HOW DEEP SHOULD A VLA THINK WHEN THINKING COSTS TIME? BUDGET-CONSTRAINED RL FOR EARLY EXIT

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) models are becoming capable generalist robot policies, but they are large, and their inference is slow on robotic hardware. Methods that make VLAs efficient are evaluated with the simulator paused while the model computes, as if the world waited. On a robot it does not: computation is consumed during control, and the value of an extra layer depends on how much control time it costs. We show in a controlled intervention that the same inference latency lowers task success by 3 to 40 points depending on how it reaches the robot, and we therefore ask how deep a VLA should think when thinking costs time. We cast adaptive depth as decision-making under a budget on the fraction of time spent inferring, and learn, with budget-constrained reinforcement learning, a lightweight policy that decides at each intermediate exit of a frozen VLA whether to act or to keep computing. Trained once per task suite in a simulator that keeps running while the model thinks, it runs 1.76× faster per inference than the full model on LIBERO-Plus, trading success for time as the budget sets, is on par with early-exit rules calibrated separately on every task at the same compute without any per-task calibration, and is ahead of an AVA-VLA-style exit gate on every task where their compute matches.

1 INTRODUCTION

Vision-language-action (VLA) models such as OpenVLA-OFT (Kim et al., 2025) and $\pi_{0.5}$ (Black et al., 2025a) map camera images and instructions to robot actions with a 3–7B-parameter backbone. Their generality comes at a price that robotic platforms feel acutely: every action requires a forward pass through billions of parameters. A growing body of work therefore makes VLAs cheaper – exiting the language model early when an intermediate layer already suffices (Yue et al., 2024; Zhang et al., 2026b; Song et al., 2026; Lei & Shan, 2026), pruning or caching visual tokens, or executing longer action chunks between inferences (Liang et al., 2026; Xu et al., 2026a;b; Shin et al., 2026).

These methods share an assumption about evaluation: the simulator pauses while the model computes, so a slower inference changes nothing but a clock. Computation is treated as a resource consumed *around* control. On a robot, it is consumed *during* control. While the policy thinks, the arm keeps executing an old command, holds still, or plays out a stale plan, and the world moves on. The value of an extra layer is therefore inseparable from the control time it costs, and a cheaper inference is worth more than the operations it saves.

We make this concrete in two observations. First, the cost of latency is real and depends on how it reaches the robot: in a controlled intervention from identical simulator states, the same three-step inference lowers success by 40 points when the robot repeats its last command, by 10 when it pauses, and by 3 under asynchronous execution. Second, most steps do not need the whole model: on held-out tasks, exiting after three quarters of the layers matches the full model. Together they pose the question of this paper: *how deep should a VLA think when thinking costs time?*

We answer it by treating adaptive depth as sequential decision-making under a time budget. A lightweight exit policy watches a frozen VLA's intermediate exits and decides, at each one, whether to act or to keep computing; it is trained by budget-constrained reinforcement learning in a simulator

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.

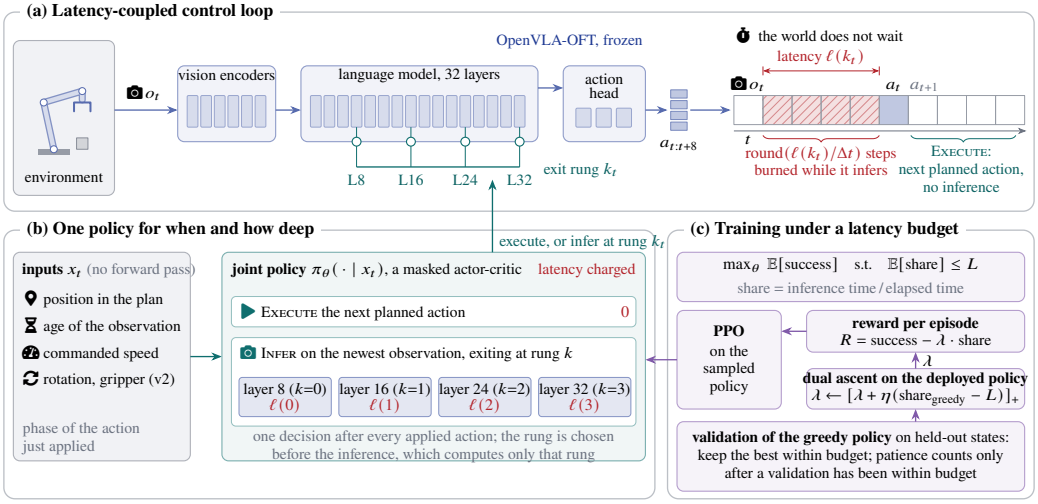

Figure 1: Method. (a) The frozen VLA (OpenVLA-OFT) runs in a simulator that keeps running while it infers: an inference exiting at rung k_t burns $\mathrm{round}(\ell(k_t)/\Delta t)$ control steps, and between inferences the robot executes the current plan. (b) The general scheduler decides when to infer and at which exit; we learn the exit decision – at each exit an inference reaches, act on it or continue – and fix the chunk at 8 actions, since a learned refresh decision reduced to it (Section 6.3). (c) It is trained by PPO under a constraint on the inference share, with the multiplier held to the share of the greedy policy that is deployed.

that keeps running while the model thinks, so that the budget binds the robot time the deployed policy actually spends. Two properties of this setting shape the method – when inference is charged in whole control steps some exits are never worth their cost, and a budget met by the stochastic training policy need not hold for the deterministic one deployed – and a consistency-based starting point and a budget enforced on the deployed policy address them. One policy serves a whole task suite, with no per-task calibration and no change to the backbone.

Contributions.

- We show that VLA computation is consumed inside the control loop, and that its cost depends on the execution model, which motivates evaluating efficient VLAs under a budget on closed-loop time (Sections 3 and 6.1).
- We learn when to stop computing with budget-constrained reinforcement learning on a frozen VLA, identify the two failure modes this setting creates, and remove them (Section 4).
- On LIBERO-Plus, one policy per task suite is on par with early-exit rules calibrated on every task at matched compute, is ahead of an AVA-VLA-style gate on every matched task, and runs 1.76× faster per inference than the full model; we show what reinforcement learning adds to its rule start (Section 6).

2 Related Work

Adaptive compute in VLAs. DeeR-VLA exits a multimodal LLM when consecutive exits agree, with thresholds set to meet a budget on average compute, peak compute or memory (Yue et al., 2024); CEED-VLA exits Jacobi decoding iterations (Song et al., 2026); A1 truncates the model and ties flow-matching steps to the exit (Zhang et al., 2026b). AVA-VLA learns a latent reasoning loop by RL and exits it with a gate calibrated after RL (Lei & Shan, 2026); its vision encoder and language model always run in full. ElegantVLA trains one Maskable-PPO scheduler over coarse levels of vision encoder, language model and action head with a FLOPs penalty (Li et al., 2026a);

EQRL selects denoising budget and chunk length by RL (Wang et al., 2026a). To our knowledge, none of these evaluations charges compute as time the world keeps running.

When to act again. Action chunking trades responsiveness for smoothness. Chunk length is chosen from action entropy (Liang et al., 2026), cross-attention entropy (Xu et al., 2026a) or denoising variance (Feng et al., 2026), learned as continue-or-replan (Xu et al., 2026b), as an action and its duration (Sukhija et al., 2026), or by a Q-value per chunk length (Shin et al., 2026); VLA-Corrector replans when a latent monitor detects deviation (Pan et al., 2026).

Acting under latency. Real-time chunking executes the previous plan while the next is computed and inpaints the overlap (Black et al., 2025b); delayed MDPs augment the state with pending actions (Katsikopoulos & Engelbrecht, 2003; Ramstedt & Pal, 2019; Xiao et al., 2020). RL fine-tuning of π_0 and $\pi_{0.5}$ at a fixed latency (Zhu et al., 2026) and a fast RL edit policy on top of a slow VLA (Dong et al., 2026) adapt the *policy* to latency; we adapt the *compute* to the state, with the policy frozen.

3 Problem: compute that costs time

Latency coupling. The frozen policy exposes exits (*rungs*) $k \in \{0, \dots, K-1\}$; an inference that exits at rung k takes $\ell(k)$ seconds. After each applied action a scheduler decides what the next control step does: EXEC the next action of the current plan, REPLAN (infer again on the cached visual encoding with fresh proprioception), or ENCODE (encode the newest observation and infer), and, for an inference, the rung. An inference of latency ℓ_t burns $b_t = \mathrm{round}(\ell_t/\Delta t)$ control steps of the environment, during which the robot executes the *filler* of an execution model: *repeat* holds the previous delta command, *pause* commands zero motion while the burned steps count toward the step limit (published evaluations also hold the robot still but charge no steps), and under *asynchronous* execution the previous plan keeps running and the new plan skips the actions it already covered (Black et al., 2025b). The observation the next inference sees has aged by every applied and burned step. Latency is *charged* from a measured table, not timed, so results do not depend on the host.

Inference share and objective. For an episode ξ, the inference share is the fraction of elapsed environment time spent inferring,

$$s(\xi) = \frac{\sum_{t \in \mathcal{F}(\xi)} \hat{\ell}_t}{N_{\mathrm{env}}(\xi)\,\Delta t} \in [0, 1], \tag{1}$$

with $\mathcal{F}$ the inference steps, $\hat{\ell}_t$ the charged latency and N_{env} the applied plus burned steps. With success $Y(\xi) \in \{0, 1\}$ and a budget L the scheduler solves

$$\max_{\theta}\ \mathbb{E}_{\xi\sim\pi_\theta}[Y]\ \text{ s.t. }\ \mathbb{E}_{\xi\sim\pi_\theta}[s] \le L, \qquad \min_{\lambda\ge 0}\max_{\theta}\ \mathbb{E}[Y - \lambda s] + \lambda L. \tag{2}$$

The share, not seconds per episode, is the matched quantity: failed episodes run to the step cap and accumulate inference, so time per episode would credit success with speed. Section A gives the charge on nested and non-nested ladders, the rounding convention and the share's floor.

4 Method: a per-suite exit policy under a compute budget

The lever is the exit layer of each inference; the method sets it by solving Equation (2) once per task suite. Two properties of the problem shape it: compute is charged in whole control steps, and the policy that is deployed is not the one the constraint is applied to.

Action space. Each inference runs the frozen backbone's nested forward pass and reaches the exits in order. At each exit k the exit policy π_θ chooses to act on that exit's action or to continue to the next; continuing past the last exit is masked, so the last exit is full depth. A walk that stops at exit k is charged exit k's latency; charging the 4 ms readout of each earlier exit changes no charged step (Section E). The refresh schedule is fixed to the backbone's 8-action chunk (Section 6); the general form, a policy that also decides when to infer again, is Section B's joint action space.

Inputs. At an exit the policy reads what that exit produced: its position in the ladder, the cumulative cost so far, the change of its action from the previous exit's relative to the action's norm, whether there was a previous exit, and (the *v2* set) that action's translation, rotation and gripper command – the phase of the task. Optionally it also reads five scalars of the episode's memory (Section B). Nothing needs an extra forward pass.

Training. Episodes run in the latency-coupled simulator, so a decision's cost is paid both as the penalty $\lambda\hat{\ell}_t/(N_{\text{env}}\Delta t)$ and as the staleness its burned steps cause. The terminal reward is the success, and Equation (2) is optimised (Algorithm 1) by PPO on the primal and dual ascent on λ (Tessler et al., 2019). Training runs to a ceiling of rounds, validates the greedy policy on held-out initial states every five rounds, keeps the best validation (within budget first, then success), and stops by patience. The backbone is frozen; the policy is a two-layer MLP of width 32.

Holding the deployed policy to the budget. PPO optimises the *sampled* policy, while validation keeps and evaluation deploys its *argmax*. Under a share constraint the two can differ sharply: where the stop probability sits near one half, the sampled policy meets the budget as a mixture while its argmax either stops at the cheapest rung everywhere or never stops early. With λ updated on the sampled share, nothing pushes back. We therefore (i) update λ on the greedy policy's share as last measured on validation (the sampled share before the first validation), and (ii) count a validation toward patience only once some validation has been within budget, so training is not stopped while λ is still pulling the deployed share down. This is an empirical mitigation, not a constrained-RL guarantee; Section 6.5 shows the failure and how far it is repaired.

Exits where the cost steps. The simulator charges an inference in whole control steps, $\text{round}(\ell/\Delta t)$, so the 32 layers of OpenVLA-OFT fall into three or four cost levels, and within a level a deeper exit costs nothing more. An exit that is not the deepest of its level is dominated. On our ladder (exits after layers 8, 16, 24 and 32, charged 2, 2, 3 and 4 control steps) layer 8 costs as much as layer 16 yet solves almost nothing; the rule start below never stops at the first exit, which removes that option by construction (Section B).

Starting from a rule, and keeping it unless beaten. Trained from random weights, the exit head learns from a terminal success spread over some hundred decisions per episode, and on OpenVLA-OFT it does not find even the simplest good rule (Section 6). Its own inputs already contain what a DeeR-style rule thresholds – the change between consecutive exits' actions, relative to the action – so we set the network's first-layer unit to $\tanh(3(t-c)/t - 9(1-h))$ on that change c and the indicator h of a previous exit, relay it through the second layer, and let the policy logits read it alone: the untrained head stops at the first exit whose action has settled, never at the first exit, and explores the other option 5% of the time. The other units start as drawn, so PPO has the whole network to improve on the rule. The head is validated before its first update, so the kept head is the rule unless training beats it on held-out validation states.

5 Experimental setup

Backbone and benchmark. OpenVLA-OFT (Kim et al., 2025) fine-tuned on LIBERO-Spatial, frozen, with language-model exits after layers 8, 16, 24 and 32 of 32 (rungs 0-3), trained by distillation to the full-depth action on 18 tasks and verified to reproduce the full model exactly at the last rung. Tasks come from LIBERO-Plus at difficulty 5 (Fei et al., 2026; Liu et al., 2023). **One exit policy per suite** is trained on 15 development tasks (initial states 40-49, validated on 35-39) and evaluated on six held-out tasks from six of LIBERO-Plus's seven perturbation families, 35 episodes each (initial states 0-34; a layout task records one initial state, so its episodes differ by seed), under the pause, at a budget share of 0.267 (the midpoint of the constant exits' shares). These tasks share base scenes with development tasks and informed the rule start's threshold grid, so we call them development-held-out; a fresh set of 25 tasks (7 robot-initialisation, 4 camera, 4 noise, 10 layout), fixed before any of them was run and used for no design choice, confirms the chosen variant once. Charged latencies are 32, 69, 111 and 150 ms per rung plus a 45 ms per-inference overhead.

Comparators, all at matched share on the same ladder. *Constant exits* E0-E3 (a fixed rung, chunk 8). The *hull* H0: the upper convex hull of constant configurations – rungs, or rungs $\times$ chunk

Table 1: What a three-step (160 ms) inference costs in closed loop: $\pi_{0.5}$, fixed five-action chunks, seven LIBERO-Plus difficulty-5 tasks, 50 episodes per cell from the same recorded states. Mean per-task difference in success, exact sign-flip test over tasks, and the episodes each contrast lost and gained. Full cells in Section E.

effect	contrast	Δ success	p	lost/gained
noise floor	`ref` again − `ref`	0.000	1.00	0/0
latency, repeat filler	repeat − `ref`	−0.400	0.016	142/2
latency, pause	pause − `ref`	−0.100	0.25	36/1
latency, asynchronous	async − `ref`	−0.029	0.25	12/2
stale plan acted on as fresh	stale − `ref`	−0.231	0.016	86/5
aligning the plan in time	async − stale	+0.203	0.016	11/82

lengths $\{1, 2, 4, 8\}$ for the joint policy – fitted on validation states and realised per task at the learned policy's own share by randomising per episode between two vertices; it is the best constant choice a deployer could make with hindsight, an envelope over the evaluated constant family, not over adaptive rules. *DeeR-VLA/A1-style exit* B4 (Yue et al., 2024; Zhang et al., 2026b) (the DeeR-style threshold below): exit when consecutive rungs' actions agree, threshold fitted at the learned policy's share. *AVA-VLA-style gate* G4 (Lei & Shan, 2026): their gate module and their paper pipeline's stage-4 label (a step is sufficient when three more steps gain less than 0.05 of value), with each rung's value *measured* by running the episode at that rung, threshold fitted at the learned policy's share (Section C). AVA-VLA's published numbers are quoted, not reproduced: no weights are released.

Statistics. Tasks are the unit. Each contrast is the mean per-task difference in success with an exact two-sided sign-flip test over tasks; per-arm success is reported with a 95% t interval over tasks. With six tasks the smallest two-sided sign-flip p is 0.031 (five: 0.063); after matched-share exclusions (3–5 tasks) no held-out contrast can reach $p < 0.05$, so held-out contrasts are descriptive and the confirmation tasks are the test. *On par* means a difference that is not significant; it is not a test of equivalence. A matched-share contrast (every contrast except those against full depth) includes a task only where both of its sides realise a share within 0.02 of the hull's realised share at our policy's target; other tasks are excluded, and each table says how many.

6 Results

Each subsection answers one part of the question of Section 1: why charge compute as time (Section 6.1), whether a budgeted exit keeps success at less time (Section 6.2), what RL adds to its rule start (Section 6.3), whether the answer holds on fresh tasks (Section 6.4), and why the budget must bind the deployed policy (Section 6.5).

6.1 Why compute must be charged as closed-loop time

The first question is whether inference time matters at all once the simulator keeps running, and in what form. From identical simulator states, one frozen policy ($\pi_{0.5}$) runs in cells that change exactly one thing (Table 1), a controlled intervention within this simulator. How much a slow inference costs depends on how its time reaches the world: 40 points under a repeated command, on every task; 10 under the pause, every loss a timeout on the two tasks with the least time to spare; 3 under asynchronous execution. Serving a three-step-old view while acting on every step costs 23 points, and asynchronous execution, which serves a view of the same age but skips the actions the previous plan already covered, recovers 20 of them. What costs success is acting on a stale plan as if it were fresh. So compute has a price in closed loop that paused evaluations cannot see, and that price is set by the execution model. The remaining experiments use the pause, the conservative choice: compute costs time and step budget, but never a wrong command, so any gain from exiting early is a gain in time, not in avoided staleness.

Table 2: Our exit policy over three training seeds on the development-held-out tasks: per-task success difference averaged over seeds, then over tasks (the task is the unit of the sign-flip test); below, each arm's mean and std over seeds (bold: best at or below the budget).

Ours	Δ success	per seed (0/1/2)	**Tasks**	up/tie/down	p	share gap (mean / max)
vs full depth (E3)	-0.044	-0.067 / -0.029 / -0.038	6	2/0/4	0.53	-0.099 / -0.065
vs best constant mixture (H0)	+0.127	+0.219 / +0.038 / +0.124	3	2/0/1	0.50	-0.001 / +0.010
vs DeeR-style threshold (B4)	+0.019	-0.029 / +0.057 / +0.029	1	1/0/0	1.00	+0.004 / +0.017

Arm	success, mean $\pm$ std over 3 seeds	ms/inf
Full depth	0.784 ± 0.003	199
Layer 24	0.786 ± 0.000	150
Hull	0.646 ± 0.064	114
DeeR-VLA/A1-style	0.733 ± 0.010	122
Ours	$\mathbf{0.740 \pm 0.023}$	124

Table 3: Success per LIBERO-Plus perturbation family on the five held-out tasks on which every baseline ran, including the AVA-VLA-style gate (its stage did not finish on the sixth, sensor-noise task; all six tasks over three seeds are in Table 2 and per family in Table 12); seed 0. Adaptive rules are fitted at our policy's share on each task. Above the light rule: more compute per inference than our budget; bold: the best success at or below it.

Method	Background	Camera	Language	Layout	Light	**Average** $\uparrow$	ms/inf $\downarrow$
Full depth (layer 32)	0.86	0.54	0.97	1.00	0.97	0.869	199
Exit at layer 24	0.94	0.66	1.00	1.00	0.97	0.914	149
Exit at layer 16	**1.00**	0.06	0.26	0.37	0.77	0.491	100
Hull (best constant mixture)	**1.00**	0.37	0.57	0.60	0.77	0.663	110
DeeR-VLA/A1-style exit	**1.00**	**0.60**	0.60	**0.97**	**0.91**	0.817	118
AVA-VLA-style gate	0.66	0.37	0.40	0.89	0.66	0.594	115
Ours	**1.00**	0.57	**0.66**	**0.97**	**0.91**	**0.823**	114

6.2 Does a budgeted exit keep success at less time per inference?

Main result. Our policy attains the highest success of all methods that stay within the budget. Averaged over three training seeds on the six held-out tasks (Table 2), it reaches 0.740, against 0.733 for the DeeR-VLA/A1-style threshold and 0.646 for the best constant mixture; only full depth and the layer-24 exit, which spend more compute per inference, score higher, and our policy needs $1.76\times$ less time per inference than full depth.

Matched compute. The advantage holds when every rule spends the same compute. With each adaptive rule fitted on every task at the share our policy realises there (Table 3, on the five tasks where every baseline ran; all six in Table 12 and Section E), our policy is at or above each of them on average: the DeeR-VLA/A1-style threshold by $+0.010$, the best constant mixture by $+0.131$ and the AVA-VLA-style gate by $+0.221$, ahead of the gate on every task where their compute matches. The constant exits show why adaptivity pays: after three quarters of the layers the policy already matches full depth on these tasks, at 25% less time, while stopping after half of them does not.

Efficiency. Following AVA-VLA, Table 4 reports the mean exit layer, the layers skipped and the speed-up in time per inference, together with the share of episodes that succeed within a robot-time deadline (Figure 2b): within a 6 s deadline our policy completes more episodes than full depth. Charged times are in whole 50 ms control steps (Section E).

No per-task calibration, no backbone training. To be fitted at our share, the DeeR-VLA/A1-style threshold and the AVA-VLA-style gate each spend 144 validation episodes on every task, and the hull 80; our policy is trained once per suite and runs on a new task as is, training 0.22M adapter and 1.4K head parameters on a frozen 7B backbone. Published efficient-VLA results are collected in Section C.

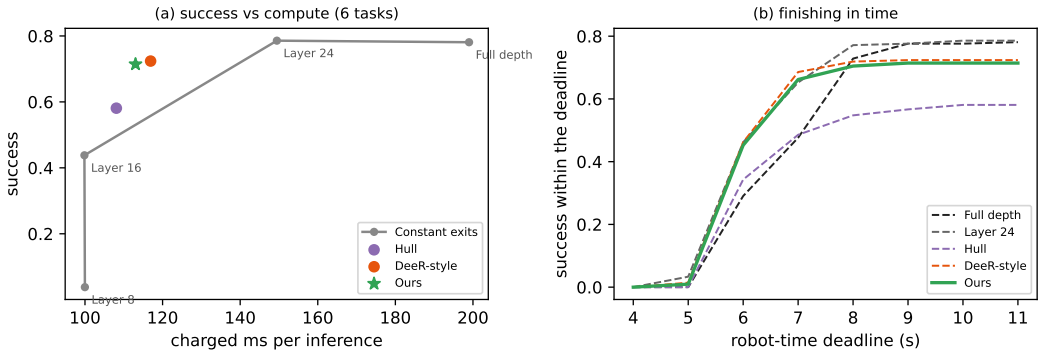

Figure 2: Held-out tasks, seed 0. (a) Success against the charged time per inference: constant exits (grey curve) and the adaptive rules, each at our policy's share on each task. (b) The share of episodes that succeed within a robot-time deadline, which credits a policy for finishing sooner; nothing is conditioned on success. Within a 6 s deadline our policy finishes more episodes than full depth, and about as many as the layer-24 exit and the DeeR-style threshold; at longer deadlines the layer-24 exit finishes more, and given unlimited time full depth does too, at 1.32× (layer 24) and 1.76× (full depth) our charged time per inference – the trade-off the budget sets, and which the threshold t moves (Section 6).

Table 4: Efficiency on the development-held-out tasks (seed 0), averaged over tasks: mean exit layer of 32, layers skipped, charged time per inference and its speed-up over full depth, inference and robot time per episode, and success within 6 and 8 seconds of robot time. Above the light rule: more compute per inference than our budget; bold: the best success at or below it. DeeR-VLA/A1- and AVA-VLA-style: their exit rules, re-implemented on our exits (Section C).

Method	mean exit layer	layers skipped	ms/inf (speed-up)	infer. s/ep	robot s/ep	success within ≤6 s	success within ≤8 s	success
Full depth	32.0	0%	199 (1.00×)	2.61	7.64	0.290	0.729	0.781
Layer 24	24.0	25%	149 (1.33×)	1.97	7.11	0.462	0.771	0.786
Layer 16	16.0	50%	100 (1.99×)	1.78	8.83	0.295	0.405	0.438
Hull	17.3	46%	108 (1.84×)	1.77	8.17	0.343	0.548	0.581
DeeR-VLA/A1-style	18.8	41%	117 (1.70×)	1.69	7.27	**0.462**	**0.719**	**0.724**
Ours	18.2	43%	113 (1.76×)	1.67	7.36	0.452	0.705	0.714

6.3 What does RL learn beyond the consistency rule?

The rule start is already strong: a single consistency threshold for the whole suite, with no per-task calibration, is above the DeeR-VLA/A1-style threshold calibrated on every task at matched compute (+0.034, Table 5).

Our policy starts as a consistency rule and is kept only if RL beats that rule on validation, so the natural question is what training changed. **In success, nothing significant**: against its own start, the rule with RL off on the same tasks and episodes (Table 5), the policy after RL is lower, not significantly, at a slightly lower share (last sentence of the result in Section 6.2). **In behaviour, one thing** (Table 10): re-applying the start rule offline to the rung actions the RL policy logged, at the same observations, the deployed (greedy) RL policy takes the rule's exit on 78% of inferences and stops *earlier* than the rule on the other 22%; it never stops deeper, and, like the rule, never at layer 8. At layers 16 and 24 alike it stops at every exit where the rule would (relative action change $c < 0.20$) and at none with $c \geq 0.28$; the fraction that stops falls from 0.92–0.96 to 0.39–0.54 across $c \in [0.24, 0.28)$. RL has therefore moved the rule's consistency threshold from 0.20 to between 0.26 and 0.28, trading a little depth for time. Outside $c \in [0.24, 0.28)$ its decision is a function of c alone, so the three action-phase inputs it also reads (this head has no episode memory) can matter only inside it. So, on this ladder, the consistency of successive exits is a signal RL does not improve on,

Table 5: Development-held-out tasks, seed 0: the rule start alone (RL off). Arms: mean over every task at its own share; contrasts: only tasks where both sides lie within 0.02 of the hull's realised share. Above the light rule: more compute per inference than our budget; bold: the best success at or below it. DeeR-VLA/A1- and AVA-VLA-style: their exit rules, re-implemented on our exits (Section C).

Method	**Arm**	**Success** ↑	**Tasks**	**Share** ↓	**ms/inf** ↓	**s/ep** ↓
Full depth (layer 32)	E3	0.800 ± 0.278	6	0.341	199	2.60
Exit at layer 24	E2	0.795 ± 0.340	6	0.278	150	1.96
Exit at layer 16	E1	0.433 ± 0.376	6	0.203	100	1.78
Hull (best constant mixture)	H0	0.648 ± 0.327	6	0.234	117	1.82
DeeR-VLA/A1-style exit	B4	0.719 ± 0.308	6	0.241	122	1.76
Ours	E4	$\mathbf{0.748 \pm 0.301}$	6	0.243	124	1.76
Contrast	Δ success	**Tasks**	**up/tie/down**	p	**ms/inf**	**s/ep**
E4 − B4	+0.034	5 (1 excl.)	3/2/0	0.25	123 / 121	1.83 / 1.83
E4 − H0	+0.074	5 (1 excl.)	3/0/2	0.44	123 / 120	1.83 / 1.88
E4 − E3	−0.052	6	1/2/3	0.50	124 / 199	1.76 / 2.60

and the value of RL here is not a better decision rule but a procedure that takes a budget and returns a deployed policy aimed at it without a per-task search.

The threshold is also a speed knob. Because what RL moves is the threshold, the rule start's t sets the compute without retraining: a larger t accepts a larger change as consistent and stops earlier (on development validation, share falls monotonically from 0.246 to 0.210 as t rises from 0.15 to 0.30). On the held-out tasks, Table 13 places each threshold and our policy after RL side by side: a lower threshold buys back full-depth success at more time per inference.

Why RL must start from a rule. Trained from random weights on the same protocol, the exit head stops after layer 8 on 40-60% of its inferences, mostly late in the episode. Layer 8 alone solves almost nothing (under 5% of episodes on the held-out tasks), yet it is charged the same two control steps as layer 16, so every such stop buys layer-8 accuracy at layer-16 cost. A terminal success spread over a hundred decisions does not reveal this; the rule start never stops at the first exit. **Why only the exit is learned.** A refresh head trained per task reduces to the constant 8-action chunk (identical success and share): under the pause an earlier refresh buys a fresher view only at the price of share, so we fix the chunk and learn the exit; where the world keeps moving (Table 1) the refresh decision has room to act.

6.4 Does the answer hold on tasks never used in design?

Confirmation. The matched-compute result carries over to tasks never used in design. Because the held-out tasks informed the rule start, the chosen policy was evaluated once, unchanged, on 25 fresh LIBERO-Plus tasks fixed before any run on them (Table 6; task accounting and statistics in Section E). Without any per-task calibration it stays on par with the consistency threshold calibrated on each task (-0.010), remains ahead of the AVA-VLA-style gate on every task where their compute matches, and needs $1.67\times$ less time per inference than full depth.

Where depth still matters. Full depth is more accurate on these harder tasks, by 0.29 on average. The constant exit after layer 24, which matched full depth on the held-out tasks, already trails it here, so these tasks need their last layers, most of all under sensor noise (Table 14). Since every rule at or below our share trails full depth, the gap reflects the budget rather than the exit rule, and our policy even under-spends that budget (share about 0.23 against 0.267). For such tasks a deployer can set a larger budget or a lower threshold, which trades compute for success on the held-out tasks (Table 13). The best constant mixture fitted per task is above ours here, not significantly.

Table 6: Fresh confirmation tasks, never used in design: the chosen variant's seed-0 head. Arms: mean over every task at its own share; contrasts: only tasks where both sides lie within 0.02 of the hull's realised share. Robot-initialisation tasks score 0 at every depth, so they tie; †only tasks where the rule could be fitted at our share; the gate's comparison, on the tasks that ran it, is in the appendix. Above the light rule: more compute per inference than our budget; bold: the best success at or below it. DeeR-VLA/A1- and AVA-VLA-style: their exit rules, re-implemented on our exits (Section C).

Method	**Arm**	**Success** ↑	**Tasks**	**Share** ↓	**ms/inf** ↓	**s/ep** ↓
Full depth (layer 32)	E3	0.531 ± 0.215	17	0.343	199	3.24
Exit at layer 24	E2	0.346 ± 0.204	17	0.275	150	2.64
Exit at layer 16	E1	0.195 ± 0.153	17	0.201	100	2.07
Hull (best constant mixture)	H0	$\mathbf{0.319 \pm 0.158}$	17	0.222	111	2.15
DeeR-VLA/A1-style exit	B4	0.250 ± 0.187	16†	0.242	124	2.46
Ours	E4	0.245 ± 0.178	17	0.234	119	2.39
Contrast	Δ success	**Tasks**	**up/tie/down**	p	**ms/inf**	**s/ep**
E4 − B4	−0.010	11 (6 excl.)	4/5/2	0.94	117 / 122	2.33 / 2.40
E4 − H0	−0.100	12 (5 excl.)	4/1/7	0.30	117 / 116	2.34 / 2.19
E4 − E3	−0.286	17	0/6/11	¡0.01	119 / 199	2.39 / 3.24

6.5 Why the budget must hold at the deployed policy

With a stop probability that genuinely varies, the mismatch of Section 4 appears (Table 17 and Figure 4): with λ on the sampled share the sampled policy met the budget while its argmax collapsed (task 2146: 0.086 success) or overspent it (632, 1056). Holding λ to the deployed share removed the collapse (2146: 0.714, within budget); on 632 and 1056 the deployed share still ended over budget, which is why a validation counts toward patience only once some validation has been within budget.

7 Limitations

Our evaluation is in simulation, on one task suite (LIBERO-Spatial under LIBERO-Plus perturbations) and one backbone for the exit policy, with a single training seed on the fresh tasks; extending it to more suites, backbones and real robots is natural future work, and the latency intervention, measured in simulation, may take different values on other platforms. Latency is modelled as a per-exit table charged in whole control steps. In our experiments, reinforcement learning mainly moves the rule's operating point towards lower compute rather than raising success (Section 6.3), so a simpler threshold search at the budget is a reasonable alternative in settings like ours; richer inputs or longer training may let a learned policy depart further from its start. On the fresh tasks, a larger budget would recover more of full depth's success.

8 Conclusion

In closed loop, thinking costs a VLA time: 3 to 40 points of success, depending on how the time reaches the robot. Under a budget on the inference share, the answer on our ladder is to exit when successive exits agree, at a threshold the budget sets; budget-constrained RL, made to work by a rule start and a dual variable held to the deployed policy, keeps that rule and re-tunes its threshold without raising success, and one policy per suite, with no per-task calibration, is on par with a threshold calibrated per task. On fresh tasks never used in design it trails full-depth success at $1.67\times$ less time per inference. There the budget, more than the exit rule, is what must change. Adaptive compute for robots should be evaluated with the control time it consumes.

AI use statement

We used generative AI tools to organise experiment results into tables, to produce plots, and to polish the language of the paper. We also used an AI coding assistant to help write and test the experiment code, to monitor runs on our cluster, and to check references against their sources. Every number

in the paper is generated by scripts from logged run artifacts; the authors reviewed all AI-assisted work and take responsibility for the content.

Reproducibility statement

All runs are launched from a pinned commit that each run records; the environments are locked (pixi manifests and lock files for each stack); latency is charged from measured tables, so results do not depend on the GPU. Tables and figures are generated from the pooled artifacts, with each table's source file and hash recorded in its LaTeX source. Code, run logs and the per-task artifacts will be released; Sections A to D give the details needed to reproduce each experiment.

References

Eitan Altman. *Constrained Markov Decision Processes*. Stochastic Modeling Series. Chapman and Hall/CRC, 1999.

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 (CoRL)*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025a.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 37596–37620, 2025b. doi: 10.52202/085713-1122.

Perry Dong, Kuo-Han Hung, Dorsa Sadigh, and Chelsea Finn. Reinforcement learning for real-time vision-language-action policies. *arXiv preprint arXiv:2609.18207*, 2026.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, Jinlan Fu, Jingjing Gong, and Xipeng Qiu. LIBERO-Plus: A progressive robustness benchmark for visual-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 38574–38583, 2026.

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.

Titong Jiang, Xuefeng Jiang, Yuan Ma, Xin Wen, Bailin Li, Kun Zhan, Peng Jia, Yahui Liu, Sheng Sun, and Xianpeng Lang. The better you learn, the smarter you prune: Towards efficient vision-language-action models via differentiable token pruning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

Konstantinos V. Katsikopoulos and Sascha E. Engelbrecht. Markov decision processes with delays and asynchronous cost collection. *IEEE Transactions on Automatic Control*, 48(4):568–574, 2003. doi: 10.1109/TAC.2003.809799.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025. doi: 10.15607/RSS.2025.XXI.017. arXiv:2502.19645.

Dianqiao Lei and Lianlei Shan. Think less, act early: Reinforced latent reasoning with early exit in vision-language-action models. In *International Conference on Machine Learning (ICML)*, volume 306 of *Proceedings of Machine Learning Research*, 2026. arXiv:2606.15099.

Wei Li, Renshan Zhang, Rui Shao, Jie He, and Liqiang Nie. CogVLA: Cognition-aligned vision-language-action models via instruction-driven routing & sparsification. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 137646–137675, 2025. doi: 10.52202/085713-4594.

Ye Li, Huanan Liu, Kangye Ji, Yuan Meng, Jiajun Fan, Yuansong Wang, Shiyu Qin, Chenglei Wu, Shu-Tao Xia, and Zhi Wang. ElegantVLA: Learning when to think for efficient vision-language-action models. *arXiv preprint arXiv:2605.29438*, 2026a.

Ye Li, Yuan Meng, Zewen Sun, Kangye Ji, Chen Tang, Jiajun Fan, Xinzhu Ma, Shu-Tao Xia, Zhi Wang, and Wenwu Zhu. SP-VLA: A joint model scheduling and token pruning approach for VLA model acceleration. In *International Conference on Learning Representations (ICLR)*, 2026b.

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.

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 (NeurIPS), Datasets and Benchmarks Track*, volume 36, pp. 44776–44791, 2023.

Ziyan Liu, Yeqiu Chen, Hongyi Cai, Tao Lin, 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*, 2025. Earlier versions titled VLA-Pruner: Temporal-Aware Dual-Level Visual Token Pruning for Efficient Vision-Language-Action Inference.

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.

Seongmin Park, Hyungmin Kim, Sangwoo Kim, Wonseok Jeon, Juyoung Yang, Byeongwook Jeon, Yoonseon Oh, and Jungwook Choi. Saliency-aware quantized imitation learning for efficient robotic control. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 13140–13150, 2025.

Simon Ramstedt and Christopher Pal. Real-time reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Boyang Shen, Kaixiang Yang, Hao Wang, Qiuyu Yu, Qiang Xie, Qiang Li, and Zhiwei Wang. LoopVLA: Learning sufficiency in recurrent refinement for vision-language-action models. *arXiv preprint arXiv:2605.09948*, 2026.

Yongjae Shin, Jongseong Chae, Seongmin Kim, Jongeui Park, and Youngchul Sung. Adaptive action chunking via multi-chunk Q value estimation. *arXiv preprint arXiv:2605.10044*, 2026.

Wenxuan Song, Jiayi Chen, Pengxiang Ding, Yuxin Huang, Han Zhao, Yinchuan Li, Yingcong Chen, Donglin Wang, and Haoang Li. Towards more efficient decoding for autoregressive vision-language-action models. In *Computer Vision – ECCV 2026*, Lecture Notes in Computer Science, pp. 494–511. Springer, 2026. doi: 10.1007/978-3-032-37577-3_27. Preprint: CEED-VLA: Consistency Vision-Language-Action Model with Early-Exit Decoding, arXiv:2506.13725.

Arnav Sukhija, Lenart Treven, Jin Cheng, Florian Dörfler, Stelian Coros, and Andreas Krause. TARC: Time-adaptive robotic control. In *Conference on Robot Learning (CoRL)*, 2026. arXiv:2510.23176.

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.

Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization. In *International Conference on Learning Representations (ICLR)*, 2019. arXiv:1805.11074.

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

Ge Wang, Xinyu Tan, Xiang Li, Man Luo, Chengsi Yao, Shenhao Yan, Jiahao Yang, Fan Feng, Honghao Cai, Xiangyuan Wang, Zhixin Mai, Yiming Zhao, Yatong Han, and Zhen Li. Elastic queries reinforcement learning: Self-aware policy execution for VLA models. *arXiv preprint arXiv:2606.14375*, 2026a.

Hanzhen Wang, Jiaming Xu, Yushun Xiang, Jiayi Pan, Yongkang Zhou, Yong-Lu Li, and Guohao Dai. SpecPrune-VLA: Accelerating vision-language-action models via action-aware self-speculative pruning. In *International Conference on Machine Learning (ICML)*, 2026b.

Songsheng Wang, Rucheng Yu, Zhihang Yuan, Chao Yu, Feng Gao, Yu Wang, and Derek F. Wong. Spec-VLA: Speculative decoding for vision-language-action models with relaxed acceptance. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 26928–26940. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1367.

Ted Xiao, Eric Jang, Dmitry Kalashnikov, Sergey Levine, Julian Ibarz, Karol Hausman, and Alexander Herzog. Thinking while moving: Deep reinforcement learning with concurrent control. In *International Conference on Learning Representations (ICLR)*, 2020. arXiv:2004.06089.

Runze Xu, Xiaolong Shan, Shuang Dai, Yu Wang, and Jincheng Yu. Knowing when to stop: Adaptive action chunking via internal cross-attention dynamics in VLAs. *arXiv preprint arXiv:2609.00908*, 2026a.

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 (NeurIPS)*, volume 38, pp. 164448–164473, 2025. doi: 10.52202/085713-5484.

Weichen Xu, Zhenhua Liu, Lin Luo, Yaobo Liang, Chengtang Yao, Qingyu Mei, Jian Cao, Xixin Cao, Xing Zhang, Jiaolong Yang, and Baining Guo. Continue or replan? Bernoulli-continuation policy learning for adaptive horizon execution. *arXiv preprint arXiv:2608.03483*, 2026b.

Zebin Yang, Yijiahao Qi, Tong Xie, Bo Yu, Shaoshan Liu, and Meng Li. DySL-VLA: Efficient vision-language-action model inference via dynamic-static layer-skipping for robot manipulation. In *ACM/IEEE Design Automation Conference (DAC)*, 2026. arXiv:2602.22896.

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 (NeurIPS)*, volume 37, pp. 56619–56643, 2024. arXiv:2411.02359.

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, 2026a.

Kaidong Zhang, Jian Zhang, Rongtao Xu, Yu Sun, Shuoshuo Xue, Youpeng Wen, Xiaoyu Guo, Minghao Guo, Weijia Liufu, Zihou Liu, Kangyi Ji, Yangsong Zhang, Jiarun Zhu, Jingzhi Liu, Zihang Li, Ruiyi Chen, Meng Cao, Jingming Zhang, Shen Zhao, Xiaojun Chang, Feng Zheng, Ivan Laptev, and Xiaodan Liang. A1: A fully transparent open-source, adaptive and efficient truncated vision-language-action model. *arXiv preprint arXiv:2604.05672*, 2026b.

Brian Zhu, Momen Khalil, E. Harrison, Emanuele Poggi, Philipp Schmitt, Bernd Kast, Philine Meister, Pranav Atreya, Qiyang Li, Finn Ferchau, Cesar Colmenero, Yash Shahapurkar, Gokul Narayanan, Melih Erdogan, Kai Wurm, Georg von Wichert, Oier Mees, Eugen Solowjow, Andrew Wagenmaker, and Sergey Levine. Learning to act while waiting: RL finetuning of generalist robot policies under inference latency. *arXiv preprint arXiv:2608.23831*, 2026.

A PROBLEM DETAILS

Charge. An inference at step t is of kind r_{t-1} and exits at rung k_t after computing rungs K_t: $K_t = \{k_t\}$ for a rule that jumps to its rung (a constant exit, the joint policy), $K_t = \{0, \dots, k_t\}$ for a rule that walks the ladder (the exit head, the threshold, the gate). On a nested ladder, whose deeper rungs continue the shallower computation, $c(K) = \ell(\max K) + (|K| - 1)\,\ell_{\mathrm{ro}}$ with ℓ_{ro} the readout each extra rung adds; a REPLAN saves the encoder's share. The burn is $b_t = \mathrm{round}(c(K_t)/\Delta t)$ with round-half-to-even; real-time chunking uses the floor (Black et al., 2025b), and an axis only qualifies if its rungs span at least two control steps, so that the comparison does not measure the rounding.

Floor of the share. No schedule whose plans hold at most H actions has a long-run share below $b/(H + b)$ with b the burn of its cheapest refresh; a budget below it is refused.

A ratio constraint. $\mathbb{E}[s]$ is a mean of per-episode ratios whose denominators depend on the whole episode, so Equation (2) is not a constrained MDP with an additive per-step cost (Altman, 1999); the dual update follows Tessler et al. (2019) without a saddle-point guarantee, and the kept checkpoint is the best validated one.

B METHOD DETAILS

Algorithm 1 Training one exit policy per task suite under a budget on the inference share.

Require: frozen VLA with exits $0, \dots, K-1$; per-exit latency ℓ_k; control step Δt; budget L on the inference share (Equation (1)); rule threshold t; development tasks with training and held-back validation states

$\pi_\theta \leftarrow$ the consistency rule: stop at the first exit after the first whose relative action change is below t
$\pi^\star \leftarrow \pi_\theta$; validate its greedy policy (success $Y^\star$, share $s^\star$); $\lambda \leftarrow 0$
for iteration $i = 1, \dots, I$ **do**
    roll out the sampled π_θ in the latency-coupled simulator: an inference that stops at exit k burns $\mathrm{round}(\ell_k/\Delta t)$ control steps while the robot executes the filler
    update θ by PPO on the reward of Equation (2): success, less λ times the charged time
    if validation is due **then**
        evaluate the *greedy* π_θ on the validation states: success Y, share s
        $\lambda \leftarrow \max\big(0,\ \lambda + \eta\,(s - L)\big)$ ▷ dual ascent on the deployed share
        if $s \le L$ and $(Y, -s) \succ (Y^\star, -s^\star)$ **then** $\pi^\star \leftarrow \pi_\theta$; $(Y^\star, s^\star) \leftarrow (Y, s)$
        end if
        count patience on feasible validations only; stop when it runs out
    end if
end for
return $\pi^\star$ ▷ the rule itself unless training beat it on validation

Heads. The exit head acts at each rung (continue or stop) from four features: the rung's position, its normalised cumulative cost, the change of its action from the rung below, and whether there was one; v2 adds $\tanh(\|a_{0:3}\|/0.3)$, $\tanh(\|a_{3:6}\|/0.05)$ and $\mathrm{clip}(a_6, -1, 1)$ of the rung's own action; the optional episode memory adds five scalars a threshold on the current inference cannot see – stall, neighbour depth, compute debt, observation change and proprioceptive stall. The refresh head chooses among EXEC, REPLAN, ENCODE from the plan position, the encoding age and the commanded speed; its v2 adds the same three phase features of the action just applied. The joint policy reads the refresh head's inputs. All heads are two-layer tanh MLPs of width 32 with a value head, trained by PPO (clip 0.2, four epochs, entropy 0.01, GAE $\lambda = 0.95$), 16 episodes per round, validation on 16 held-out episodes every five rounds.

Composition versus joint policy. The composed arm runs the refresh head and the exit head together. It differs from the joint policy in four ways, not one: joint versus separate training; when the rung is decided (before the inference, jumping to it, versus walking rungs $0..k$ and paying a

readout at each); the exit head running under a learned refresh schedule it was not trained under; and a latency model with an encoder share the exit head never saw. J3 − J1 is reported with this confound attached.

Checks. Every run is re-checked from its logs: each decision's features and probabilities are recomputed from the checkpoint (the v2 phase from the logged applied action), every episode is verified to have run under the declared filler, and arms are verified to start from identical states (a task whose arms do not is excluded before its outcomes are read).

C BASELINES

Hull. All constant configurations are evaluated on the validation states; the comparator at share s is their upper convex hull, realised per episode by randomising between the two neighbouring vertices.

DeeR-VLA-style threshold. Walk the ladder and stop at the first rung whose action differs from the previous rung's by less than a threshold (Yue et al., 2024); the threshold is fitted on the validation episodes at the learned policy's share, searched again between the closest candidate's neighbours when the grid straddles it, and recorded as not buildable, with the reason, when no threshold reaches the share.

AVA-VLA gate. AVA-VLA trains its exit gate in a fourth stage, after behaviour cloning, a latent warm-up and PPO, with every other module frozen: a reasoning step is labelled sufficient when the stage-3 critic's value three steps ahead exceeds its own by less than 0.05, the look-ahead clamped to the last step, trained by binary cross-entropy, and the loop stops when the gate's score exceeds 0.55 (Lei & Shan, 2026). On our ladder the gate module is theirs, the label is their stage-4 rule, and the value of rung k for an episode is measured by running that episode with every inference executing rung k, where their critic estimates it; the threshold is fitted at the learned policy's share. An earlier build that labelled rows with the episode's outcome is recorded as not theirs and is not reported.

AVA-VLA as reported. Table 7 quotes Lei & Shan (2026) (arXiv v1): their Table 1 (one policy for all four suites) and Table 2 (latency on LIBERO-Spatial, one A100, batch size 1). Their paper is not self-consistent on the latency without early exit: 312 ms in Tables 2 and 5, 156 ms for a fixed depth of five in Table 13; we quote Table 2.

Table 7: AVA-VLA as reported by Lei & Shan (2026): success (%) on the four LIBERO suites, one policy for all suites (their Table 1), and latency on LIBERO-Spatial (their Table 2). Not reproduced: no weights are released.

	Spatial	Object	Goal	Long	Average	mean / P90 latency
OpenVLA-OFT (as reported there)	97.7	98.0	96.1	95.3	96.8	–
AVA-VLA, without early exit	–	–	–	–	–	312 / 340 ms (5.0 steps)
AVA-VLA	97.8	99.4	97.8	98.1	98.3	145 / 189 ms (2.3 steps)

AVA-VLA and other efficient VLAs. AVA-VLA's deployed inference runs the vision encoder and the language model in full, plus a text-only language-model pass and a latent loop of at most five small steps; its exit shortens only that loop, and the VLA itself is retrained in four stages to obtain it (Lei & Shan, 2026). On the same backbone its cost per inference is therefore at least one full OpenVLA-OFT forward (an analytical bound, not a measurement), which exiting after layer 24 undercuts by 25%. As reported, on the four standard LIBERO suites AVA-VLA reaches 98.3% average success against 96.8% for OpenVLA-OFT in their one-policy setting, at 145 ms per inference on an A100 with its exit and 312 ms without (Section C); those are their protocol and hardware, not comparable to our charged latencies, and the same holds for the other efficient VLAs whose published LIBERO results Table 8 collects. Many of those make each layer or token cheaper – caching or pruning visual tokens (e.g. VLA-Cache, SpecPrune-VLA, FlashVLA), speculative decoding (SpecVLA), quantization (SQIL, QuantVLA) – which is complementary to choosing how many layers

to run. The methods whose decision rule transfers to our exits we re-run under our protocol on the same episodes (Tables 4, 6 and 12): the consistency exit of DeeR-VLA (Yue et al., 2024) and A1 (Zhang et al., 2026b), and AVA-VLA's gate (G4), trained on its own stage-4 label per suite and compared with ours at our policy's share on every task where it can reach that share; where even its most expensive threshold spends less, it has no point at matched compute and is reported as not buildable. Our re-implemented gate falls below the constant mixture at the same compute, so that comparison bounds our re-implementation of their rule, not AVA-VLA itself. Methods evaluated on other benchmarks only – DeeR-VLA on CALVIN (Yue et al., 2024), ElegantVLA on SimplerEnv (Li et al., 2026a) – are compared on our ladder instead (Section 5).

Table 8: Efficient VLAs as reported on standard LIBERO (not re-run), grouped by what they save: success of the method / its base model (%, four-suite mean) and the efficiency each paper reports. There the simulator pauses during inference; our row charges inference time, on a harder benchmark, and is not comparable. *the paper's own base evaluation; §mean of its per-suite numbers, by us; ¶visual-token FLOPs; †LIBERO-Long only; ‡language-model latency only.

Method	builds on	success / base	efficiency (method vs. base)	source, hardware
Adaptive depth or early exit: how much of the model runs per step				
AVA-VLA (Lei & Shan, 2026)	OpenVLA-OFT	98.3 / 96.8	145 vs. 312 ms (no exit)	Tab. 1–2, A100
DySL-VLA (Yang et al., 2026)	OpenVLA-OFT	96.5 / 97.1	27.4 vs. 53.0 ms‡	Tab. 3, A6000
A1 (Zhang et al., 2026b)	Molmo 7B (own)	96.6 / 96.5	205.0 vs. 243.0 TFLOPs / episode	Tab. 1, 5, n.r.
LoopVLA (Shen et al., 2026)	Qwen3-VL 2B (own)	96.0 / 94.9	10.93 vs. 10.49 Hz	Tab. 1, RTX 4090
RD-VLA (Tur et al., 2026)	MiniVLA 0.5B (own)	92.5 / 93.0	7.93 vs. 12 iterations	Tab. I, n.r.
Fewer or cheaper tokens: pruning, caching, routing				
LightVLA (Jiang et al., 2026)	OpenVLA-OFT	97.4 / 94.8*	21 vs. 34 ms; 3.6 vs. 8.8 TFLOPs	Tab. I–II, H20
SP-VLA (Li et al., 2026b)	OpenVLA	74.9 / 74.6	0.23 vs. 0.32 s	Tab. 1, 5, RTX 4090
VLA-Cache (Xu et al., 2025)	OpenVLA-OFT	97.4 / 96.8*	62.59 vs. 79.05 ms; 3.097 vs. 4.013 TFLOPs	Tab. 2, RTX 4090
SpecPrune-VLA (Wang et al., 2026b)	OpenVLA-OFT	96.1 / 96.6*	75.1 vs. 109.0 ms; 43% of FLOPs	Tab. 1, 10, A800
CogVLA (Li et al., 2025)	OpenVLA 7B	97.4 / 76.5	0.091 vs. 0.254 s (OFT 0.132 s); 2.72 vs. 8.48 TFLOPs	arXiv v3 Tab. 1, 3, A800
VLA-Pruner (Liu et al., 2025)	π_0	95.1 / 94.0*	69.45 vs. 104.53 ms (50% tokens)	Tab. 4, RTX 4090
FlashVLA (Tan et al., 2025)	OpenVLA	73.7 / 74.4*	52.9 vs. 82.7 ms; 0.58 vs. 1.31 TFLOPs¶	Tab. 1, H100
Faster decoding or lower precision				
OpenVLA-OFT (Kim et al., 2025)	OpenVLA 7B	97.1 / 76.5	0.112 vs. 0.240 s; 71.4 vs. 4.2 Hz	Tab. I–II, A100
CEED-VLA (Song et al., 2026)	OpenVLA	62.2† / 60.4†	25.60 vs. 7.08 Hz	arXiv v1 Tab. 2, H100
Spec-VLA (Wang et al., 2025)	OpenVLA	75.1§ / 76.0§	1.22–1.42× decoding speed	Tab. 1, A100
SQIL (Park et al., 2025)	OpenVLA	73.2 / 73.8*§	374.7 vs. 955.2 ms; 4.0 vs. 15.2 GB (INT4)	Tab. 1, 3, 9, Jetson AGX Orin
QuantVLA (Zhang et al., 2026a)	$\pi_{0.5}$	97.6 / 97.1	1.28 vs. 4.27 GB (LLM+DiT, W4A8)	Tab. 2, A100
This paper: LIBERO-Plus difficulty 5, held-out tasks, inference time charged to the robot				
Ours (exit policy, frozen backbone)	OpenVLA-OFT	71.4 / 78.1	113 vs. 199 ms charged (1.76×); no per-task calibration	Table 4

D STATISTICS

The unit is the task. For a contrast between arms a and b the statistic is the mean over tasks of $Y_a - Y_b$; its null distribution is the exact sign-flip distribution over the 2^n sign patterns. Per-arm intervals are t intervals over tasks; a bootstrap over so few tasks is anti-conservative and is not used. Four of the seven latency tasks are perturbations of one base task; with the base task as the unit, the smallest possible p is 0.125, and every contrast at 0.016 has the same sign on all four.

E ADDITIONAL RESULTS

Statistics of the main results. On the development-held-out tasks (6 tasks), our policy reaches 0.714 success at 113 ms charged per inference; it is below, not significantly, full depth (0.781 at 199 ms; -0.067 (2/0/4 tasks up/tied/down, $p = 0.38$; 15 episodes won, 29 lost)) at 1.76× less time per inference. At the same share it is on par with the DeeR-style threshold ($+0.010$ (1/1/1 tasks up/tied/down, $p = 1.00$; 7 episodes won, 6 lost)); is on par with or above the best constant mixture ($+0.131$ (3/2/0 tasks up/tied/down, $p = 0.25$; 34 episodes won, 11 lost)). The AVA-VLA-style gate's stage finished on 5 of these tasks: its threshold could not reach our share on 0 (not buildable), and it ran on 5, of which 4 matched our share; there ours is on par with or above it ($+0.221$ (4/0/0 tasks up/tied/down, $p = 0.12$; 41 episodes won, 10 lost); Table 11). Over 3 training seeds the difference to the DeeR-style threshold is $+0.019$ on the 1 task where every seed matched its share (per seed -0.029, $+0.057$, $+0.029$; tasks up/down 1/0, $p = 1.00$), and to full depth -0.044 over 6 tasks (Table 2). Against its own start, the rule alone with RL off, on the same tasks and episodes, the policy after RL is below, not significantly, it in success (-0.033, $p = 0.16$, 6 tasks) at a share -0.016 different.

Table 9: Development-held-out tasks, seed 0: our exit policy (rule start + constrained RL) against every baseline. Arms: mean over every task at its own share; contrasts: only tasks where both sides lie within 0.02 of the hull's realised share. Above the light rule: more compute per inference than our budget; bold: the best success at or below it. DeeR-VLA/A1- and AVA-VLA-style: their exit rules, re-implemented on our exits (Section C).

Method	**Arm**	**Success** ↑	**Tasks**	**Share** ↓	**ms/inf** ↓	**s/ep** ↓
Full depth (layer 32)	E3	0.781 ± 0.287	6	0.341	199	2.61
Exit at layer 24	E2	0.786 ± 0.358	6	0.278	149	1.97
Exit at layer 16	E1	0.438 ± 0.387	6	0.203	100	1.78
Hull (best constant mixture)	H0	0.581 ± 0.306	6	0.220	108	1.77
DeeR-VLA/A1-style exit	B4	$\mathbf{0.724 \pm 0.305}$	6	0.232	117	1.69
Ours	E4	0.714 ± 0.334	6	0.227	113	1.67
Contrast	**Δ success**	**Tasks**	**up/tie/down**	p	**ms/inf**	**s/ep**
E4 − B4	+0.010	3 (3 excl.)	1/1/1	1.00	113 / 115	1.70 / 1.72
E4 − H0	+0.131	5 (1 excl.)	3/2/0	0.25	113 / 110	1.73 / 1.83
E4 − E3	−0.067	6	2/0/4	0.38	113 / 199	1.67 / 2.61

Fresh-task accounting and statistics. Of the 25 fresh tasks, 17 are pooled here; tasks 424, 1521, 1825, 1826, 1850 are excluded by the pairing check (the arms' first actions differ, so the perturbation is not reproduced across arms), and 3 had not finished by the submission deadline. On the fresh confirmation tasks (17 tasks; on 5 of them every arm scores the same, so they carry no comparison), our policy reaches 0.245 success at 119 ms charged per inference; it trails full depth (0.531 at 199 ms; -0.286 (0/6/11 tasks up/tied/down, $p < 0.01$; 8 episodes won, 178 lost)) at 1.67× less time per inference. At the same share it is on par with the DeeR-style threshold (-0.010 (4/5/2 tasks up/tied/down, $p = 0.94$; 16 episodes won, 20 lost)); is below, not significantly, the best constant mixture (-0.100 (4/1/7 tasks up/tied/down, $p = 0.30$; 42 episodes won, 84 lost)). The AVA-VLA-style gate's stage finished on 9 of these tasks: its threshold could not reach our share on 2 (not buildable), and it ran on 7, of which 4 matched our share; there ours is on par with or above it ($+0.200$ (3/1/0 tasks up/tied/down, $p = 0.25$; 30 episodes won, 2 lost); Table 15).

Detailed tables. Per arm, with 95% t intervals over tasks and every contrast (Table 9); the rule start alone, RL off (Table 5), and what RL changes in it (Table 10); and the confirmation per family and efficiency (Tables 14 and 16). Every table is generated from its pool, whose path and SHA-256 are in the table's source.

Table 10: What RL changes in its rule start, on the development-held-out tasks (seed 0, 35 episodes per task). Top: exit layers of each head in its own episodes. Middle: at each of the RL head's 3,099 inferences, the exit the start rule ($t = 0.20$) would take on the same logged rung actions (re-derived offline; on the rule head's own 2,993 inferences it reproduces every exit). Bottom: the fraction of visits to layers 16 and 24 at which the deployed (greedy) RL head stops, by the relative action change c between that exit and the one below, the quantity the rule thresholds (visits in parentheses).

Exit layer, share of inferences	layer 8	layer 16	layer 24	layer 32	mean layer
Rule start (RL off)	0.000	0.605	0.301	0.094	19.9
After RL	0.000	0.759	0.212	0.029	18.2
After RL vs the start rule at the same observation	same exit	shallower	deeper		
share of inferences	0.783	0.217	0.000		
Fraction that stops after RL, by action change c	$c < 0.20$	$[0.20, 0.24)$	$[0.24, 0.26)$	$[0.26, 0.28)$	$c \geq 0.28$
at layer 16	1.00 (1818)	1.00 (372)	0.92 (123)	0.39 (126)	0.00 (660)
at layer 24	1.00 (517)	1.00 (100)	0.96 (26)	0.54 (26)	0.00 (78)

Table 11: Development-held-out tasks whose gate stage finished (seed 0): the AVA-VLA-style gate beside every arm. Arms: mean over every task at its own share; contrasts: only tasks where both sides lie within 0.02 of the hull's realised share. †only tasks where the rule could be fitted at our share. Above the light rule: more compute per inference than our budget; bold: the best success at or below it. DeeR-VLA/A1- and AVA-VLA-style: their exit rules, re-implemented on our exits (Section C).

Method	**Arm**	**Success** ↑	**Tasks**	**Share** ↓	**ms/inf** ↓	**s/ep** ↓
Full depth (layer 32)	E3	0.869 ± 0.236	5	0.341	199	2.47
Exit at layer 24	E2	0.914 ± 0.181	5	0.279	149	1.80
Exit at layer 16	E1	0.491 ± 0.479	5	0.203	100	1.73
Hull (best constant mixture)	H0	0.663 ± 0.293	5	0.224	110	1.71
DeeR-VLA/A1-style exit	B4	0.817 ± 0.249	5	0.234	118	1.58
AVA-VLA-style gate	G4	0.594 ± 0.264	5	0.229	115	1.84
Ours	E4	$\mathbf{0.823 \pm 0.242}$	5	0.229	114	1.56
Contrast	Δ success	**Tasks**	**up/tie/down**	p	**ms/inf**	**s/ep**
E4 − B4	+0.010	3 (2 excl.)	1/1/1	1.00	113 / 115	1.70 / 1.72
E4 − H0	+0.164	4 (1 excl.)	3/1/0	0.25	114 / 112	1.60 / 1.78
E4 − E3	−0.046	5	2/0/3	0.69	114 / 199	1.56 / 2.47
E4 − G4	+0.221	4 (1 excl.)	4/0/0	0.12	114 / 117	1.60 / 1.91

Table 12: Success per LIBERO-Plus perturbation family on the development-held-out tasks (one task each; seed 0), the average over tasks, and the charged time per inference. Adaptive rules are fitted at our policy's share on each task. Above the light rule: more compute per inference than our budget; bold: the best success at or below it. DeeR-VLA/A1- and AVA-VLA-style: their exit rules, re-implemented on our exits (Section C).

Method	Background	Camera	Language	Layout	Light	Noise	**Average** ↑	ms/inf ↓
Full depth (layer 32)	0.86	0.54	0.97	1.00	0.97	0.34	0.781	199
Exit at layer 24	0.94	0.66	1.00	1.00	0.97	0.14	0.786	149
Exit at layer 16	**1.00**	0.06	0.26	0.37	0.77	0.17	0.438	100
Hull (best constant mixture)	**1.00**	0.37	0.57	0.60	0.77	0.17	0.581	108
DeeR-VLA/A1-style exit	**1.00**	**0.60**	0.60	**0.97**	**0.91**	**0.26**	**0.724**	117
Ours	**1.00**	0.57	**0.66**	**0.97**	**0.91**	0.17	0.714	113

Table 13: The speed/accuracy knob on the development-held-out tasks every operating point ran (seed 0): the rule start at several thresholds t (no retraining) and our policy after RL. Differences are per task, then averaged; the comparison to the DeeR-style threshold is at each point's own share, over the tasks where the two shares matched.

Operating point	success	share	ms/inf	speed-up	vs full depth (up/tie/down)	vs DeeR-style at its share
Rule start, $t = 0.15$	0.857	0.268	142	1.40×	+0.014 (2/1/1)	+0.038 (2/1/0 of 3)
Rule start, $t = 0.20$	0.821	0.245	125	1.60×	-0.043 (1/2/1)	+0.043 (3/1/0 of 4)
Ours: RL from $t = 0.20$	0.800	0.229	114	1.75×	-0.043 (2/0/2)	+0.010 (1/1/1 of 3)

Table 14: Success per perturbation family on the fresh confirmation tasks (tasks per family in parentheses), the average over tasks, and the charged time per inference. Above the light rule: more compute per inference than our budget; bold: the best success at or below it. DeeR-VLA/A1- and AVA-VLA-style: their exit rules, re-implemented on our exits (Section C). †Over the tasks where the rule could be fitted at our share (not buildable on the others).

Method	Camera (4)	Layout (7)	Noise (2)	Robot init (4)	**Average** ↑	ms/inf ↓
Full depth (layer 32)	0.74	0.59	0.97	0.00	0.531	199
Exit at layer 24	0.33	0.45	0.71	0.00	0.346	150
Exit at layer 16	0.23	0.33	0.06	**0.00**	0.195	100
Hull (best constant mixture)	0.35	**0.47**	0.36	**0.00**	**0.319**	111
DeeR-VLA/A1-style exit† (16 of 17)	0.44	0.26	**0.46**	**0.00**	0.250	124
Ours	**0.46**	0.27	0.20	**0.00**	0.245	119

Table 15: Fresh confirmation tasks whose gate stage finished, where the gate ran or was recorded as not buildable at our share. Arms: mean over every task at its own share; contrasts: only tasks where both sides lie within 0.02 of the hull's realised share. †only tasks where the rule could be fitted at our share. Above the light rule: more compute per inference than our budget; bold: the best success at or below it. DeeR-VLA/A1- and AVA-VLA-style: their exit rules, re-implemented on our exits (Section C).

Method	**Arm**	**Success** ↑	**Tasks**	**Share** ↓	**ms/inf** ↓	**s/ep** ↓
Full depth (layer 32)	E3	0.600 ± 0.275	9	0.342	199	3.21
Exit at layer 24	E2	0.467 ± 0.336	9	0.275	150	2.55
Exit at layer 16	E1	0.251 ± 0.270	9	0.201	100	2.02
Hull (best constant mixture)	H0	$\mathbf{0.410 \pm 0.253}$	9	0.226	114	2.13
DeeR-VLA/A1-style exit	B4	0.349 ± 0.320	9	0.243	124	2.38
AVA-VLA-style gate	G4	0.245 ± 0.357	7†	0.225	113	2.27
Ours	E4	0.368 ± 0.326	9	0.236	120	2.30

Contrast	Δ success	**Tasks**	**up/tie/down**	p	**ms/inf**	**s/ep**
E4 − B4	+0.012	7 (2 excl.)	3/3/1	0.62	119 / 123	2.31 / 2.39
E4 − H0	−0.045	7 (2 excl.)	3/0/4	0.73	119 / 118	2.31 / 2.22
E4 − E3	−0.232	9	0/2/7	0.02	120 / 199	2.30 / 3.21
E4 − G4	+0.200	4 (5 excl.)	3/1/0	0.25	114 / 111	2.17 / 2.28

Table 16: Efficiency on the fresh confirmation tasks. Above the light rule: more compute per inference than our budget; bold: the best success at or below it. DeeR-VLA/A1- and AVA-VLA-style: their exit rules, re-implemented on our exits (Section C). †Over the tasks where the rule could be fitted at our share (not buildable on the others); its speed-up is over full depth on those tasks.

Method	mean exit layer	layers skipped	ms/inf (speed-up)	infer. s/ep	robot s/ep	success within ≤6 s	success within ≤8 s	success
Full depth	32.0	0%	199 (1.00×)	3.24	9.43	0.003	0.207	0.531
Layer 24	24.0	25%	150 (1.33×)	2.64	9.64	0.029	0.308	0.346
Layer 16	16.0	50%	100 (2.00×)	2.07	10.30	0.000	0.175	0.195
Hull	14.3	55%	111 (1.79×)	2.15	9.83	**0.013**	**0.264**	**0.319**
DeeR-VLA/A1-style† (16 of 17)	19.9	38%	124 (1.61×)	2.46	10.18	0.004	0.195	0.250
Ours	19.1	40%	119 (1.67×)	2.39	10.19	0.002	0.190	0.245

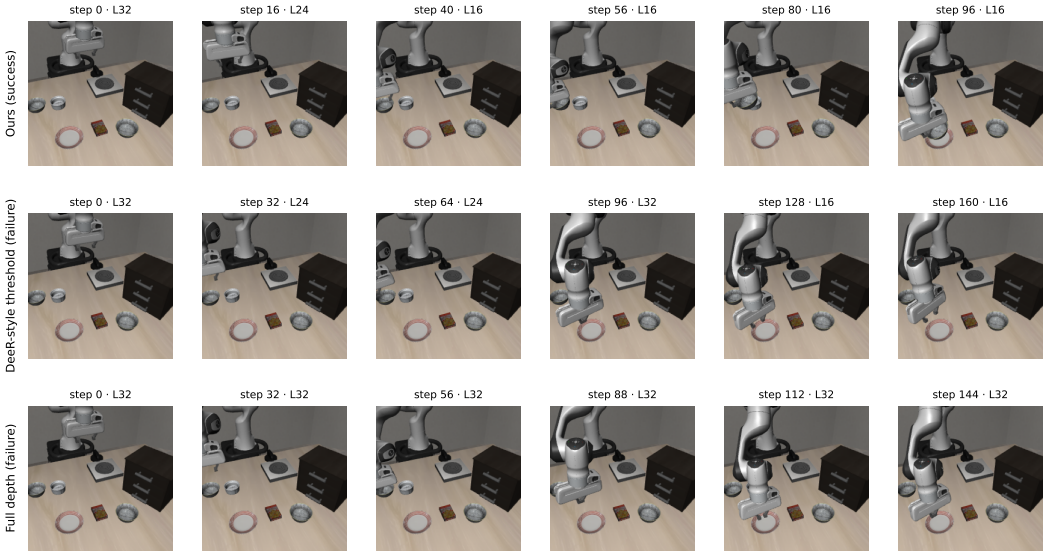

Figure 3: One held-out episode (task 632, camera viewpoint, initial state 0) under three rules (rows) at evenly spaced inferences, each frame labelled with its control step and exit layer. Selected because the outcomes differ (ours succeeds; the DeeR-style threshold at the same share and full depth do not): it shows the exit pattern, not typical success. Third-person camera shown; the policy also reads the wrist camera.

Table 17: The training/deployment mismatch (Section 4) on the first five tasks of the v2 exit head (a sixth, 312, was refused by the pairing check): success @ share of the deployed (greedy) head with λ on the sampled share and on the deployed share, beside the hull at the latter's share and the DeeR-style threshold. Budget 0.267.

task	λ on sampled	λ on deployed	hull	threshold	layer 24
2146	0.086 @ 0.261	0.714 @ 0.206	0.771	0.829	0.971 @ 0.283
632	0.600 @ 0.334	0.514 @ 0.320	0.714	0.600	0.714 @ 0.277
1056	0.943 @ 0.341	0.943 @ 0.341	1.000	0.943	1.000 @ 0.278
1733	–	0.971 @ 0.225	0.514	0.457	1.000 @ 0.280
170	–	0.857 @ 0.265	1.000	0.971	0.914 @ 0.280

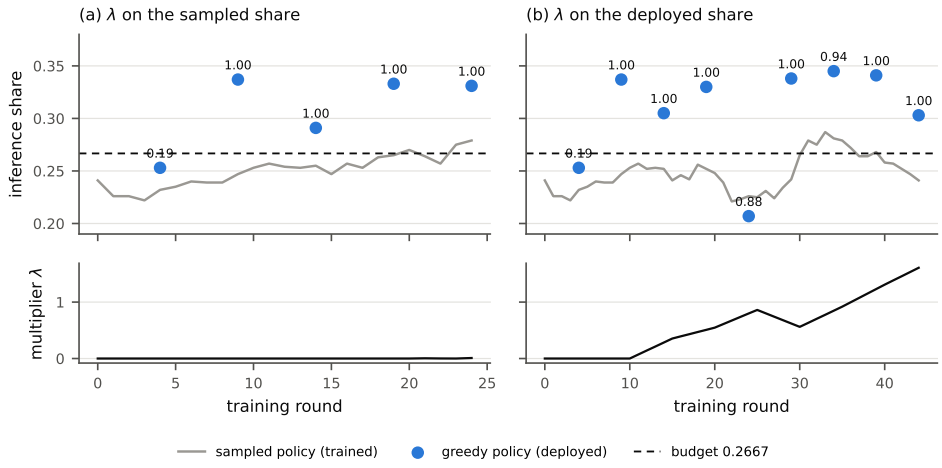

Figure 4: Training the v2 exit head on task 2146 at budget 0.267. Grey: the share of the sampled policy PPO trains; blue: the greedy policy that validation keeps and evaluation deploys, labelled with its validation success; below, the multiplier. (a) With λ on the sampled share, the sampled policy sits at the budget, λ stays at zero, and the deployed policy is either infeasible or collapsed (0.19). (b) With λ on the deployed share, λ rises and the deployed policy reaches 0.88 within budget at round 24.

Charging the readout. On the nested ladder an inference that stops at exit j costs that exit's latency plus one readout per exit it queried before it. Our timing charges the readout as free (the measured value was clamped to zero); a later measurement gives 4.14 ms. Charged, the four exits cost 76.8, 118.1, 164.4 and 207.3 ms instead of 76.8, 114.0, 156.1 and 194.9 ms, which at a 50 ms control step burn 2, 2, 3 and 4 steps either way: no inference crosses a step, so every share and every simulated outcome is exactly as run, and so is every charged time in the tables. Only the unrounded model time of the rules that walk the ladder grows: ours queries 2.55-2.62 exits per inference (+6.4-6.7 ms) and the DeeR-style threshold 2.48-2.63 (+6.1-6.7 ms); constant exits and the hull query one. Charged and model time are different scales and are not mixed in any comparison.

Latency cells. Pooled over seven tasks (successes of 350): `ref` 313, `ref` again 313, repeat 173, pause 278, async 303, stale 232.

Why the v1 exit head does not move. At rung 0 its four inputs are the same for every observation, so its stop probability there is a constant; across states it ranged 0.41-0.41 on task 2146 and 0.20-0.20 on task 170. A per-episode clairvoyant rung chooser bounds what any exit rule could gain at the head's share: about 0.16 on three of six tasks and nothing on the other three.