training_ICCV_2023_paper.html.

Jianke Zhang, Yanjiang Guo, Xiaoyu Chen, Yen-Jen Wang, Yucheng Hu, Chengming Shi, and Jianyu Chen. HiRT: Enhancing robotic control with hierarchical robot transformers. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 933–946. PMLR, 2025. URL https://proceedings.mlr.press/v270/zhang25b.html.

Jingxuan Zhang, Yunta Hsieh, Zhongwei Wan, Haokun Lin, Xin Wang, Ziqi Wang, Yingtie Lei, and Mi Zhang. QuantVLA: Scale-calibrated post-training quantization for vision-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 39539–39549, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Zhang_QuantVLA_Scale-Calibrated_Post-Training_Quantization_for_Vision-Language-Action_Models_CVPR_2026_paper.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://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.

# APPENDIX

## A EXTENDED RELATED-WORK TAXONOMY

Figure 11 places the five families in Section 2 under two objectives: reducing computation within each VLA call and reducing the number of full VLA calls. The latter has four branches: adaptive execution chunk length, action reuse and extrapolation, action retrieval, and cheaper learned policies.

### A.1 REDUCE COMPUTATION WITHIN A VLA CALL

This family reduces the cost of one invocation through low-bit execution (Zhang et al., 2026), token pruning or reuse (Liu et al., 2026b; Xu et al., 2025), adaptive depth (Yue et al., 2024), latent prediction (Liu et al., 2026a), and post-backbone action caching (Oi et al., 2026). These methods lower selected terms inside a call, but the remaining terms impose an Amdahl-style ceiling. They can complement VLA-ULAP by reducing computation in the remaining VLA calls.

### A.2 ADAPTIVE EXECUTION CHUNK LENGTH

This family builds on action chunking, in which a policy predicts a sequence of future actions in one call (Zhao et al., 2023). AutoHorizon reads the model's action self-attention to estimate its predictive limit (Wang et al., 2026); AAC uses action entropy to choose an execution prefix (Liang et al., 2026); DVAC chooses the low-denoising-variance prefix of one flow-policy output (Feng et al., 2026); and A3 accepts the longest prefix that remains conditionally consistent (Chen et al., 2026). VLA-Corrector truncates a chunk when latent visual dynamics indicate persistent deviation (Pan et al., 2026), while TempoWAM either continues the remaining world-action-model chunk or replans (Ye et al., 2026). These methods use different reliability signals but select from the chunk already generated; after that horizon, another policy call or a longer-horizon policy is required.

### A.3 ACTION REUSE AND EXTRAPOLATION

This family derives skipped actions from recent motion. FlashVLA, evaluated on OpenVLA (Kim et al., 2025), repeats the previous action when recent action directions and selected visual-token sets remain stable (Tan et al., 2025). SP-VLA fits a ridge regressor to a recent action buffer, extrapolates the next action, and reuses the preceding gripper state (Li et al., 2026). These paths are inexpensive, but their substitute action is inherited from earlier motion rather than selected from the state that now requires an action. Reuse and extrapolation can extend beyond the generated chunk, but sustained reliance on earlier motion can sharply degrade success. Our controlled SP-VLA comparison demonstrates this trade-off under aggressive call reduction (Section 4.2). ULAP instead predicts a fresh chunk conditioned on current observations, state, and executed action history.

### A.4 RETRIEVE EXECUTABLE ACTIONS

Action retrieval instead returns an executable sequence from memory without running the VLA on that decision. RT-Cache encodes the current RGB frame with frozen DINOv2 (Oquab et al., 2024) and SigLIP (Zhai et al., 2023), concatenates their features, and retrieves a multi-step trajectory snippet (Kwon et al., 2025). ALT trains a contrastive fusion encoder over the current first-person end-effector view, third-person view, and end-effector pose; cosine lookup returns the demonstration chunk linked to the nearest trajectory frame, while a similarity threshold flags unsupported observations (He et al., 2026).

Both methods face limited cache support. RT-Cache reports 0% success when no in-domain frame exists for the tested object–camera–pose combination, and image-key near-misses under occlusion, pose shifts, clutter, or scene dynamics (Kwon et al., 2025). ALT can detect an unsupported query and trigger a safe fallback, but does not supply the correct action chunk for that OOD state (He et al., 2026). RT-Cache also runs two frozen foundation vision encoders for every query and searches a 2,176-dimensional index. In contrast, ULAP predicts a new action chunk from observations and executed action history rather than selecting a stored chunk. VLA-ULAP interleaves these local pre-

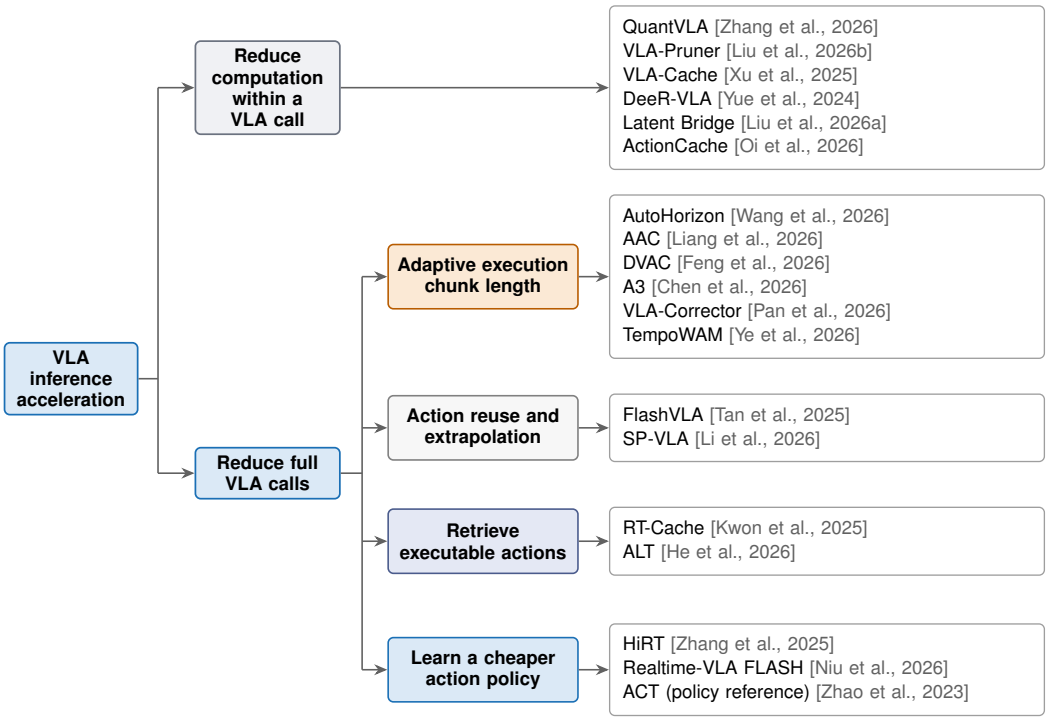



Figure 11: **Two objectives and five families of VLA inference acceleration methods.** In-call computation reduction forms one family; full-call reduction branches into the other four. ACT is a standard lightweight-policy reference, adapted in our comparison rather than originally proposed as a VLA accelerator. Author–year labels link to verified primary records.

dictions with calls to the unchanged base policy, using the VLA's broader generalization capabilities to help maintain task success under out-of-distribution conditions.

### A.5 Learn a cheaper action policy

HiRT combines a slowly updated VLM representation with a fast policy that conditions on observations and the cached latent (Zhang et al., 2025). When the VLM is remote, refreshing that latent requires feature transfer. Realtime-VLA FLASH uses a draft model coupled to the main image encoder and Action Expert verification (Niu et al., 2026), rather than a fully independent local path. In contrast, ULAP predicts complete chunks without base-VLA hidden states or online verification and is trained independently of the frozen VLA.

ACT predicts action chunks from images and robot state through imitation learning (Zhao et al., 2023). It is not originally a VLA-call scheduler; we adapt it as a standard lightweight action-chunk predictor under the same hybrid schedule as ULAP, testing the local model rather than a different call-allocation rule (Section 4.2).

# B ULAP TRAINING AND EVALUATION SETTINGS

**Predictor architecture.** Table 2 specifies the shared predictor blocks. Let $H$, $D_a$, and $D_p$ denote the executed chunk length, action dimension, and proprioception dimension. Their values are $(16, 7, 8)$ for GR00T, $(7, 7, 8)$ for VLA-JEPA, $(16, 7, 9)$ for Cosmos-Policy, $(16, 6, 6)$ for SO-101, and $(5, 7, 8)$ for the LIBERO-Safety predictor. The benchmark subsections below specify the camera and task-conditioning inputs.

Table 2: **ULAP predictor specification.** Linear layers include bias; dropout is disabled at inference. The frozen visual encoder is Theia-Tiny.

| Block | Structure |
|---|---|
| Input projections | Visual tokens: LayerNorm then $192 \to 192$ linear. Actions and state: $D_a \to 192$ and $D_p \to 192$ linear. Learned position/type embeddings and a 192-D task vector. |
| Transformer | 3 pre-LayerNorm layers, width 192, 8 attention heads, FFN width 768, GELU, dropout 0.15; final LayerNorm. Aggregation-token projection $192 \to 128$, then $\ell_2$ normalization. |
| State MLP | $(HD_a + D_p) \to 512 \to 512 \to 128$; GELU after the first two layers, dropout 0.15 after the first GELU, then $\ell_2$ normalization. |
| Fusion | Concatenate both 128-D outputs and divide by $\sqrt{2}$ (256-D). |
| Action head | Concatenate each learned 192-D step token with the fusion vector; LayerNorm(448), $448 \to 192$, GELU, dropout 0.15, $192 \to D_a$. All $H$ steps share this head. |

Cosmos-Policy additionally projects its precomputed 1024-D language vector through LayerNorm and a $1024 \to 192$ linear layer and adds it to the task-conditioned aggregation token.

**Optional ensemble.** We use two predictors by default (four for CALVIN, Appendix B.4), with the same architecture and optimization settings. Diversity comes from repartitioning training/validation episodes within each task while preserving their counts; all samples from an episode remain in the same partition. The test set is unchanged. Each model computes normalization statistics from its own training partition and selects its checkpoint using validation macro-task standardized RMSE.

The predictors receive the same inputs and share frozen Theia features computed once per local decision. Each action chunk is denormalized using its model-specific statistics and clipped to the action bounds before equal-weight FP32 averaging across all steps and action dimensions, including the gripper. The averaged actions are executed and supply the subsequent action history. The base VLA, initial VLA call, and fixed schedule remain unchanged. Additional computation consists of lightweight predictor passes and action averaging, without repeating visual encoding.

## B.1 LIBERO × GR00T N1.7

**Evaluation.** We use NVIDIA GR00T-N1.7-LIBERO on four suites of ten tasks each. Every condition evaluates official initial-state IDs 0–49 per task (2,000 episodes), with environment seed 7 and policy seeds 7–56. Both paths execute 16-step chunks; the episode limit is 720 control steps. The fixed scheduler evenly spreads local decisions, accounting for the mandatory first VLA chunk. Requested local fractions are 0.2–1.0 in increments of 0.1, plus the VLA-only reference. All nine settings use the same suite-specific checkpoints and initial-state protocol.

**Training.** Suite-specific predictors are trained on successful GR00T rollouts collected with a 100-rollout budget per task, using episode-level splits and training-only normalization. Training uses Smooth-L1 ($\beta = 1$), AdamW (learning rate $6 \times 10^{-4}$, weight decay 0.05), batch 96, 120 epochs, eight warmup epochs followed by cosine decay, bf16 training autocast with FP32 parameters and validation, and seed 0; validation selects the checkpoint without refitting. Inputs are current primary/wrist images, proprioception, executed action history, and task conditioning. Predictor-training initial states are disjoint from evaluation.

**Input ablations (Appendix C.1).** Both variants retain current images and task conditioning and use the same data, optimizer, training seed, checkpoint selection, and evaluation protocol as the full model. Removing the state MLP retains history and proprioception in the Transformer; removing action history removes it from both branches while retaining proprioception. Each of the nine fixed-schedule settings evaluates 2,000 episodes against the same VLA-only reference (1,863 successes; 24,466 calls).

**Metrics (simulation evaluations).** Success rate is the fraction of evaluation episodes that succeed. VLA-call reduction compares total VLA calls across all evaluation episodes, including both successful and failed trials, against the VLA-only baseline: $1 - \sum_i C_i^{\text{hybrid}} / \sum_i C_i^{\text{exact}}$. This is not the local-decision fraction. It captures the realized reduction in total VLA calls during execution, including changes in episode length.

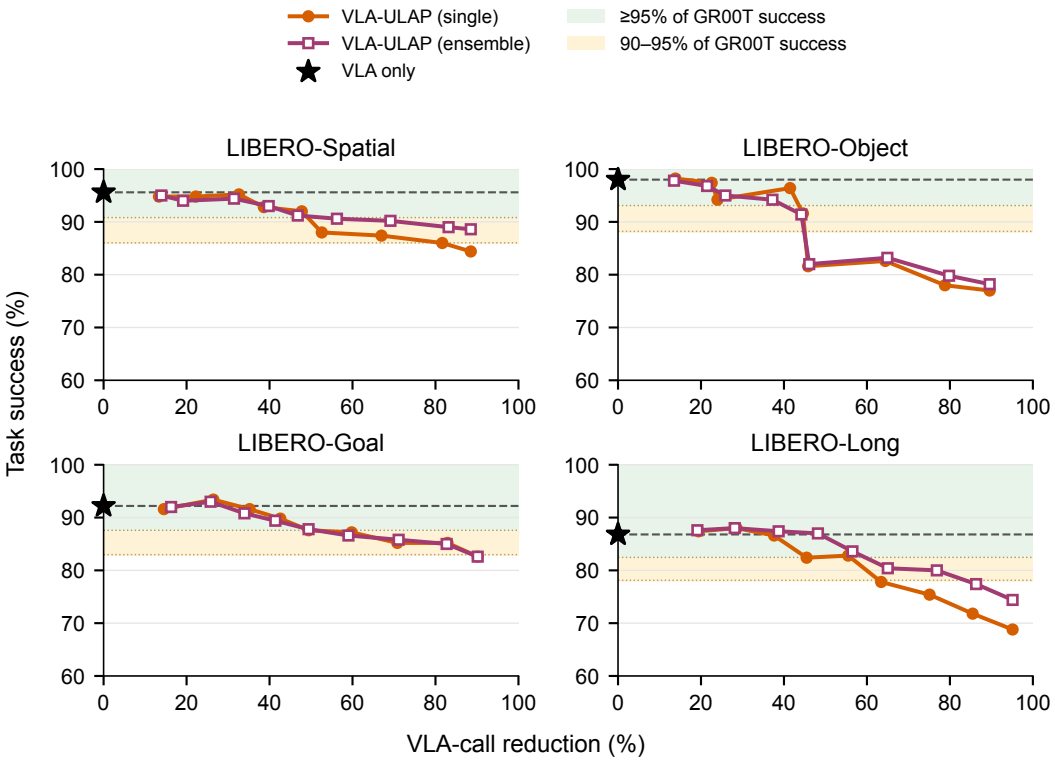



Figure 12: **GR00T N1.7 with ULAP by LIBERO suite.** Each point covers 500 episodes. Stars/dashed lines mark the baseline success rate. Green and amber bands indicate ≥95% and 90–95% retention. Orange circles show the single predictor; raspberry squares show the two-model ensemble. Both use fixed scheduling.

**Ensemble results.** The two-model ensemble uses the same nine local fractions and paired initial states as the single predictor. At local fraction 1.0, success increases from 78.20% to 80.95%, with both methods reducing VLA calls by 91.83%. The corresponding gains on LIBERO-Long, Object, Goal, and Spatial are 5.60, 1.20, 0.00, and 4.20 pp, respectively.

### B.2 LIBERO × VLA-JEPA

**Evaluation.** We use the released VLA-JEPA LIBERO checkpoint with its state-conditioned action head, ten DDIM steps, and Franka action normalization. Each condition evaluates the same 40 tasks and official initial-state IDs 0–49 (2,000 episodes), with environment seed 7 and ten settling steps. Both paths execute seven-step chunks. Episode limits are 250, 280, 300, and 520 control steps for Spatial, Object, Goal, and Long, respectively. VLA-call seeds depend on the base seed, suite, trial, and control-step index, independently of the requested local fraction. The fixed schedule and requested fractions match Appendix B.1. ULAP predicts actions directly in bf16, without running the VLA action head.

**Training.** Four suite-specific predictors are trained on successful VLA-JEPA rollouts: 3,912 trajectories from 100 attempts per task. The recorded simulator-state audit confirms no overlap with evaluation initial states ($L_\infty$ tolerance $10^{-4}$). Examples pair current primary/wrist camera views, current proprioception, and the previous executed chunk with the next action chunk. Task-local episode splits are 70/15/15% for training, validation, and test, with training-only normalization. Optimization uses the same loss, AdamW settings, batch size, epoch count, warmup/cosine schedule, and bf16 precision as GR00T, but training seed 7. Validation selects one checkpoint per suite, without refitting. The predictor has 1,871,815 trainable parameters, excluding frozen Theia. Previous images are omitted: 98 visual tokens and nine action/state/aggregation tokens give 107 input tokens. The four predictors are trained from initialization with the same recipe on their respective suites' rollouts.

**Ensemble.** The two-model ensemble follows the shared setup in Appendix B. For each suite, the second predictor uses an episode-level training/validation repartition (split seed 20260923), training seed 0, 120 epochs, and batch size 96. The test split is excluded, normalization uses training data only, and validation selects the checkpoint without refitting. The averaged action chunk is passed through the evaluator's usual gripper binarization. We evaluate the same initial states at nine local fractions from 0.2 to 1.0.

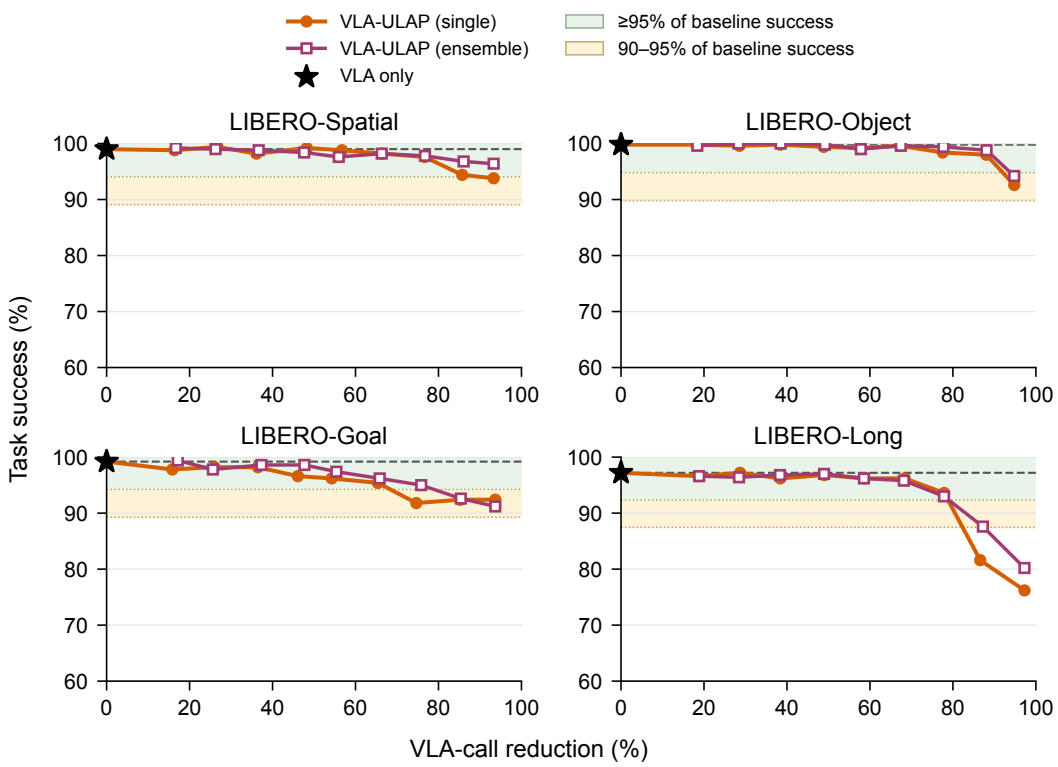



Figure 13: **VLA-JEPA with ULAP by LIBERO suite.** Each point covers 500 episodes. Each panel uses its own VLA-only call total and baseline success rate (star/dashed line). Green and amber bands indicate $\geq$95% and 90–95% retention. Filled circles denote single predictors; open squares denote ensembles.

**ACT, SP-VLA, and RT-Cache comparison.** All methods use the same 40 tasks, 50 initial states per task, environment seed, and episode limits as above; the hybrid methods share the same VLA checkpoint. ACT (Zhao et al., 2023) uses four suite-specific validation-selected policies with current front/wrist images, proprioception, and task identity. ACT is trained on the same successful VLA-JEPA rollout corpus and episode splits as ULAP, with the same 120-epoch budget. The LeRobot ACT implementation uses an ImageNet-initialized ResNet18, a width-512 Transformer, and CVAE training. Inference uses FP32 and no temporal ensembling, with the CVAE latent variable $z$ fixed to its prior mean of zero rather than sampled. ACT replaces the local branch under the same fixed scheduler at fractions 0.2/0.5/0.8/0.9/1.0, generating and executing seven actions per query. Even at 1.0, the initial query uses the VLA. This is our ACT-based bypass baseline, not the original ACT evaluation protocol.

SP-VLA's scheduling component (Li et al., 2026) uses seven-step exact chunks; local ridge extrapolation generates one action from six previous actions ($\lambda = 10^{-4}$). We disable token pruning to focus on reducing complete VLA calls rather than computation within each call. The three plotted settings comprise the official freshness/motion gates and two experimental high-reduction variants: motion thresholds scaled by $\times 8$ and $\times 32$, with consecutive extrapolation capped at 4 and 8 steps, respectively, replacing the official requirement for four exact actions among the last six. These variants were calibrated on 160 successful training/validation trajectories disjoint from evaluation, then frozen.

RT-Cache (Kwon et al., 2025) retrieves from task-specific memories built from the same corpus's 2,740 training-split successful teacher rollouts. It encodes the current primary-camera image with frozen DINOv2 and SigLIP, normalizes the concatenated 2,176-D feature, and averages the top-five chunks under exact cosine similarity. We sweep chunk lengths 4/6/8/12/16, constructed from

executed teacher actions. Retrieval starts at the first decision, without VLA calls or fallback. This LIBERO implementation uses in-memory search rather than RT-Cache's database-serving stack.

**Successful-episode inference costs.** For each successful episode $i$, we estimate $T_i = N_{V,i}t_V + N_{L,i}t_L$ and $E_i = N_{V,i}e_V + N_{L,i}e_L$. Here, $T_i$ and $E_i$ are the episode's estimated total inference time and GPU-board energy. $N_{V,i}$ and $N_{L,i}$ count VLA and local-path invocations in episode $i$, respectively. $t_V, t_L$ are the measured mean times per invocation, and $e_V, e_L$ are the corresponding mean GPU-board energies. The local path is ULAP, ACT, SP-VLA extrapolation, or RT-Cache retrieval. For standalone RT-Cache, $N_{V,i} = 0$ and local costs are task- and chunk-specific. We then average $T_i$ and $E_i$ over that method's successful episodes. Counts include the initial query and any partially executed final chunk. Batch-one RTX A6000 profiles give 212.195 ms/27.3115 J per VLA query, 24.503 ms/2.7668 J per ACT query, and 17.043 ms/1.6741 J per ULAP query including Theia. SP-VLA costs 0.176–0.177 ms and 0.00402–0.00417 J per extrapolated action, using the stable final two profiling blocks. ULAP uses the two current camera views; its profile pools 7,011 calls across six blocks in two sessions, each following 300 warm-up calls.

Energy includes gross GPU-board consumption during inference, including resident idle power for CPU-based SP-VLA, but excludes CPU/host energy, simulation, communication, server queuing, and route-switch transients.

### B.3 RoboCasa × Cosmos-Policy

**Evaluation.** We use NVIDIA Cosmos-Policy-RoboCasa-Predict2-2B in bf16 with TF32 disabled and five denoising steps. The base policy generates 32 actions and executes the first 16; ULAP predicts and executes 16 actions. Each condition evaluates 24 tasks across five (layout, style) pairs, $(1, 1)$, $(2, 2)$, $(4, 4)$, $(6, 9)$, and $(7, 10)$, with ten episodes per pair (1,200 episodes). Object split B is used throughout evaluation. Environment seeds vary by task and episode, while the VLA seed is 195. The same identities, initial-state hashes, and object hashes are matched across conditions. Ten settling steps precede control, which terminates on success or at the official task-specific step limit. We use the fixed scheduler with requested local fractions $0.2, 0.3, \ldots, 1.0$, always starting with one VLA call. All nine fractions use the same checkpoint and 1,200 initial conditions.

**Training.** A single predictor covers all 24 tasks, using current primary, secondary, and wrist views, nine-dimensional proprioception, executed action history, and precomputed frozen language embeddings. Successful teacher rollouts from object split A provide 106,061 training examples at stride four; validation and test splits are separated by collection episode, with per-task training-only normalization. We use standardized Smooth-L1 loss ($\beta = 1$), AdamW (learning rate $6 \times 10^{-4}$, weight decay 0.05), global batch size 192, task-balanced sampling, 120 epochs, eight warmup epochs followed by cosine decay, bf16, and seed 0. The predictor has 2,118,791 trainable parameters, excluding frozen encoders. The Transformer input consists of 165 tokens.

**Task-level results.** Figure 14 shows single-predictor and ensemble results across all 24 tasks; the ensemble covers requested local fractions 0.2–1.0. For the single predictor, at requested local fraction 0.5, success is 769/1,200 (64.08%), or 97.47% of baseline, with 48.79% fewer VLA calls. At local fraction 0.6, the higher reduction of 56.69% retains 91.76% of baseline success (724/1,200). The ensemble also preserves success across individual tasks. With an operating point selected separately for each task, it retains at least 90% of VLA-only success while reducing VLA calls by at least 26.63% on every task, including reductions of at least 40% on 19 of 24 tasks. The small success gain at local fraction 0.2 is a point estimate, not evidence of general superiority over the base policy.

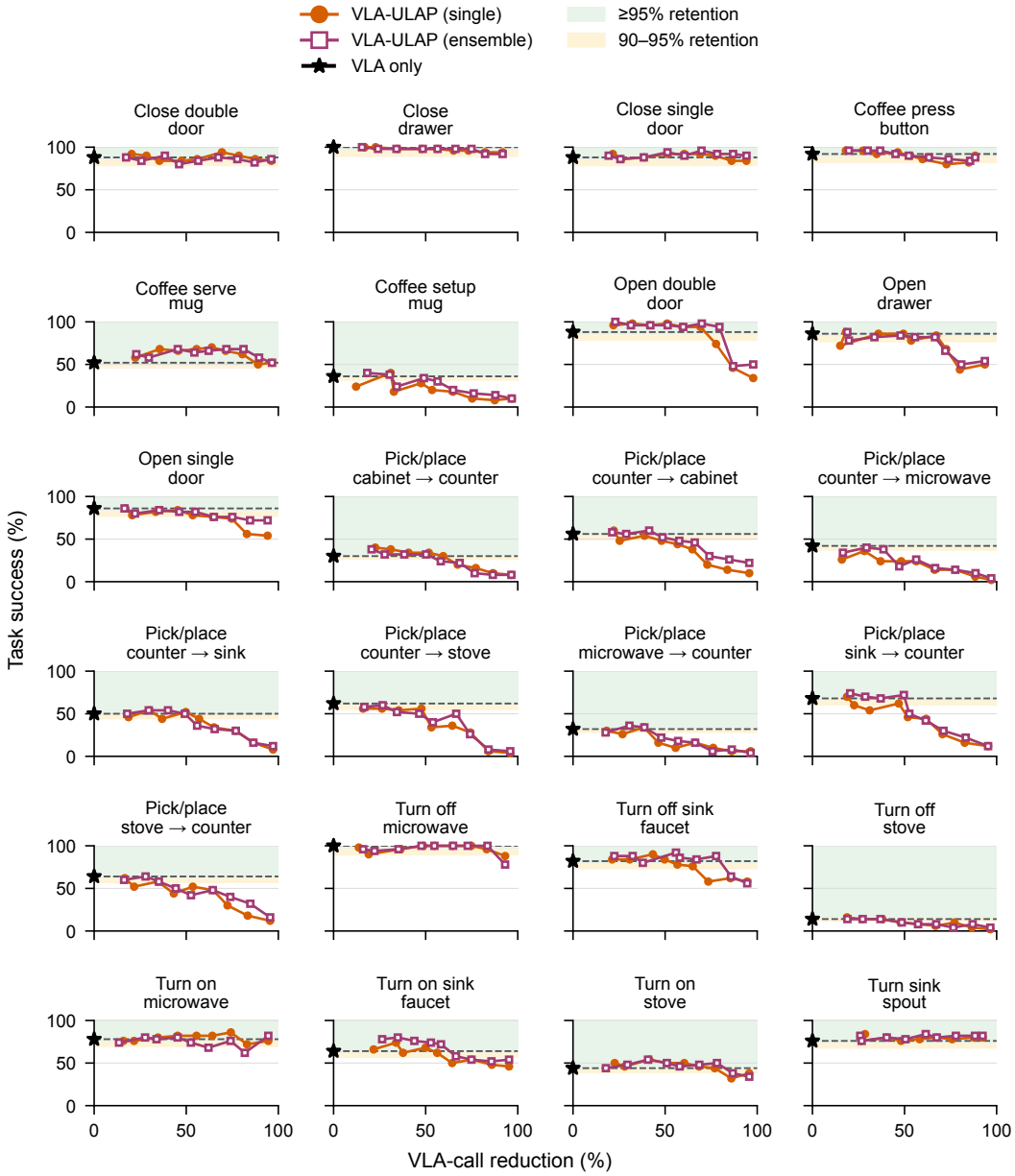



Figure 14: **RoboCasa success across all 24 tasks.** Each point uses 50 matched episodes. The black star and dashed line mark VLA-only success; bands indicate $\geq 95\%$ and 90–95% retention. Orange circles show the single predictor and raspberry squares the two-model ensemble, both with fixed scheduling. Call reduction is measured against the corresponding task baseline.

### B.4 CALVIN × X-VLA

**Motivation.** We use CALVIN (Mees et al., 2022) to evaluate long-horizon, multi-task execution and generalization across environments. Five language instructions are executed consecutively without resetting the environment, so each task inherits the state and any prediction errors left by earlier tasks. Predictors trained on environments A, B, and C are evaluated in D, which changes scene textures and the positions of fixtures such as drawers, sliding doors, and switches. This tests transfer of known manipulation skills to a different visual appearance and spatial layout. We pair ULAP with X-VLA (Zheng et al., 2026) and use a four-model ensemble to improve robustness against error accumulation over these longer chains.

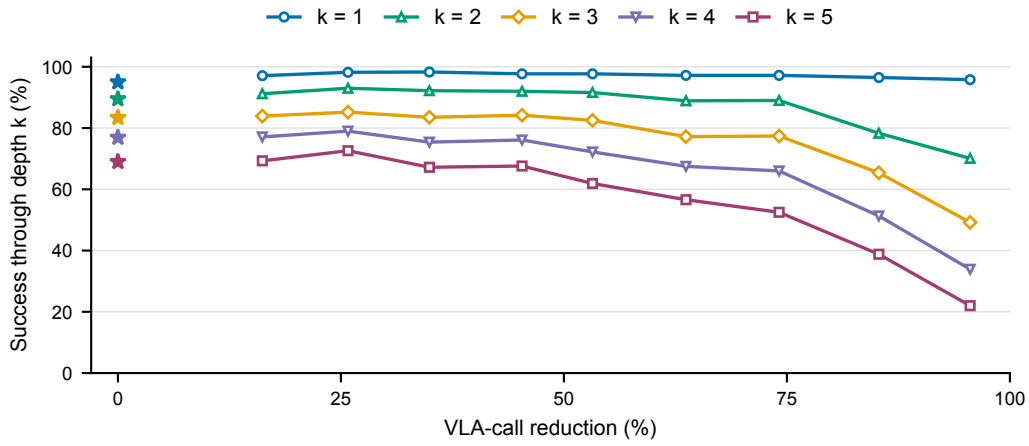



Figure 15: **CALVIN results by instruction depth.** Success rate for completing the first $k$ instructions consecutively, with one curve per depth $k = 1, \ldots, 5$. VLA-call reduction uses all 1,000 chains. Stars mark VLA-only baselines in matching depth colors.

**Nearly half the calls removed with strong chain completion.** At requested local fraction $\rho = 0.5$, VLA-ULAP reduces VLA calls by **45.3%** while retaining **98.0% of the baseline five-task success rate** (Figure 6). Mean completed tasks do not decrease, reaching 4.176 versus 4.138 for VLA only.

Figure 15 shows that at $\rho = 0.5$, even the longest chains incur only a small success-rate decrease, at most 1.4 pp across all depths. With more aggressive skipping, however, performance declines more strongly as the number of chained tasks grows, consistent with prediction errors accumulating during extended local execution. These results support substantial call reduction in long-horizon execution while identifying the growing importance of VLA calls as chains lengthen. Each lightweight predictor also shares its parameters across all 34 task categories, using language to select the skill rather than requiring a separate network per task. The ensemble combines four such shared predictors.

**Training and evaluation settings.** We evaluate the same 1,000 official five-instruction chains in environment D with the frozen X-VLA checkpoint fine-tuned on ABC. Each instruction has a 360-control-step budget at 30 Hz; a timeout ends the chain. Both paths execute 20-step chunks. The fixed scheduler starts with a VLA call and retains its state across instruction changes, while discarding any unexecuted actions for the previous instruction.

For chain $i$, let $L_i \in \{0, \ldots, 5\}$ be the number of instructions completed consecutively before failure or full completion. At instruction depth $k$, success means completing all of the first $k$ instructions. With $N = 1{,}000$ evaluated chains, its rate and the mean completed-task count are

$$S_k = \frac{1}{N} \sum_{i=1}^{N} I(L_i \geq k), \qquad \overline{L} = \frac{1}{N} \sum_{i=1}^{N} L_i = \sum_{k=1}^{5} S_k,$$

where the indicator $I(\cdot)$ equals one when its condition holds and zero otherwise. Figure 15 plots $100S_k$ for each $k$. Every depth uses all $N$ chains, including those that fail earlier. Call reduction likewise uses total VLA calls over all chains, relative to the matched VLA-only baseline.

Each ensemble member is trained on successful subtask segments from 960 ABC rollout sequences, with 120 additional sequences for validation. The four members use different trajectory-level splits of the same 1,080-sequence pool. No D trajectories enter predictor training. Current static and wrist views, proprioception, and 20 executed actions are supplemented by a frozen 384-dimensional language embedding, cached per instruction. The shared Theia-Tiny encoder supplies $7 \times 7$ tokens per view. Each member has 2.01M trainable parameters across a three-layer, width-192 Transformer, a state MLP, and a flat one-pass chunk head. Training uses masked Smooth-L1 loss and AdamW for 120 epochs, batch size 96, learning rate $6 \times 10^{-4}$, and eight warmup epochs followed by cosine decay. Decoded position and gripper predictions are averaged; averaged rotations are projected onto $\mathrm{SO}(3)$.

### B.5 REAL ROBOT × GR00T N1.7

**Protocol.** Each task uses a task-specific GR00T N1.7 checkpoint fine-tuned on 100 human demonstrations from ten object placements (P1–P10), selected at training step 2,000. The paired evaluation uses five trials at each P1–P10 placement and five at each held-out H1–H5 placement, in the same frozen trial order for both policies. OOD here denotes unseen object placements, not unseen object categories or tasks. Control is synchronous at a target 30 Hz, with a common fixed home pose, unchanged safety clamps, and a 60-second timeout. Success requires the object to remain in its target container, released and clear of the gripper, for two seconds. Both paths execute 16 actions per decision. The first decision uses GR00T, followed by the fixed evenly spread schedule with requested local fraction 0.75 for both tasks. Only final valid trials after infrastructure-failure recollection are counted.

**Predictor and training.** Frozen Theia features from current and previous external/wrist images are fused with the measured six-dimensional state and the previous 16 executed actions. Normalized Transformer and state-MLP branches have fixed equal weight; a parallel temporal head predicts a $16 \times 6$ action chunk. The five body axes are relative to the current measured state, while the gripper target is absolute. Each task's 100 human demonstrations are split 80/10/10 by episode for training/validation/test, with stride-four examples and training-only normalization. Training uses standardized Smooth-L1 loss ($\beta = 1$), AdamW (learning rate $6 \times 10^{-4}$, weight decay 0.05), batch 96 and 120 epochs, with eight warmup epochs and cosine decay. Two independently trained predictors (seeds 7 and 0) share frozen Theia features. Each member decodes and clips its absolute actions before their equal-weight mean is executed, including the gripper channel.

**Cloud VLA-call reduction.** For efficiency and safety, clearly failed physical runs were stopped early. We therefore compare mean cloud VLA calls per successful episode under VLA-ULAP and GR00T-only, respectively: $R = 1 - \overline{C}_{\mathrm{ULAP,\,success}}/\overline{C}_{\mathrm{GR00T,\,success}}$. These calls invoke the base VLA on the remote GPU. Ping-pong uses 1,088 cloud VLA calls across 63 baseline successes and 359 across 70 VLA-ULAP successes; Glue-stick uses 1,368 across 72 and 390 across 73, respectively. Table 3 compares single-predictor and ensemble results. Despite using a higher requested local fraction (0.75 versus 0.60 and 0.67), the ensemble completes more trials successfully on both tasks.

Table 3: **Single and ensemble physical evaluations.** Cloud VLA calls are means per successful episode.

| Task | Policy | Local fraction | ID | OOD | Cloud VLA calls |
|---|---|---|---|---|---|
| Ping-pong | GR00T | 0 | 41/50 | 22/25 | 17.27 |
| | Single | 0.60 | 40/50 | 20/25 | 8.23 |
| | Ensemble | 0.75 | 48/50 | 22/25 | 5.13 |
| Glue-stick | GR00T | 0 | 47/50 | 25/25 | 19.00 |
| | Single | 0.67 | 48/50 | 24/25 | 7.07 |
| | Ensemble | 0.75 | 48/50 | 25/25 | 5.34 |

**A6000–Jetson cost model.** We profile the SO-101 Glue-stick GR00T checkpoint on RTX A6000 at batch one, with four denoising steps and its bf16-compute/FP32-parameter configuration. The path includes image packing, preprocessing, model inference, action decoding and CPU output of 16 actions. GR00T computes its native 40-position horizon and returns the first 16 actions. Both devices use the same saved Glue-stick validation query. After 30 warmup calls and thermal stabilization, 300 synchronized calls average 289.27 ms for GR00T and 20.72 ms for ensemble ULAP. Jetson Orin Nano uses 15 W DVFS, FP32/TF32 CUDA Graphs, shared Theia and separate streams for the two predictor members.

Energy is idle-subtracted on both devices. We integrate power over a separate continuous-inference window and subtract resident-model idle power measured before and after it. For active duration $T$ and $N$ completed calls, $e = (E_{\mathrm{active}} - P_{\mathrm{idle}}T)/N$. This gives 40.462 J per GR00T call from NVML GPU-board power and 0.122 J per ensemble ULAP call from Jetson VDD_IN module power (CPU, GPU and DRAM). Table 4 compares single and ensemble inference under the same DVFS

configuration and saved input. Sharing frozen vision keeps the ensemble's full-path latency overhead to 5.0% and energy overhead to 7.1%.

Table 4: **Measured Jetson Orin Nano costs.** Latency is the mean of 300 calls; energy is idle-subtracted per completed inference in a separate continuous window.

| Local predictor | Mean latency (ms) | Energy (J) | Active / idle power (W) |
|---|---|---|---|
| Single | 19.73 | 0.1140 | 11.01 / 5.69 |
| Ensemble (default) | 20.72 | 0.1221 | 11.12 / 5.68 |

For each policy, we average cloud VLA and local ULAP invocation counts over its successful episodes and use $\widehat{C} = \overline{N}_{\mathrm{VLA}} c_{\mathrm{A6000}} + \overline{N}_{\mathrm{ULAP}} c_{\mathrm{Jetson}}$. Here, $\overline{N}_{\mathrm{VLA}}$ and $\overline{N}_{\mathrm{ULAP}}$ are the mean numbers of cloud VLA calls and local ULAP predictions per successful episode, respectively. $c_{\mathrm{A6000}}$ and $c_{\mathrm{Jetson}}$ are the measured time or energy per call on the respective devices, and $\widehat{C}$ is the estimated mean total inference time or energy per successful episode. Camera acquisition, communication, queueing and actuation are outside the estimate. Energy also excludes the A6000 host and subtracts device idle energy within active windows.

**Illustrative deployment profiles.** To illustrate the broad differences in compute capability, power range, and price between the two devices, Table 5 summarizes their published specifications. This complements the measured application costs in Figure 10.

Table 5: **Remote GPU and local edge-device specifications.** NVIDIA's A6000 datasheet and Jetson announcement (NVIDIA, 2022; Sheshadri et al., 2024). Peak compute uses different precision/sparsity and is not a speedup comparison. The A6000 price is an indicative GPU-only retail quote (September 2026); the Jetson price is the announced developer-kit offer. Our ULAP measurements use 15 W DVFS, not the highest Super power mode.

| | RTX A6000 | Jetson Orin Nano Super (8GB) |
|---|---|---|
| Role in VLA-ULAP | Remote VLA GPU | Local ULAP device |
| Memory | 48 GB GDDR6 | 8 GB LPDDR5 |
| Memory bandwidth | 768 GB/s | 102 GB/s |
| Published peak compute | 38.7 TFLOPS (FP32) | 67 TOPS (sparse INT8) |
| Specified power | 300 W maximum board | 7 / 15 / 25 W module modes |
| Indicative price | ∼US$6,400 (GPU only) | US$249 (kit, December 2024) |

Figure 1 uses the measured physical Glue-stick profiles from the A6000–Jetson cost model above. Both paths output $16 \times 6$ action chunks. GR00T N1.7 on RTX A6000 takes 289.3 ms and 40.46 J per inference, while ensemble ULAP on Jetson Orin Nano takes 20.7 ms and 0.122 J. Energy covers the GPU board for A6000 and the VDD_IN module for Jetson, including its CPU, GPU, and DRAM; both values subtract resident-model idle power. Energy per inference is obtained from continuous-inference windows, rather than multiplying separately profiled latency by power.

The communication scale in Figure 1 follows DaDu-Corki's RoboFlamingo setup, with a Franka Panda communicating with a V100 server over Wi-Fi (Huang et al., 2025). Figure 2 of Huang et al. (2025) shows image-transfer latency of tens of milliseconds and energy of hundreds of millijoules per frame. These are coarse visual readings of its plot. The study measures actual frame transfers and integrates measured power over time. These communication-stage values are not a separately verified observation-upload/action-return round trip, and do not enter our inference-device cost estimates. Figure 1 reports the mean success rate over the two dynamic tasks, the Glue-stick successful-episode sums of VLA and ULAP inference costs, and the measured per-inference branch costs described above. Its episode costs use both paths, not just avoided VLA calls. Battery-life benefits concern low-power ULAP deployment instead of carrying a full-VLA GPU onboard. The drawing's robot counts illustrate the capacity opportunity, not measured server limits. With fewer requests per robot, a shared service can devote more compute time to other clients; the feasible robot count also depends on request deadlines, queueing, batching, and the set of resident policies. The

affordable edge device and reduced per-robot inference demand support battery-constrained robots and shared-GPU fleets.

### B.6 LIBERO-SAFETY × $\pi_{0.5}$

**Latency-aware evaluation.** We use the LIBERO-Safety-fine-tuned $\pi_{0.5}$ checkpoint and the L1 `obstacle_avoidance` tasks 2 and 4, selected by an earlier latency sweep over the suite's five tasks. Each evaluation condition uses the same 50 official initial states four times, with seed base 195 and a 600-step limit. The VLA generates ten actions; both paths execute five per decision. The first decision always invokes the VLA, followed by a fixed 50% local schedule. At 20 Hz, one control period is 50 ms. Batch-one measurements on RTX A6000 give mean request–response latencies of 18.0–18.2 ms for ULAP and 147.5 ms for $\pi_{0.5}$ including client preprocessing. These correspond to less than one and approximately three control periods, respectively. The evaluation uses three injected delay steps for the VLA and zero additional steps for local prediction. Inference is issued before the current interval ends; execution continues from the previous chunk during the delay, and the elapsed prefix of the arriving chunk is skipped.

**Predictor training.** Separate predictors use successful, zero-delay teacher rollouts, with 368 Moka-pot and 418 Mug episodes split 70/15/15% by episode for training, validation, and testing. Collection initial states are separated from the evaluation states by a configuration-space distance check. This variant uses current camera views only, frozen Theia-Tiny features, current proprioception, and the five previously executed actions. Each predictor has 1,862,151 trainable parameters excluding Theia and produces five actions in parallel. Training uses standardized Smooth-L1 loss, AdamW (learning rate $6 \times 10^{-4}$, weight decay 0.05), batch size 96, and up to 120 epochs with early stopping.

# C Ablation studies

## C.1 Input ablations on GR00T N1.7

We ablate the state MLP and executed-action history to assess ULAP's design (Figure 16). Removing the state MLP lowers success at all but the smallest local fraction. In most evaluated settings with moderate VLA-call reduction ($\leq 50\%$) and high success rates, action history yields higher success rates at comparable VLA-call reduction. This suggests history is useful under limited distribution drift; its benefit reverses under aggressive bypass, plausibly as local predictions shift histories away from the training distribution. Retraining and evaluation settings are specified in Appendix B.1.

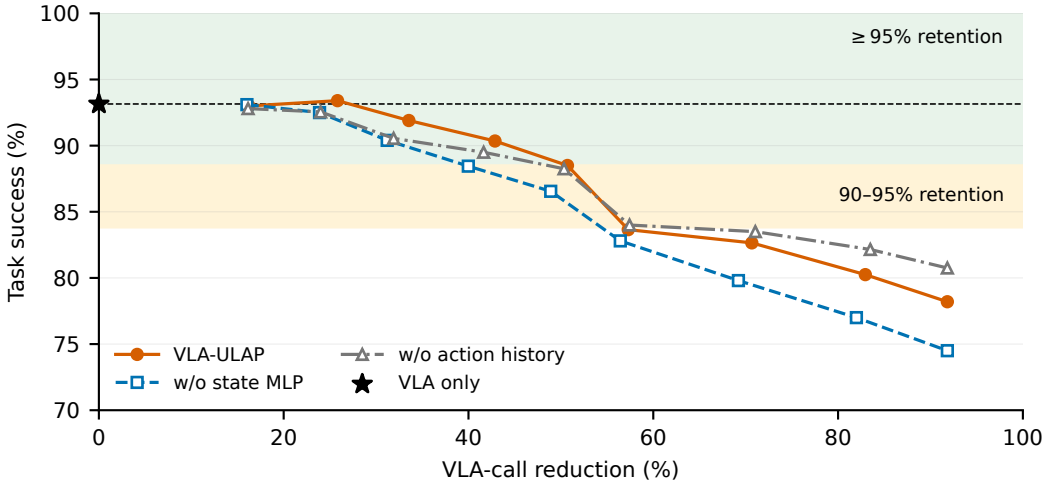



Figure 16: **Input ablations on GR00T N1.7.** Colors and retention bands follow Figure 3.

The sweep connects requested local fractions $\rho = 0.2, 0.3, \ldots, 1.0$. VLA-only succeeds in 1,863/2,000 episodes (93.15%) with 24,466 calls.

### C.2 VISION-BACKBONE SCALE ON GR00T N1.7

We replace frozen Theia-Tiny with frozen Theia-Small while retaining the full-input ULAP design. The backbone grows from 5.52M to 21.67M parameters; both use the same image resolution, patch size, and pooled token count. Only the input normalization and projection widen to accommodate Small's features; the remaining predictor architecture, training data, hyperparameters, and validation-based checkpoint selection are unchanged. Both variants use training seed 0 and the matched fixed-schedule evaluation in Appendix B.1.

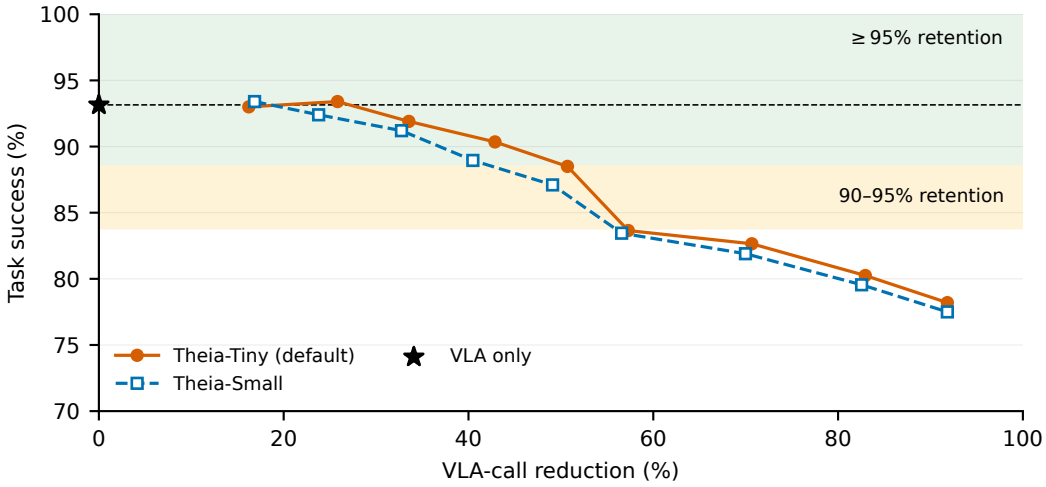



Figure 17: **Scaling the frozen vision backbone from Theia-Tiny to Theia-Small.** Each point covers 2,000 matched episodes across all 40 LIBERO tasks, with 16-step chunks and requested local fractions $\rho = 0.2, 0.3, \ldots, 1.0$. VLA-call reduction includes failures and uses the same VLA-only denominator as Figure 16. Axes, baseline, and retention bands follow that figure.

**Theia-Tiny is sufficient for this setting.** Scaling to Small yields no statistically significant success difference at any tested local fraction (two-sided exact McNemar tests, Holm correction across the nine fractions; all adjusted $p > 0.71$). Tiny achieves higher success with at least as much VLA-call reduction at eight of the nine settings (Figure 17). Thus, the larger backbone offers no demonstrated benefit here, supporting Theia-Tiny as the compact default for ULAP.

### C.3 SPATIAL VISUAL-TOKEN RESOLUTION ON GR00T N1.7

The backbone-scale ablation asks how much visual capacity ULAP needs; we next test how much spatial detail it should preserve. We pool the same frozen Theia-Tiny features to $1 \times 1$, $7 \times 7$, or the native $14 \times 14$ grid per camera, giving 2, 98, or 392 visual tokens across two views. Adding 16 action tokens, one proprioception token, and one task-conditioned aggregation token gives 20, 116, or 410 tokens in total. Only the visual positional-embedding size changes with the grid; all variants retain the same training data, seed 0, 120-epoch training recipe, and validation-based checkpoint selection. We evaluate 16-step chunks on the same 2,000 episodes per fixed-schedule setting as Appendix B.1.

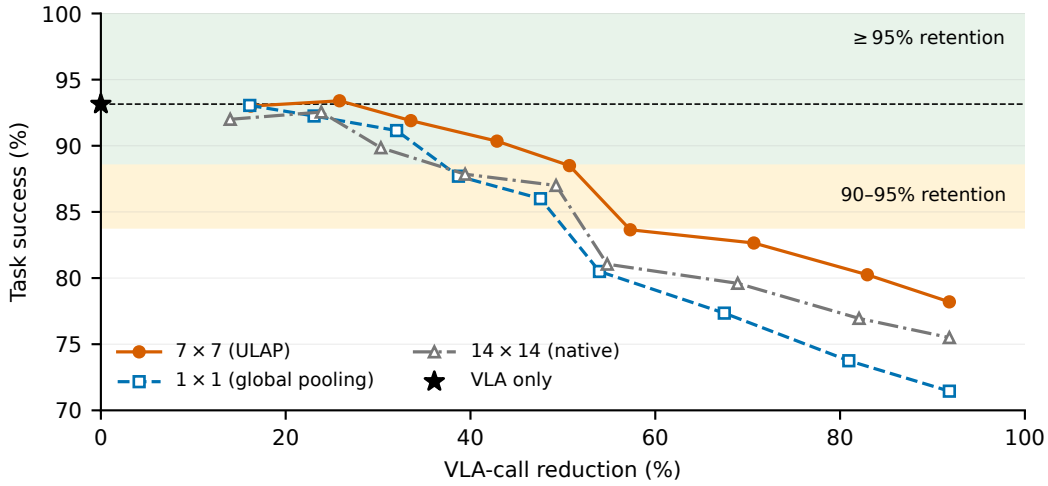



Figure 18: **Spatial pooling balances detail and compactness.** Each point covers 2,000 matched episodes across 40 LIBERO tasks. Lines connect the nine fixed schedules, $\rho = 0.2, 0.3, \ldots, 1.0$. VLA-call reduction compares total calls over all episodes with the matching VLA-only baseline. The star, dashed baseline and retention bands follow Figure 16.

**A compact spatial grid outperforms both extremes.** The $7 \times 7$ grid achieves at least as much call reduction and higher success than $1 \times 1$ at eight of nine settings, and than $14 \times 14$ at all nine settings (Figure 18). At the highest call reduction, 91.83%, success is 78.20% with $7 \times 7$, compared with 71.45% for global pooling and 75.50% for the native grid. Thus, collapsing each view to one token loses useful spatial information, while retaining four times as many visual tokens does not improve success under the same training recipe. Together with the backbone-scale result, this supports a small frozen encoder with a moderately pooled spatial representation, rather than either discarding spatial structure or increasing visual capacity indiscriminately.

**Why can the finer grid hurt?** On Spatial, Object, and Goal, $14 \times 14$ yields lower training RMSE but higher validation RMSE than $7 \times 7$, consistent with overfitting under the shared training recipe. Pooling $2 \times 2$ neighborhoods removes 46.5–48.5% of within-frame spatial variation in these suites. This suggests that moderate pooling regularizes the predictor by suppressing fine-grained, rollout-specific variation while retaining the spatial structure needed for action prediction.

# D EXPLORING ADAPTIVE CLOUD–LOCAL SCHEDULING

**Motivation.** Fixed scheduling distributes VLA calls evenly, regardless of the current scene or the local predictor's reliability. We investigate whether adaptive cloud–local scheduling can achieve higher task success rates than fixed scheduling by allocating VLA calls to decisions where they are more useful. To keep the additional routing computation small, we use information already available from observations and ULAP—visual features, predicted actions, or predictor embeddings—rather than train another routing model. We first compare three signals offline, then examine the strongest candidate, risk-aware scheduling, in closed-loop evaluation.

**Candidate signals.** We consider three signals for deciding when to prioritize a VLA call. *Image change* measures the change in Theia features over one executed chunk. For each camera, we average-pool the $7 \times 7$ token grid to $2 \times 2$, normalize each cell's feature, and average squared distances between matching current and previous cells. *Action change* is the RMS difference between ULAP's predicted chunk and the last executed chunk, aligned by relative step and standardized using training-set action statistics, including the gripper. *Embedding risk* measures how far ULAP's current representation lies from same-task training representations, as defined below. None requires an extra learned encoder. Image history is used only by the image-change scheduler, not as an additional predictor input.

**Offline comparison.** We replay held-out recorded trajectories for GR00T on LIBERO (40 tasks, 376 episodes, $H = 16$), VLA-JEPA on LIBERO (40 tasks, 581 episodes, $H = 7$), and Cosmos-Policy on RoboCasa (24 tasks, 376 episodes, $H = 16$). Task-specific CDFs are calibrated on the original validation episodes. All signals use percentile threshold 0.80, gap factor 1.20, and requested local fractions 0.2–0.9 with the same credit-based scheduler below. The gap factor limits the interval from an advanced VLA call to the next one to 1.20 times the nominal fixed-call interval, rounded up to a whole number of decisions. We measure action RMSE in training-standardized coordinates. Local decisions incur their prediction error against the recorded VLA chunk, whereas VLA decisions have zero error against that reference. For each episode, we compare each candidate with fixed scheduling up to the last boundary at which their cumulative VLA-call counts agree, including the initial VLA call. We compute RMSE from all retained chunks within each task, then average equally over tasks and the eight fractions.

Table 6: **Offline action-RMSE reduction relative to fixed scheduling (%).** Positive values indicate lower error on matched-call prefixes. Tasks and requested local fractions are equally weighted.

| Model / benchmark | Image change | Action change | Embedding risk |
|---|---|---|---|
| GR00T / LIBERO | -0.22 | +0.52 | **+1.39** |
| VLA-JEPA / LIBERO | +0.52 | +1.78 | **+2.13** |
| Cosmos-Policy / RoboCasa | +0.22 | +1.33 | **+1.61** |

Table 6 shows that embedding risk gives the largest RMSE reduction in all three pairs. These offline results support choosing risk-aware scheduling for the closed-loop comparison below. Offline replay holds observations and executed histories fixed, so we use it to screen routing signals rather than to infer task-success gains.

**Risk score.** We want to prioritize the VLA when the current state is unfamiliar to ULAP and its local prediction may be less reliable. We use distance from training representations as a proxy for this unfamiliarity. Let $\hat{z}_t$ be ULAP's normalized current embedding and $\mathcal{N}_{32}(\hat{z}_t)$ its 32 nearest same-task training embeddings. Their mean squared distance is

$$d_t = \frac{1}{32} \sum_{i \in \mathcal{N}_{32}(\hat{z}_t)} \max\big(0,\, 2 - 2\hat{z}_t^\top \hat{z}_i\big)\,, \qquad \hat{z} = \frac{z}{\|z\|_2}. \tag{1}$$

To make distances comparable within each task, a frozen empirical CDF $F_\tau$ converts $d_t$ to its percentile among that task's validation distances, $r_t = F_\tau(d_t)$. A score of at least 0.80 flags a distance in the highest 20% of the calibration distribution and requests an earlier VLA call. This is a risk score, not a failure probability.

**Credit-based scheduling.** The scheduler moves VLA calls toward high-risk decisions while tracking the requested call budget. Think of credit as an allowance that accumulates at each decision and is spent on VLA calls. A risky state can justify spending that allowance early, but the resulting debt must not postpone the next VLA call for too long. The initial chunk always uses the VLA and is outside this budget.

For requested local fraction $\rho = p/q$ in lowest terms, set $a = q - p$ and initial credit $B_0 = a - 1$. Each subsequent decision adds $a$ credits, $A_t = B_{t-1} + a$, and a VLA call spends $q$. Thus, credit accrues at a rate corresponding to the target VLA fraction $a/q = 1 - \rho$. If $A_t \geq q$, the scheduler makes a regular VLA call. Otherwise, it may borrow credit to call the VLA early when $E_t$ holds below. The gap factor $\gamma = 1.20$ caps the post-borrowing interval at $G$ decisions, or 1.20 times the nominal VLA-call interval, rounded up:

$$G = \left\lceil \frac{\gamma}{1-\rho} \right\rceil, \qquad E_t = (r_t \geq 0.80) \wedge (A_t > 0) \wedge (A_t - q \geq q - Ga). \tag{2}$$

$$V_t = \begin{cases} 1, & A_t \geq q \text{ or } E_t, \\ 0, & \text{otherwise}, \end{cases} \qquad B_t = A_t - qV_t. \tag{3}$$

The three conditions in $E_t$ require high risk, a positive credit balance before borrowing, and enough remaining credit to fund another VLA call within $G$ decisions. The last condition follows because the balance after the early call is $A_t - q$, and the next $G$ decisions add $Ga$ credits. Here $V_t = 1$ selects the VLA, otherwise ULAP; $B_t$ is the balance carried to the next decision. Disabling borrowing recovers fixed scheduling. ULAP computes the risk embedding at every post-initial boundary, and only the selected chunk is executed. Both schedulers use the same predictor and matched evaluation episodes, with 16-step chunks for GR00T and Cosmos-Policy and seven-step chunks for VLA-JEPA.

**Closed-loop results.** We next test whether the offline error reduction translates into higher task success when actions affect subsequent observations. Figure 19 compares eight requested local fractions per pair, with threshold 0.80 throughout. The horizontal axis is $100\big(1 - \sum_i C_i^{\text{method}} / \sum_i C_i^{\text{VLA}}\big)$, where $C_i$ counts VLA calls in evaluation episode $i$, including failures. Risk awareness improves GR00T success at seven fractions, but the pattern is not consistent. On VLA-JEPA it improves four, ties one, and lowers three; at fraction 0.9, success drops from 91.60% to 90.30% while call reduction also drops from 86.52% to 85.97%. On RoboCasa, success and call reduction both drop at fraction 0.5 (64.08% to 63.25% and 48.79% to 46.85%). We find no clear, consistent gain in the success–call-reduction trade-off despite the favorable offline ranking. We therefore use fixed scheduling as the standard configuration.

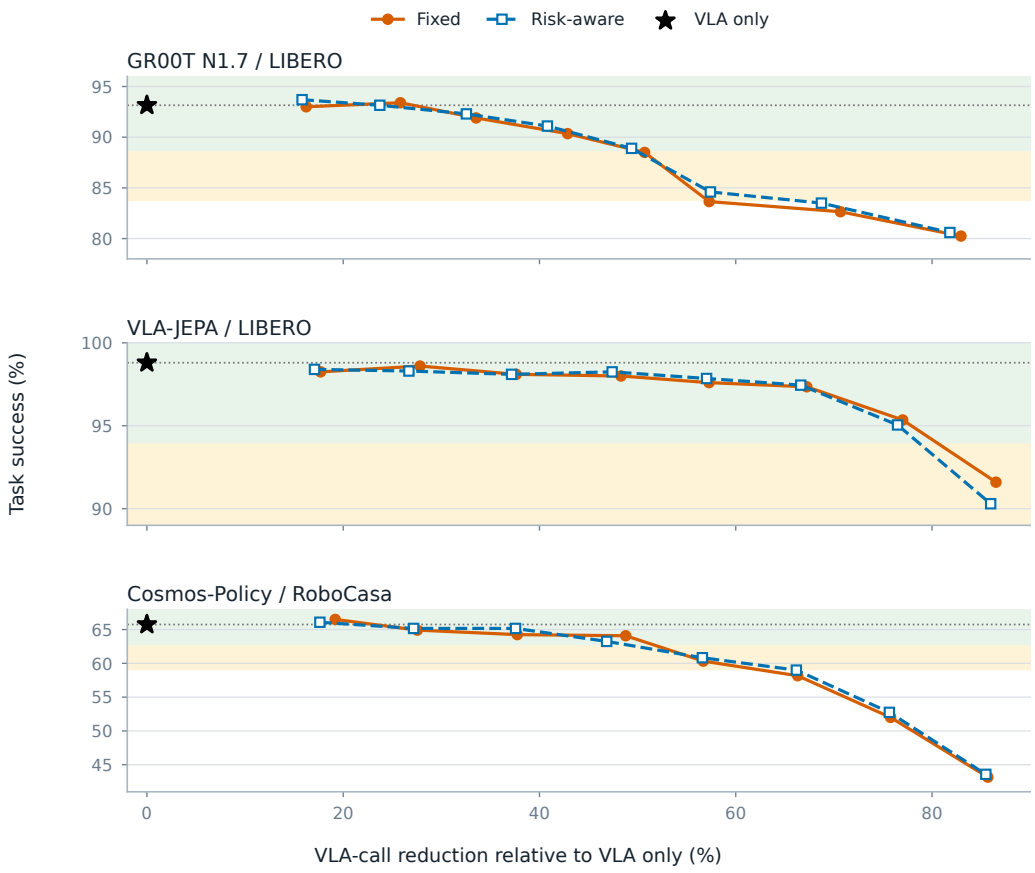



Figure 19: **Closed-loop evaluation of the selected risk-aware scheduler.** Success rate versus realized VLA-call reduction at requested local fractions 0.2–0.9. Black stars mark VLA-only baselines; green and amber bands indicate at least 95% and 90–95% of the baseline success rate, respectively. Each LIBERO point uses 2,000 episodes and each RoboCasa point 1,200; call totals include failures.

## E LOCAL PREDICTION VERSUS LONGER ACTION-CHUNK EXECUTION

Can simply executing longer chunks replace a local predictor? We test this with SmolVLA (Shukor et al., 2025), whose evaluated checkpoint generates 50 actions per call. Several checkpoints in our main experiments instead use short output horizons fixed during pretraining or benchmark fine-tuning, so extending their execution horizon would require changing the base policy. SmolVLA lets us isolate longer execution without retraining the VLA. We increase the executed prefix from $H = 20$ to 30, 40, or 50 steps, whereas VLA-ULAP keeps $H = 20$ and predicts fresh chunks at local decisions.

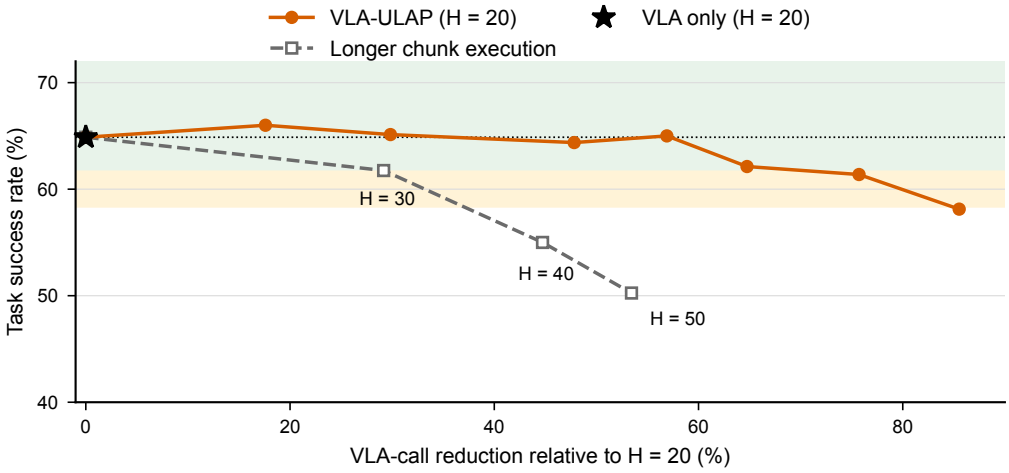



Figure 20: **Local prediction retains success better than extending open-loop execution.** SmolVLA on LIBERO; $H$ is the number of actions executed before a new decision. Call reduction is relative to SmolVLA-only at $H = 20$. Green and amber bands denote $\geq$95% and 90–95% retention of that baseline's success rate.

**Higher success at comparable or greater call reduction.** Figure 20 shows that longer open-loop execution rapidly loses success. Increasing $H$ from 20 to 50 reduces calls by 53.4%, but success falls from 64.88% (519/800) to 50.25% (402/800). VLA-ULAP instead achieves 65.00% success (520/800) at a larger 56.9% call reduction: a **14.75 pp advantage** over $H = 50$. The same pattern holds at a more moderate reduction: VLA-ULAP reaches 64.38% (515/800) at 47.8% reduction, versus 55.00% (440/800) at 44.7% reduction with $H = 40$. Even at 85.5% reduction, VLA-ULAP retains 58.13% success (465/800), exceeding $H = 50$ while using substantially fewer VLA calls. These results support reducing calls through fresh, state-conditioned predictions rather than simply postponing the next observation-conditioned decision.

**Training and evaluation settings.** Each condition uses the same 800 episodes: four LIBERO suites, ten tasks per suite, and official initial states 0–19, with environment seed 1000. The frozen `smolvla_libero` checkpoint (revision `6721902b`) always generates 50 actions. ULAP uses current camera views, proprioception, and the preceding 20 executed actions; one predictor per suite is trained on official demonstrations with a 70/15/15 episode split and selected by validation action RMSE. Training uses AdamW for 120 epochs, learning rate $6 \times 10^{-4}$, and seed 0. Fixed scheduling evaluates local fractions 0.2, $1/3$, 0.5, 0.6, 0.7, 0.8, and 0.9 after an initial VLA call. Success and call reduction use all evaluation episodes; reduction is $1 - C/C_{20}$, where $C$ and $C_{20}$ are total VLA calls in the evaluated condition and the $H = 20$ baseline, respectively.