DEPTHFLOW-VLA: ROUTED LOW-NOISE FLOW REFINEMENT FOR COMPACT VLA MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Compact vision-language-action (VLA) models commonly allocate the same action-decoding budget to every chunk despite substantial variation in prediction difficulty. We present DepthFlow-VLA, a coarse-to-refine flow-matching decoder in which Stage A first produces a coarse action and a lightweight router selectively invokes restart-style low-noise Stage B. Training combines difficulty-weighted velocity supervision, routing supervision, and a Stage-B objective constructed from replayed Stage-A predictions. On LIBERO, Stage-A-only, adaptive, and uniform-refinement modes achieve 95.9%, 98.3%, and 98.5%, respectively, while adaptive routing uses 7.54 rather than 13.0 decoder evaluations per chunk. On SimplerEnv-WidowX, adaptive and uniform refinement achieve 80.2% and 81.3%, and equal-budget random routing reaches 76.0%. A separately trained vanilla flow-matching control reaches 94.8% on LIBERO and 72.2% on WidowX at six decoder evaluations. On a single H100 with batch size 1, adaptive routing reduces mean end-to-end latency from 59.2 to 42.6 ms relative to uniform refinement.

1 INTRODUCTION

Vision-language-action (VLA) models are a promising paradigm for robotic manipulation, connecting pretrained multimodal representations with robot control (Zitkovich et al., 2023; Sapkota et al., 2025). By inheriting semantic priors from large-scale vision-language pretraining, they offer richer grounding and stronger instruction following than conventional end-to-end policies, and have emerged as a natural foundation for language-conditioned manipulation across diverse objects, scenes, and task descriptions. Nevertheless, many high-performing VLA systems rely on multi-billion-parameter backbones, which hinders affordable training, iteration on modest hardware, and deployment on resource-constrained robots.

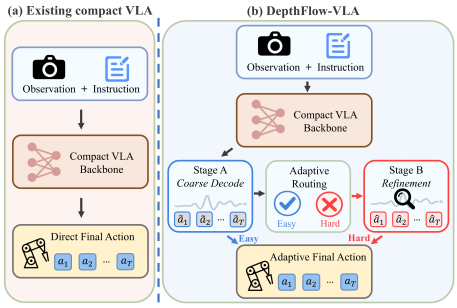

Figure 1: Fixed-path compact VLA decoders process every sample along a single-stage path, whereas DepthFlow-VLA performs coarse Stage-A decoding followed by adaptive routing that invokes Stage-B refinement only for higher-priority samples.

Compact and efficient VLAs have therefore attracted growing attention, with recent work seeking to reduce model size, adaptation cost, and training overhead while preserving manipulation performance (Yu et al., 2025; Shukor et al., 2025; Wang et al., 2026b; Dong et al., 2025; Kim et al., 2025). In parallel, the decoding stage itself has become an increasingly active design axis, encompassing continuous action modeling, iterative refinement, and adaptive inference (Black et al., 2024; Liang et al., 2025; Jiang et al., 2025; Wang et al., 2026a). A common limitation of fixed-path compact VLAs, illustrated in Fig. 1, is that they allocate the same decoding path to every sample even though action difficulty varies substantially across action chunks. This motivates a natural question: can compact VLAs benefit from allocating decoder compute adaptively?

We address this question with DepthFlow-VLA, which reallocates compute inside the continuous action decoder itself, rather than through execution-schedule adaptation or an external correction head. Under a fixed compact backbone, the decoder first produces a coarse action estimate via a short

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.

Stage-A path, and invokes a dedicated Stage-B process only when the router assigns the coarse state a high refinement score. Stage B is not simply a deeper sampler along the same trajectory: the coarse prediction is a structured action-space state that both determines whether refinement is needed and anchors a re-corrupted low-noise correction flow. This turns action generation from uniformly deeper integration into coarse-prediction-conditioned error correction. A joint training objective couples difficulty-aware velocity supervision, depth-selection learning, and a targeted low-noise refinement loss constructed from re-corrupted Stage-A coarse predictions.

Our contributions are threefold. (i) We adapt restart-style re-corruption and correction to compact VLA action chunks, using the Stage-A coarse prediction as the correction anchor. (ii) We introduce a coarse-state-conditioned router that preserves most of the full-refinement gain while substantially reducing the expected decoder budget. (iii) We evaluate the router through same-checkpoint random-routing controls, compute-controlled baselines, paired offline benefit metrics, and cross-embodiment simulation and real-world experiments.

2 Related Work

2.1 VLA Models for Robotic Manipulation

VLA models unify perception, language understanding, and action prediction within a single framework, leveraging pretrained vision-language priors for grounded physical interaction. Early transformer-based robot policies such as RT-1 (Brohan et al., 2022) demonstrated the value of large-scale real-world manipulation data, and RT-2 (Zitkovich et al., 2023) established the VLA formulation by incorporating web-scale vision-language pretraining into robotic control. RoboFlamingo (Li et al., 2023) further showed that open-source VLMs can be adapted into effective robot imitators with relatively simple modifications. Building on these foundations, recent research has broadened into efficiency, reasoning, decoding strategies, and real-time adaptation (Liu et al., 2024; Din et al., 2025; Sapkota et al., 2025), setting the stage for the more focused directions reviewed below.

2.2 Compact and Efficient VLA Models

Reducing training and deployment cost while preserving manipulation performance has become a prominent research direction. TinyVLA (Wen et al., 2025) emphasizes fast inference and data efficiency under small model sizes, while SmolVLA (Shukor et al., 2025) reduces compute through efficient architecture design and system-level optimization. A parallel line of work improves efficiency on the action side: VLA-Adapter (Wang et al., 2026b) introduces a lightweight policy module with Bridge Attention that injects vision-language conditions into the action space; VITA-VLA (Dong et al., 2025) transfers action capabilities from a small action expert to a VLM and thereby avoids expensive end-to-end pretraining; and OpenVLA-OFT (Kim et al., 2025) obtains gains through improved decoding and continuous action representations. Taken together, compact VLA work has largely targeted backbone simplification, lightweight bridging, and adaptation strategies, while typically leaving the per-sample decoding procedure itself uniform across the dataset.

2.3 Action Generation and Decoding in VLA Models

How actions are produced from multimodal context is a central design choice. Early VLAs such as OpenVLA (Kim et al., 2024) tokenize actions and predict them autoregressively, which aligns naturally with language-model-style training. To improve action quality, parallelism, and correction ability, Discrete Diffusion VLA (Liang et al., 2025) applies discrete diffusion over action chunks while retaining the token interface, and DFM-VLA (Chen et al., 2026) uses discrete flow matching for iterative full-sequence refinement, thereby alleviating the irreversible commitment problem of strictly sequential decoding. A parallel strand models actions directly in continuous space: π_0 (Black et al., 2024) augments vision-language features with a flow-based action expert and shows that continuous flow matching effectively captures high-frequency action chunks. For continuous flow decoders, the integration schedule determines the number of decoder evaluations used to generate an action chunk. A fixed step count gives every chunk the same decoding budget; DepthFlow-VLA instead selects whether to refine each coarse prediction.

2.4 Adaptive Computation in Deep Networks

The underlying principle of spending more compute on harder inputs has been explored extensively outside robotics, starting from Adaptive Computation Time (Graves, 2016) for recurrent networks and continuing into modern transformer architectures. Recent efficient model design has increasingly shifted from static compression toward input-dependent computation, in which different tokens or samples receive different amounts of processing. In vision transformers, Mixture of Nested Experts (Jain et al., 2024) dynamically routes visual tokens to experts with different compute–accuracy trade-offs. In language models, AdaMoE (Zeng et al., 2024) introduces token-adaptive routing with null experts, and AdaInfer (Fan et al., 2024) skips unnecessary layers at inference time based on intermediate signals. Generative modeling also studies the trade-off between sampling cost and quality. Consistency models (Song et al., 2023) support one- or few-step generation through a learned consistency mapping, whereas restart sampling (Xu et al., 2023) alternates forward noising with backward ODE integration.

Adaptive and restart-style VLA inference. Restart sampling alternates forward noising with backward ODE integration to reduce accumulated generation error (Xu et al., 2023). The re-corruption used by Stage B is directly related to this principle: both perturb an intermediate estimate before a new reverse integration. DepthFlow-VLA builds on this restart principle by conditioning both the decision to restart and the restart state on the Stage-A coarse action, yielding a per-chunk refinement branch inside a compact VLA decoder.

Recent VLA methods adapt inference along complementary axes. AsyncVLA (Jiang et al., 2025) uses a confidence rater to identify low-confidence action tokens and regenerates them with token-specific asynchronous flow times. DepthFlow-VLA instead makes one decision for the complete action chunk and either retains the Stage-A prediction or invokes a shared low-noise restart. AutoHorizon (Wang et al., 2026a) dynamically changes how many actions from a predicted chunk are executed, adapting the observation–execution schedule rather than the amount of action-generation compute. These methods adapt different parts of inference: AutoHorizon changes the execution horizon, AsyncVLA regenerates selected action tokens, and DepthFlow-VLA allocates a restart-style refinement stage at the action-chunk level.

3 Methodology

DepthFlow-VLA allocates action-decoder computation through a routed coarse-to-refine process. Stage A produces a coarse action estimate that serves as both the routing state and the anchor for refinement. A lightweight router then selects action chunks for low-noise Stage-B correction. As illustrated in Fig. 2, the framework comprises a compact vision-language backbone, a shared two-stage action decoder, an adaptive routing module, and a joint training objective.

3.1 Preliminaries

Problem Formulation. At each control timestep t, the robot receives an observation $o_t = \{I_t^1, \ldots, I_t^n, \ell_t, s_t\}$ consisting of n RGB images, a language instruction ℓ_t, and proprioceptive state s_t. The policy maps o_t to a continuous action chunk $A_t = [a_t, a_{t+1}, \ldots, a_{t+H-1}] \in \mathbb{R}^{H \times d_a}$, where H is the action horizon and d_a the action dimension. The predicted chunk is then executed in a receding-horizon manner, with the next chunk re-predicted from updated observations.

Flow Matching Action Generation. We model action chunks with conditional flow matching. Let $x \in \mathbb{R}^{H \times d_a}$ denote a normalized action chunk, $\epsilon \sim \mathcal{N}(0, I)$ Gaussian noise, and $x_\tau = \tau\epsilon + (1-\tau)x$ the noisy sample at $\tau \in (0, 1]$. Given vision-language features Z_t, the action decoder predicts the velocity field $\epsilon - x$ via

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{\tau,x,\epsilon}\left[\|v_\theta(x_\tau, Z_t, s_t, \tau) - (\epsilon - x)\|_2^2\right]. \tag{1}$$

At inference, Euler integration from $\tau = 1$ to $\tau = 0$ recovers the final action chunk.

Compact VLA Backbone. We instantiate DepthFlow-VLA on a compact SmolVLM-based backbone (Marafioti et al., 2025) equipped with a lightweight Transformer action decoder. The backbone

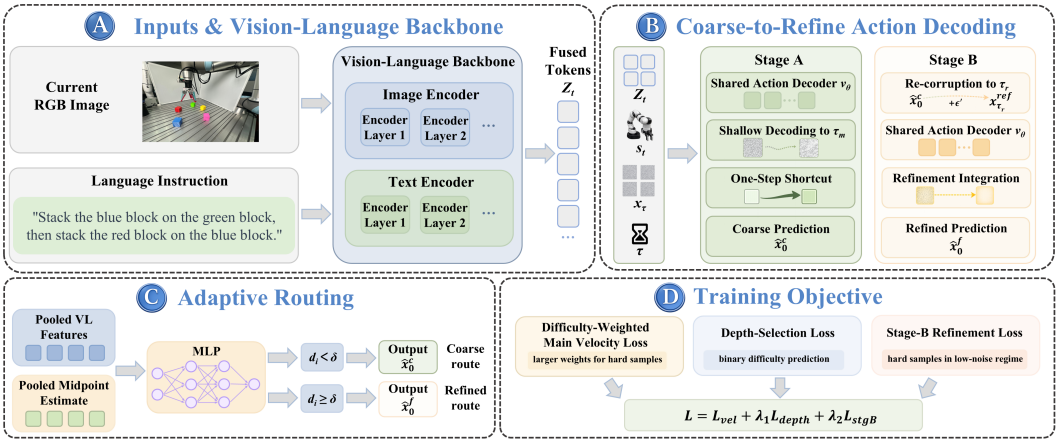

Figure 2: Overview of DepthFlow-VLA. (A) A compact vision-language backbone produces fused tokens Z_t. (B) The action decoder first performs Stage-A coarse decoding and, when selected, Stage-B low-noise refinement starting from the coarse prediction. (C) An adaptive router maps pooled vision-language features and the coarse action state to a routing decision. (D) Training jointly optimizes the main velocity loss, the depth-selection loss, and the Stage-B refinement loss.

produces fused tokens $Z_t = E_\phi(I_t^{1:n}, \ell_t)$, which are concatenated with proprioceptive features, a time embedding, and the noised action chunk, and are then processed by the decoder with standard self-attention. Our method operates purely on the action-decoding side; the backbone architecture and initialization are kept fixed across variants, and unless otherwise stated the backbone is fine-tuned jointly with the action decoder using a reduced learning-rate multiplier.

3.2 Two-Stage Action Decoding

Stage A: Initial Decoding. Starting from Gaussian noise $x_1 \sim \mathcal{N}(0, I)$, we integrate the velocity field from $\tau = 1$ to an intermediate time $\tau_m \in (0, 1)$ with K_c Euler steps:

$$x_{\tau - \Delta\tau} = x_\tau + (-\Delta\tau)\, v_\theta(x_\tau, Z_t, s_t, \tau), \qquad \Delta\tau = \frac{1 - \tau_m}{K_c}. \tag{2}$$

From the resulting state x_{τ_m}, a one-step ODE shortcut yields the Stage-A output

$$\hat{x}_0^{\mathrm{c}} = x_{\tau_m} + (-\tau_m)\, v_\theta(x_{\tau_m}, Z_t, s_t, \tau_m). \tag{3}$$

Geometrically, this amounts to a first-order extrapolation from x_{τ_m} toward the clean action manifold using the velocity at τ_m; we treat it as a coarse estimate used for subsequent routing and refinement rather than as an exact solution of the full decoding trajectory.

Stage B: Correction-Oriented Low-Noise Refinement. Instead of continuing Stage-A integration with additional steps, Stage B takes the coarse prediction $\hat{x}_0^{\mathrm{c}}$ as its starting point and performs targeted correction in a dedicated low-noise regime closer to the clean action. We first re-corrupt $\hat{x}_0^{\mathrm{c}}$ with a small amount of noise at level $\tau_r \in (0, \tau_m)$,

$$x_{\tau_r}^{\mathrm{ref}} = (1 - \tau_r)\, \hat{x}_0^{\mathrm{c}} + \tau_r\, \epsilon', \qquad \epsilon' \sim \mathcal{N}(0, I), \tag{4}$$

and then denoise back to $\tau = 0$ with K_f Euler steps:

$$x_{\tau - \delta\tau} = x_\tau + (-\delta\tau)\, v_\theta(x_\tau, Z_t, s_t, \tau), \qquad \delta\tau = \frac{\tau_r}{K_f}. \tag{5}$$

The resulting output $\hat{x}_0^{\mathrm{f}}$ is the refined prediction. Stage B thus plays a distinct functional role—correcting the Stage-A output rather than extending its integration trajectory—while sharing all parameters with Stage A and adding no decoder-side capacity. During training, we replay Stage A for difficult samples, re-corrupt the resulting coarse prediction, and supervise the shared decoder along the corresponding correction path, as detailed in Appendix A.3. This exposes Stage B to the coarse-conditioned states it must correct at inference. Appendix D.1 reports lower success when this targeted refinement objective is removed.

3.3 Depth Selection Signal

Not every sample benefits equally from refinement. We therefore attach a lightweight depth-selection head aligned with the Stage-A split point τ_m. During training, a binary difficulty label is derived from a residual-based proxy at the Stage-A midpoint, and a small MLP is trained to predict this label from pooled vision-language features and midpoint action estimates. Using the batch median as the threshold provides an approximately balanced, batch-relative supervision signal for refinement priority. Appendix A.1 gives the full construction.

At inference, we convert the predicted logit into a routing score $d_i = \sigma(\hat{y}_i)$ and apply a threshold δ:

$$\hat{A}_t^i = \begin{cases} \hat{x}_0^{\mathrm{c},i}, & \text{if } d_i < \delta, \\ \hat{x}_0^{\mathrm{f},i}, & \text{otherwise.} \end{cases} \tag{6}$$

Chunks with scores below δ retain the Stage-A output; the remaining chunks receive Stage-B refinement. We use $\delta = 0.7$ as the default operating point and report a threshold sweep in Appendix D.2 to characterize the accuracy–compute trade-off without retraining. Because the routing score is trained as a relative priority signal, redeployment to a substantially different task distribution may require threshold recalibration.

3.4 Training Objective

The overall training objective combines three terms,

$$\mathcal{L} = \mathcal{L}_{\mathrm{vel}}(\theta) + \lambda_1 \mathcal{L}_{\mathrm{depth}}(\psi) + \lambda_2 \mathcal{L}_{\mathrm{stgB}}(\theta), \tag{7}$$

where $\mathcal{L}_{\mathrm{vel}}$ is a difficulty-weighted main velocity loss that up-weights hard samples, $\mathcal{L}_{\mathrm{depth}}$ is a binary cross-entropy loss on the routing head, and $\mathcal{L}_{\mathrm{stgB}}$ supervises the shared decoder on correction paths initialized from re-corrupted Stage-A coarse predictions; λ_1 and λ_2 are balancing coefficients. Full definitions are deferred to Appendices A.2 and A.3, with the complete training and inference procedures summarized in Appendix A.4.

4 Experiments

4.1 Experimental Setup

4.1.1 Simulation Benchmarks

LIBERO (Liu et al., 2023). We evaluate on the four standard LIBERO suites—Spatial, Object, Goal, and Long—each containing 10 manipulation tasks with 500 expert demonstrations.

SimplerEnv-WidowX (Li et al., 2024). We additionally evaluate on the WidowX track of SimplerEnv, a simulated real-to-sim benchmark based on the BridgeData-V2 (Walke et al., 2023) distribution. We report closed-loop success rates on its four standard tasks, each evaluated over 24 rollouts per seed, following the official SimplerEnv evaluation protocol. The policy is trained separately on BridgeData-V2; results on the Google Robot track of SimplerEnv are reported in Appendix C as additional cross-embodiment evaluation.

4.1.2 Training Setup

Following recent VLA practice (Liang et al., 2025; Wang et al., 2026b), we train DepthFlow-VLA with a unified recipe across simulation experiments. We initialize from a public vision-language checkpoint of our backbone (Marafioti et al., 2025) and train a separate DepthFlow-VLA checkpoint for each benchmark: one jointly fine-tuned on the training splits of all four LIBERO suites for the LIBERO evaluation, and one trained on BridgeData-V2 (Walke et al., 2023) for the SimplerEnv-WidowX evaluation. No additional robotic pretraining data is used. Decoder hyperparameters $(\tau_m, \tau_r, K_c, K_f, \delta)$ are kept identical across the two separately trained checkpoints to evaluate the same decoding configuration across embodiments. Optimization settings and compute resources are reported in Appendices B.5 and B.6. The cross-paper tables use published results under each work's reported training and evaluation settings. Controlled comparisons use the same backbone or checkpoint, as specified for each experiment.

Table 1: Main comparison on the standard LIBERO benchmark. We report success rates (%) on LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long, together with the average performance. DepthFlow-VLA attains strong results with a compact 0.5B model size.

Model	Size (B)	Spatial	Object	Goal	Long	Avg
Large Models ($\geq$ 4B)						
FlowVLA (Zhong et al., 2025)	8.5	93.2	95.0	91.6	72.6	88.1
UniVLA (Bu et al., 2025)	7	96.5	96.8	95.6	92.0	95.2
OpenVLA (Kim et al., 2024)	7	84.7	88.4	79.2	53.7	76.5
DD-VLA (Liang et al., 2025)	7	97.2	98.6	97.4	92.0	96.3
MemoryVLA (Shi et al., 2025)	7	98.4	98.4	96.4	93.4	96.7
PD-VLA (Song et al., 2025)	7	95.5	96.7	94.9	91.7	94.7
MolmoAct (Lee et al., 2025)	7	87.0	95.4	87.6	77.2	86.6
ThinkAct (Huang et al., 2025)	7	88.3	91.4	87.1	70.9	84.4
CoT-VLA (Zhao et al., 2025)	7	87.5	91.6	87.6	69.0	83.9
WorldVLA (Cen et al., 2025)	7	87.6	96.2	83.4	60.0	81.8
TraceVLA (Zheng et al., 2024)	7	84.6	85.2	75.1	54.1	74.8
FPC-VLA (Yang et al., 2026)	7	86.2	87.0	92.0	82.2	86.9
4D-VLA (Zhang et al., 2025)	4	88.9	95.2	90.9	79.1	88.6
SpatialVLA (Qu et al., 2025)	4	88.2	89.9	78.6	55.5	78.1
Small Models ($<$ 4B)						
π_0 (Black et al., 2024)	3	96.8	98.8	95.8	85.2	94.2
π_0-FAST (Pertsch et al., 2025)	3	96.4	96.8	88.6	60.2	85.5
$\pi_{0.5}$ (Black et al., 2025)	3	**98.8**	98.2	**98.0**	92.4	96.9
NORA-Long (Hung et al., 2025)	3	92.2	95.4	89.4	74.6	87.9
SmolVLA (Shukor et al., 2025)	2.2	93.0	94.0	91.0	77.0	88.8
GR00T-N1 (Bjorck et al., 2025)	2.2	94.4	97.6	93.0	90.6	93.9
FLOWER (Reuss et al., 2025)	1	97.1	96.7	95.6	93.5	95.7
Tiny Models ($<$ 1B)						
VLA-Adapter (Wang et al., 2026b)	0.5	97.8	99.2	97.2	95.0	97.3
VLA-OS (Gao et al., 2025)	0.5	87.0	96.5	92.7	66.0	85.6
UniAct (Zheng et al., 2025)	0.5	77.0	87.0	77.0	70.0	77.8
DepthFlow-VLA	**0.5**	98.4	**99.4**	97.8	**97.6**	**98.3**

4.2 Simulation Results

4.2.1 Results on LIBERO

Table 1 reports the main cross-paper comparison. DepthFlow-VLA achieves 98.3% average success with a 0.5B backbone, the highest average among the baselines listed in the table. Its strongest relative result is on LIBERO-Long, where it reaches 97.6%, 2.6 points above the strongest listed baseline. Controlled comparisons are reported in Tables 3 and 5. At six decoder evaluations, vanilla FM reaches 94.8%, compared with 95.9% for the Stage-A mode of the trained DepthFlow checkpoint (Appendix B.4). Seed-level counts are provided in Appendix B.2.

4.2.2 Results on SimplerEnv-WidowX

As shown in Table 2, DepthFlow-VLA reaches 80.2% average success with a 0.5B backbone. Within the listed published baselines, this is 15.6 points above FPC-VLA, although FPC-VLA remains stronger on Stack.

Using the same BridgeData-V2-trained checkpoints, Stage-A-only decoding, equal-budget random routing, adaptive routing, and uniform refinement achieve 75.0%, 76.0%, 80.2%, and 81.3%, respectively. Adaptive routing improves over Stage A by 5.2 points and over random routing by 4.2 points. Separately trained vanilla FM reaches 72.2% and 73.6% at 6 and 13 decoder evaluations. Full results are reported in Appendices B.3 and B.4; Google Robot results are provided in Appendix C.

Table 2: Results on SimplerEnv-WidowX. Published baseline values are transcribed from the cited works under their reported evaluation settings; selected averages are recomputed as the simple mean of the four displayed tasks. DepthFlow-VLA is trained on BridgeData-V2 without additional robotic pretraining and is averaged over three seeds. Same-checkpoint inference controls are reported in Appendix B.3.

Method	Spoon	Carrot	Stack	Eggplant	Avg
RT-1-X (O'Neill et al., 2024)	0.0	4.2	0.0	0.0	1.1
Octo-Base (Team et al., 2024)	12.5	8.3	0.0	43.1	16.0
Octo-Small (Team et al., 2024)	47.2	9.7	4.2	56.9	29.5
OpenVLA (Kim et al., 2024)	0.0	0.0	0.0	4.1	1.0
OpenVLA-OFT (Kim et al., 2025)	12.5	4.2	8.3	37.5	15.6
RoboVLM (Li et al., 2026)	29.2	25.0	12.5	58.3	31.3
π_0 (Black et al., 2024)	29.1	0.0	16.7	62.5	27.1
π_0-FAST (Pertsch et al., 2025)	29.1	21.9	10.8	66.6	32.1
DD-VLA (Liang et al., 2025)	29.2	29.2	20.8	70.8	37.5
GR00T-N1 (Bjorck et al., 2025)	62.5	45.8	16.7	20.8	36.5
ThinkAct (Huang et al., 2025)	58.3	37.5	8.7	70.8	43.8
FLOWER (Reuss et al., 2025)	71.0	13.0	8.0	88.0	45.0
FPC-VLA (Yang et al., 2026)	58.3	45.8	**79.2**	75.0	64.6
DepthFlow-VLA	**87.5**	**66.7**	70.8	**95.8**	**80.2**

Table 3: Component ablation on LIBERO. Uniform refinement applies re-corruption and Stage-B denoising to every action chunk, whereas DepthFlow-VLA invokes the same refinement adaptively. Avg. denotes average success, and N_{dec} denotes the expected number of action-decoder evaluations per chunk.

Variant	Spatial	Object	Goal	Long	Avg.	N_{dec}	Stage-B Ratio
Stage A	94.6	97.4	96.4	95.2	95.9	6.0	0%
Uniform refinement	98.6	99.4	98.0	97.8	98.5	13.0	100%
DepthFlow-VLA	98.4	99.4	97.8	97.6	**98.3**	**7.5**	**22%**

4.3 Ablation Studies

4.3.1 Component Ablation

Stage A uses $K_c + 1$ evaluations of v_θ, including the one-step shortcut at τ_m, and Stage B adds K_f evaluations when invoked. For a Stage-B routing ratio ρ, the expected number of action-decoder evaluations per chunk is therefore

$$N_{\mathrm{dec}} = (K_c + 1) + \rho K_f.$$

Table 3 separates restart-style refinement from adaptive routing. Uniform refinement improves average success from 95.9% to 98.5%, establishing the value of the coarse-anchored correction path. DepthFlow-VLA reaches 98.3% while invoking Stage B for only 22% of action chunks. Under the default configuration, this reduces the expected decoder budget from 13.0 to 7.54 evaluations per chunk, a 42.0% reduction relative to uniform refinement. Under the batch-size-one H100 protocol in Appendix B.8, mean end-to-end latency decreases from 59.2 to 42.6 ms, a 28.0% reduction.

The aggregate success gap between adaptive and uniform refinement is small, but rollout success does not measure chunk-level routing accuracy. We evaluate routing quality directly against paired offline benefit below.

4.3.2 Routing Quality Against Paired Refinement Benefit

Score separation alone does not establish whether the router selects chunks for which Stage B is useful. We therefore evaluate the inference-time score on 200 held-out LIBERO action chunks, with 50 chunks from each suite and at most one chunk from each episode. For each chunk, Stage-B error is averaged over five independent re-corruption draws.

Let

$$E(\hat{x}, x) = \frac{1}{H} \sum_{h=1}^{H} \|\hat{x}_h - x_h\|_2 , \qquad \Delta_i = E(\hat{x}_i^{\mathrm{c}}, x_i) - E(\hat{x}_i^{\mathrm{f}}, x_i). \tag{8}$$

A chunk is labeled as meaningfully improved when Stage B reduces its action error by at least 5% relative to Stage A.

At $\delta = 0.7$, 30 of the 44 routed chunks exhibit meaningful Stage-B benefit, compared with 19 of the 156 skipped chunks. This corresponds to 68.2% precision and 61.2% recall, indicating useful but incomplete ranking of refinement benefit. We therefore treat the router output as a refinement-priority score rather than a calibrated probability of closed-loop improvement. The oracle definition and routing-score distributions are provided in Appendix D.3.

Table 4: Router quality against paired offline Stage-B benefit on 200 held-out chunks. OGR denotes oracle-gain recovery at the same 44-chunk refinement budget.

Metric	Result
Spearman correlation	0.36
AUROC	0.75
AUPRC / base rate	0.54 / 0.245
Precision	68.2%
Recall / FNR	61.2% / 38.8%
Offline OGR	73.8%

4.3.3 Compute-Controlled Analysis

To separate the effect of additional decoder evaluations from the effect of the refinement state, we compare DepthFlow-VLA with deeper single-stage integration, routed continuation, direct local correction without re-corruption, and uniform restart-style refinement. The routed continuation and no-re-corruption controls use the same router and the same 22% Stage-B budget as DepthFlow-VLA.

Table 5: Compute-controlled and Stage-B state comparison on LIBERO. All routed variants use $(K_c, K_f) = (5, 7)$ and the same 22% refinement budget. Avg. v_θ Evals denotes the expected logical number of action-decoder evaluations per action chunk. All inference-only variants use the same fully trained DepthFlow-VLA checkpoint and differ only in their decoding procedure.

Variant	Avg. v_θ Evals	Spatial	Object	Goal	Long
Single-stage (K=8)	8.0	95.8	98.2	96.8	95.6
Single-stage (K=12)	12.0	96.2	98.4	97.2	96.2
Routed continuation	7.5	96.8	98.6	97.4	96.6
Local correction w/o re-corruption	7.5	96.6	98.4	97.2	96.4
DepthFlow-VLA	**7.5**	98.4	99.4	97.8	97.6
Uniform refinement	13.0	98.6	99.4	98.0	97.8

At 7.54 expected decoder evaluations per chunk, DepthFlow-VLA outperforms both routed continuation and direct correction without re-corruption under the same routing budget. It also outperforms the 12-evaluation single-stage control on every suite while using approximately 37% fewer action-decoder evaluations. Under this trained checkpoint, the results favor coarse-anchored re-corruption over the tested alternatives, indicating that additional integration depth alone does not explain the gain. Uniform restart-style refinement serves as the non-adaptive full-budget restart reference.

Routing threshold. Appendix D.2 varies δ using the same three trained checkpoints. With $\delta = 0.9$, Stage B is invoked for 7% of chunks, yielding 97.4% success with 6.5 expected decoder evaluations. Success rises to 98.0% with 7.0 evaluations at $\delta = 0.8$, while $\delta = 0.5$ reaches 98.5% with 8.5 evaluations and a 35% routing ratio. The two lower thresholds tested yield the same rounded success rate. These operating points adjust the accuracy–compute trade-off without retraining.

Latency breakdown. The H100 measurements in Appendix B.8 report an 18.4 ms backbone pass, 3.1 ms per decoder evaluation, and 0.4 ms of aggregate routing and re-corruption overhead under the default adaptive setting. The backbone runs once per chunk in all modes, so its cost remains when Stage B is skipped. This common cost explains why the end-to-end latency reduction is smaller than

the reduction in decoder evaluations. Each measurement uses 50 warm-up iterations followed by 500 timed iterations. Timing excludes environment stepping and data loading.

4.3.4 Training Objective Ablation

Removing hard-sample weighting has a modest effect, whereas removing the depth-selection or Stage-B refinement objective causes larger drops, particularly on LIBERO-Long. Full results are reported in Appendix D.1.

4.4 Real-World Experiments

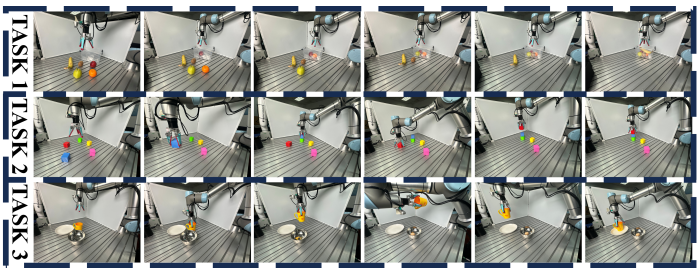

Figure 3: Representative real-world rollouts of DepthFlow-VLA on three long-horizon tabletop tasks: five-fruit collection, two-step color block stacking, and cup pouring with placement.

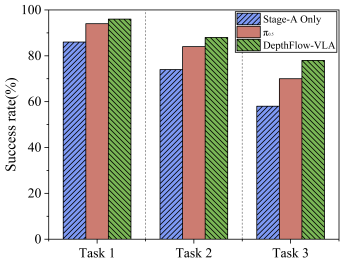

Figure 4: Real-world success rates (%) on the three long-horizon tabletop tasks.

We evaluate DepthFlow-VLA on a real-world tabletop platform with a UR5e arm, a three-finger gripper, and a fixed RealSense D435i camera. The policy takes RGB observations, a language instruction, and proprioceptive state as input. As shown in Fig. 3, we consider three long-horizon tasks: five-fruit collection, where the robot places five fruits into a basket; two-step color block stacking, where it stacks the blue block on the green block and then the red block on the blue block; and cup pouring with placement, where it pours small balls from a cup into a bowl and then places the cup onto a plate. DepthFlow-VLA and its Stage-A-only variant are initialized from the same LIBERO-trained DepthFlow-VLA checkpoint. For $\pi_{0.5}$, we use our independently trained LIBERO checkpoint. Each method is then adapted separately to each target task using the same 100 demonstrations, action normalization procedure, optimization budget, and evaluation reset distribution. Object poses are randomized, and we report success rates over 50 trials per task; a trial succeeds only if the full sequence is completed.

DepthFlow-VLA achieves 96%, 88%, and 78% success on fruit collection, block stacking, and pouring, respectively, compared with 86%, 74%, and 58% for Stage-A-only decoding and 94%, 84%, and 70% for $\pi_{0.5}$. The gaps over Stage A are 10, 14, and 20 percentage points, with the largest difference on pouring. However, the 95% Clopper–Pearson intervals over 50 trials overlap across methods (Appendix D.8), so the differences should be viewed as directional. The results indicate that the same routing mechanism can be transferred to the UR5e setting without embodiment-specific redesign, although broader real-world validation is still needed.

5 Conclusion

We introduced DepthFlow-VLA, a compact VLA decoder that selectively routes coarse Stage-A actions to restart-style low-noise refinement. Controlled comparisons examine the training objectives and the roles of refinement, routing, and integration depth. At the default operating point, DepthFlow-VLA approaches uniform-refinement accuracy with 42.0% fewer decoder evaluations and 28.0% lower mean end-to-end latency on a single H100. Results on LIBERO, SimplerEnv, and UR5e support routed coarse-to-refine decoding as a practical efficiency mechanism for compact VLA policies.

AI USE STATEMENT

Generative AI tools were used solely for language editing and proofreading. All AI-assisted revisions were reviewed by the authors. The authors take full responsibility for the final content of the paper.

REPRODUCIBILITY STATEMENT

Appendices A.1–A.3 define the routing supervision and training objectives, and Appendix A.4 provides the training and inference algorithms. Simulation evaluation protocols and seed-level success counts are reported in Appendices B.1 and B.2. Appendices B.3 and B.4 describe the same-checkpoint inference ablation on SimplerEnv-WidowX and the vanilla flow-matching control, respectively. Training hyperparameters, compute resources, decoder-compute accounting, and latency measurement procedures are given in Appendices B.5–B.8. Appendix C describes the additional SimplerEnv Google Robot (Fractal) evaluation. Appendices D.1–D.6 provide additional ablations, routing analyses, paired-benefit diagnostic details, and decoding-sensitivity studies. Appendices D.7 and D.8 describe the real-world setup, evaluation protocol, and results.

REFERENCES

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

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

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

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

Jiayi Chen, Wenxuan Song, Shuai Chen, Jingbo Wang, Zhijun Li, and Haoang Li. Dfm-vla: Iterative action refinement for robot manipulation via discrete flow matching. *arXiv preprint arXiv:2603.26320*, 2026.

Muhayy Ud Din, Waseem Akram, Lyes Saad Saoud, Jan Rosell, and Irfan Hussain. Vision language action models in robotic manipulation: A systematic review. *arXiv preprint arXiv:2507.10672*, 2025.

Shaoqi Dong, Chaoyou Fu, Haihan Gao, Yi-Fan Zhang, Chi Yan, Chu Wu, Xiaoyu Liu, Yunhang Shen, Jing Huo, Deqiang Jiang, et al. Vita-vla: Efficiently teaching vision-language models to act via action expert distillation. *arXiv preprint arXiv:2510.09607*, 2025.

Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, Yequan Wang, and Zhongyuan Wang. Not all layers of llms are necessary during inference. *arXiv preprint arXiv:2403.02181*, 2024.

Chongkai Gao, Zixuan Liu, Zhenghao Chi, Junshan Huang, Xin Fei, Yiwen Hou, Yuxuan Zhang, Yudi Lin, Zhirui Fang, Zeyu Jiang, et al. Vla-os: Structuring and dissecting planning representations and paradigms in vision-language-action models. *arXiv preprint arXiv:2506.17561*, 2025.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Chi-Pin Huang, Yueh-Hua Wu, Min-Hung Chen, Yu-Chiang Frank Wang, and Fu-En Yang. Thinkact: Vision-language-action reasoning via reinforced visual latent planning. *arXiv preprint arXiv:2507.16815*, 2025.

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

Gagan Jain, Nidhi Hegde, Aditya Kusupati, Arsha Nagrani, Shyamal Buch, Prateek Jain, Anurag Arnab, and Sujoy Paul. Mixture of nested experts: Adaptive processing of visual tokens. *Advances in Neural Information Processing Systems*, 37:58480–58497, 2024.

Yuhua Jiang, Shuang Cheng, Yan Ding, Feifei Gao, and Biqing Qi. Asyncvla: Asynchronous flow matching for vision-language-action models. *arXiv preprint arXiv:2511.14148*, 2025.

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

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

Jason Lee, Jiafei Duan, Haoquan Fang, Yuquan Deng, Shuo Liu, Boyang Li, Bohan Fang, Jieyu Zhang, Yi Ru Wang, Sangho Lee, et al. Molmoact: Action reasoning models that can reason in space. *arXiv preprint arXiv:2508.07917*, 2025.

Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, et al. Vision-language foundation models as effective robot imitators. *arXiv preprint arXiv:2311.01378*, 2023.

Xinghang Li, Peiyan Li, Long Qian, Minghuan Liu, Dong Wang, Jirong Liu, Bingyi Kang, Xiao Ma, Xinlong Wang, Di Guo, et al. What matters in building vision–language–action models for generalist robots. *Nature Machine Intelligence*, pp. 1–15, 2026.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024.

Zhixuan Liang, Yizhuo Li, Tianshuo Yang, Chengyue Wu, Sitong Mao, Tian Nian, Liuao Pei, Shunbo Zhou, Xiaokang Yang, Jiangmiao Pang, et al. Discrete diffusion vla: Bringing discrete diffusion to action decoding in vision-language-action policies. *arXiv preprint arXiv:2508.20072*, 2025.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Jiaming Liu, Mengzhen Liu, Zhenyu Wang, Pengju An, Xiaoqi Li, Kaichen Zhou, Senqiao Yang, Renrui Zhang, Yandong Guo, and Shanghang Zhang. Robomamba: Efficient vision-language-action model for robotic reasoning and manipulation. *Advances in Neural Information Processing Systems*, 37:40085–40110, 2024.

Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, et al. Smolvlm: Redefining small and efficient multimodal models. *arXiv preprint arXiv:2504.05299*, 2025.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903. IEEE, 2024.

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

Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, et al. Spatialvla: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025.

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

Ranjan Sapkota, Yang Cao, Konstantinos I Roumeliotis, and Manoj Karkee. Vision-language-action models: Concepts, progress, applications and challenges. *arXiv preprint arXiv:2505.04769*, 2025.

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. *arXiv preprint arXiv:2508.19236*, 2025.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

Wenxuan Song, Jiayi Chen, Pengxiang Ding, Han Zhao, Wei Zhao, Zhide Zhong, Zongyuan Ge, Zhijun Li, Donglin Wang, Jun Ma, et al. Pd-vla: Accelerating vision-language-action model integrated with action chunking via parallel decoding. *arXiv preprint arXiv:2503.02310*, 2025.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. 2023.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pp. 1723–1736. PMLR, 2023.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Ramana Rao Kompella, and Gaowen Liu. Vla knows its limits. *arXiv preprint arXiv:2602.21445*, 2026a.

Yihao Wang, Pengxiang Ding, Lingxiao Li, Can Cui, Zirui Ge, Xinyang Tong, Wenxuan Song, Han Zhao, Wei Zhao, Pengxu Hou, et al. Vla-adapter: An effective paradigm for tiny-scale vision-language-action model. In *Proceedings of the AAAI conference on artificial intelligence*, volume 40, pp. 18638–18646, 2026b.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.

Yilun Xu, Mingyang Deng, Xiang Cheng, Yonglong Tian, Ziming Liu, and Tommi Jaakkola. Restart sampling for improving generative processes. *Advances in Neural Information Processing Systems*, 36:76806–76838, 2023.

Yifan Yang, Zhixiang Duan, Tianshi Xie, Fuyu Cao, Pinxi Shen, Peili Song, Chenyang Zhao, Piaopiao Jin, Guokang Sun, Shaoqing Xu, et al. Fpc-vla: A vision-language-action framework with a supervisor for failure prediction and correction. *Expert Systems with Applications*, pp. 131742, 2026.

Zhaoshu Yu, Bo Wang, Pengpeng Zeng, Haonan Zhang, Ji Zhang, Zheng Wang, Lianli Gao, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. A survey on efficient vision-language-action models. *arXiv preprint arXiv:2510.24795*, 2025.

Zihao Zeng, Yibo Miao, Hongcheng Gao, Hao Zhang, and Zhijie Deng. Adamoe: Token-adaptive routing with null experts for mixture-of-experts language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 6223–6235, 2024.

Jiahui Zhang, Yurui Chen, Yueming Xu, Ze Huang, Yanpeng Zhou, Yu-Jie Yuan, Xinyue Cai, Guowei Huang, Xingyue Quan, Hang Xu, et al. 4d-vla: Spatiotemporal vision-language-action pretraining with cross-scene calibration. *arXiv preprint arXiv:2506.22242*, 2025.

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

Jinliang Zheng, Jianxiong Li, Dongxiu Liu, Yinan Zheng, Zhihao Wang, Zhonghong Ou, Yu Liu, Jingjing Liu, Ya-Qin Zhang, and Xianyuan Zhan. Universal actions for enhanced embodied foundation models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 22508–22519, 2025.

Ruijie Zheng, Yongyuan Liang, Shuaiyi Huang, Jianfeng Gao, Hal Daumé III, Andrey Kolobov, Furong Huang, and Jianwei Yang. Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. *arXiv preprint arXiv:2412.10345*, 2024.

Zhide Zhong, Haodong Yan, Junfeng Li, Xiangchen Liu, Xin Gong, Tianran Zhang, Wenxuan Song, Jiayi Chen, Xinhu Zheng, Hesheng Wang, et al. Flowvla: Visual chain of thought-based motion reasoning for vision-language-action models. *arXiv preprint arXiv:2508.18269*, 2025.

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

APPENDIX

A ADDITIONAL METHOD DETAILS

A.1 STAGE-A-ALIGNED DEPTH SUPERVISION

We construct depth supervision from a training-time proxy aligned with the Stage-A split point τ_m. For each sample i, we draw a noisy action at τ_m,

$$x_{\tau_m}^{(i)} = \tau_m \epsilon^{(i)} + (1 - \tau_m) x^{(i)}, \qquad \epsilon^{(i)} \sim \mathcal{N}(0, I),$$

and obtain a one-step estimate

$$\hat{x}_{0,\mathrm{mid}}^{(i)} = x_{\tau_m}^{(i)} + (-\tau_m)\, v_\theta(x_{\tau_m}^{(i)}, Z_t^{(i)}, s_t^{(i)}, \tau_m). \tag{9}$$

The midpoint estimate is detached and used both as an input to the routing head and to construct a training-time difficulty proxy. We compute its reconstruction residual by averaging the action-dimensional error over the horizon:

$$r_i = \frac{1}{H} \sum_{h=1}^{H} \left\| \hat{x}_{0,\mathrm{mid},h}^{(i)} - x_h^{(i)} \right\|_2 . \tag{10}$$

The residual proxy is converted into a binary difficulty label using the batch median,

$$y_i = \mathbf{1}[r_i \geq \mathrm{median}(\{r_j\})] \tag{11}$$

so that $y_i = 1$ marks samples whose midpoint residual is at least the batch median. These batch-relative labels supervise the routing head, determine the main-loss weights, and select samples for Stage-B training. The midpoint estimate, residual proxy, and labels are detached during optimization. The inference-time routing threshold is set by δ.

The depth-selection head is a two-layer MLP whose inputs are detached during training,

$$\hat{y}_i = g_\psi\big([\bar{Z}_t^i \,\|\, \bar{\hat{x}}_{0,\mathrm{mid}}^{\,i}]\big), \tag{12}$$

where $\bar{Z}_t^i$ is the mean-pooled vision-language feature and $\bar{\hat{x}}_{0,\mathrm{mid}}^{\,i}$ is the pooled midpoint estimate. It is trained with binary cross-entropy on logits,

$$\mathcal{L}_{\mathrm{depth}}(\psi) = \mathbb{E}\big[-y_i \log \sigma(\hat{y}_i) - (1 - y_i) \log\big(1 - \sigma(\hat{y}_i)\big)\big], \tag{13}$$

and yields the inference-time difficulty score $d_i = \sigma(\hat{y}_i)$.

A.2 DIFFICULTY-WEIGHTED MAIN VELOCITY LOSS

To emphasize difficult samples, we up-weight those with $y_i = 1$ by

$$w_i = 1 + \alpha y_i, \tag{14}$$

with $\alpha = 0.5$ in all experiments. The resulting weighted main velocity loss is

$$\mathcal{L}_{\mathrm{vel}}(\theta) = \frac{1}{B} \sum_{i=1}^{B} w_i \left\| v_\theta(x_{\tau^{(i)}}^{(i)}, Z_t^{(i)}, s_t^{(i)}, \tau^{(i)}) - u_{\tau^{(i)}}^{(i)} \right\|_2^2, \tag{15}$$

with target velocity $u_{\tau^{(i)}}^{(i)} = \epsilon^{(i)} - x^{(i)}$.

A.3 STAGE-B TARGETED TRAINING

For each difficult sample $i \in \mathcal{H}$, where $\mathcal{H} = \{i : y_i = 1\}$, we first replay the Stage-A decoding procedure. Specifically, we draw $\epsilon_{\mathrm{c}}^{(i)} \sim \mathcal{N}(0, I)$ and compute

$$\hat{x}_{0,\mathrm{rep}}^{\mathrm{c},(i)} = \mathrm{sg}\Big[\mathrm{StageA}_\theta\big(\epsilon_{\mathrm{c}}^{(i)}, Z_t^{(i)}, s_t^{(i)}; \tau_m, K_c\big)\Big], \tag{16}$$

where StageA_θ denotes the K_c-step integration and one-step shortcut in Eqs. 2 and 3, and $\mathrm{sg}[\cdot]$ denotes stop-gradient.

We then apply the same re-corruption rule used by Stage B:

$$z_{\tau_r}^{(i)} = (1-\tau_r)\,\hat{x}_{0,\mathrm{rep}}^{\mathrm{c},(i)} + \tau_r\,\epsilon_{\mathrm{r}}^{(i)}, \qquad \epsilon_{\mathrm{r}}^{(i)} \sim \mathcal{N}(0, I). \tag{17}$$

To define supervision over the Stage-B interval, we sample $\tau_s^{(i)} \sim \mathrm{Uniform}(0, \tau_r)$ and construct the linear correction path

$$z_{\tau_s^{(i)}}^{(i)} = x^{(i)} + \frac{\tau_s^{(i)}}{\tau_r}\left(z_{\tau_r}^{(i)} - x^{(i)}\right), \tag{18}$$

with velocity target

$$u_{\mathrm{rep}}^{(i)} = \frac{z_{\tau_r}^{(i)} - x^{(i)}}{\tau_r}. \tag{19}$$

The Stage-B refinement loss is therefore

$$\mathcal{L}_{\mathrm{stgB}}(\theta) = \frac{1}{|\mathcal{H}|}\sum_{i\in\mathcal{H}}\left\| v_\theta(z_{\tau_s^{(i)}}^{(i)}, Z_t^{(i)}, s_t^{(i)}, \tau_s^{(i)}) - u_{\mathrm{rep}}^{(i)} \right\|_2^2. \tag{20}$$

The path reaches the replayed Stage-B initialization at $\tau_s^{(i)} = \tau_r$ and the demonstrated action at $\tau_s^{(i)} = 0$. When the Stage-A prediction equals the demonstrated action, the construction reduces to the standard flow-matching path and velocity target.

A.4 Algorithmic Summary

We summarize the training and inference procedures of DepthFlow-VLA in Algorithms 1 and 2. Training jointly optimizes a flow-matching velocity loss, a depth-selection loss, and a Stage-B refinement loss constructed from replayed coarse states. At inference time, the model produces a Stage-A coarse prediction and invokes Stage-B refinement only for samples whose routing score meets or exceeds the threshold.

For brevity, we denote the per-sample flow-matching velocity loss at noise level τ by

$$\ell_{\mathrm{FM}}(i, \tau, \epsilon) = \left\| v_\theta(\tau\epsilon + (1-\tau)x_i,\ Z_i, s_i, \tau) - (\epsilon - x_i) \right\|_2^2,$$

and write $\bar{\cdot}$ for mean pooling over tokens (for Z) or over the action horizon (for $\hat{x}_0$). Quantities under $\mathrm{sg}[\cdot]$ are treated as constants by the optimizer. Unless stated otherwise, the backbone ϕ is updated jointly with θ, ψ; gradients from $\mathcal{L}_{\mathrm{depth}}$ are blocked at $\bar{Z}_i$ via $\mathrm{sg}[\cdot]$ so that the routing head does not reshape the visual representation.

Algorithm 1 DepthFlow-VLA Training

Require: Dataset $\mathcal{D}$; backbone E_ϕ, decoder v_θ, depth head g_ψ; hyperparameters $\tau_m, \tau_r, K_c, \lambda_1, \lambda_2, \alpha$

for each training iteration **do**
  Sample $\{(o_i, x_i)\}_{i=1}^{B} \sim \mathcal{D}$ with $o_i = \{I_i^{1:n}, \ell_i, s_i\}$; encode $Z_i \leftarrow E_\phi(I_i^{1:n}, \ell_i)$
  {Difficulty estimation (stop-gradient)}
  Sample $\epsilon_i^m \sim \mathcal{N}(0, I)$; set $x_{\tau_m}^{(i)} = \tau_m \epsilon_i^m + (1 - \tau_m) x_i$
  $\hat{x}_{0,\mathrm{mid}}^{(i)} = \mathrm{sg}\Big[x_{\tau_m}^{(i)} - \tau_m \, v_\theta(x_{\tau_m}^{(i)}, Z_i, s_i, \tau_m)\Big]$
  $r_i = \frac{1}{H} \sum_{h=1}^{H} \|\hat{x}_{0,\mathrm{mid},h}^{(i)} - x_{i,h}\|_2; \quad y_i = \mathbf{1}[r_i \geq \mathrm{median}(\{r_j\})]$
  {Routing head and depth-selection loss}
  $\hat{y}_i = g_\psi\Big([\mathrm{sg}[\bar{Z}_i] \,\|\, \bar{\hat{x}}_{0,\mathrm{mid}}^{(i)}]\Big); \quad \mathcal{L}_{\mathrm{depth}} = \frac{1}{B} \sum_i \mathrm{BCEWithLogits}(\hat{y}_i, y_i)$
  {Difficulty-weighted main velocity loss}
  Sample $b_i \sim \mathrm{Beta}(1.5, 1)$, $\tau_i = 0.999\, b_i + 0.001$, $\epsilon_i \sim \mathcal{N}(0, I)$
  $\mathcal{L}_{\mathrm{vel}} = \frac{1}{B} \sum_i (1 + \alpha y_i) \, \ell_{\mathrm{FM}}(i, \tau_i, \epsilon_i)$
  {Stage-B refinement loss on replayed coarse states}
  $\mathcal{H} \leftarrow \{i : y_i = 1\}$
  if $|\mathcal{H}| > 0$ **then**
    Sample $\epsilon_i^{\mathrm{c}} \sim \mathcal{N}(0, I)$ for $i \in \mathcal{H}$
    $\hat{x}_{0,\mathrm{rep}}^{\mathrm{c},(i)} \leftarrow \mathrm{sg}[\mathrm{StageA}_\theta(\epsilon_i^{\mathrm{c}}, Z_i, s_i; \tau_m, K_c)]$
    Sample $\epsilon_i^{\mathrm{r}} \sim \mathcal{N}(0, I)$ for $i \in \mathcal{H}$
    $z_{\tau_r}^{(i)} \leftarrow (1 - \tau_r) \hat{x}_{0,\mathrm{rep}}^{\mathrm{c},(i)} + \tau_r \epsilon_i^{\mathrm{r}}$
    Sample $\tau_i^s \sim \mathrm{Uniform}(0, \tau_r)$ for $i \in \mathcal{H}$
    $z_{\tau_i^s}^{(i)} \leftarrow x_i + \frac{\tau_i^s}{\tau_r}(z_{\tau_r}^{(i)} - x_i)$
    $u_{\mathrm{rep}}^{(i)} \leftarrow (z_{\tau_r}^{(i)} - x_i)/\tau_r$
    $\mathcal{L}_{\mathrm{stgB}} = \frac{1}{|\mathcal{H}|} \sum_{i \in \mathcal{H}} \Big\| v_\theta(z_{\tau_i^s}^{(i)}, Z_i, s_i, \tau_i^s) - u_{\mathrm{rep}}^{(i)} \Big\|_2^2$
  else
    $\mathcal{L}_{\mathrm{stgB}} \leftarrow 0$
  end if
  Update ϕ, θ, ψ on $\mathcal{L} = \mathcal{L}_{\mathrm{vel}} + \lambda_1 \mathcal{L}_{\mathrm{depth}} + \lambda_2 \mathcal{L}_{\mathrm{stgB}}$
end for

Algorithm 2 DepthFlow-VLA Routed Inference

Require: Observation $o_t = \{I_t^{1:n}, \ell_t, s_t\}$; backbone E_ϕ, decoder v_θ, depth head g_ψ; τ_m, τ_r, Stage-A/B steps K_c, K_f, threshold δ

Encode $Z_t \leftarrow E_\phi(I_t^{1:n}, \ell_t)$; sample $x_1 \sim \mathcal{N}(0, I)$
{Stage A: coarse decoding}
$x_{\tau_m} \leftarrow \textsc{EulerIntegrate}(x_1, 1, \tau_m, K_c)$
$\hat{x}_0^{\mathrm{c}} = x_{\tau_m} - \tau_m \, v_\theta(x_{\tau_m}, Z_t, s_t, \tau_m)$
{Adaptive routing}
$d = \sigma\big(g_\psi\big([\bar{Z}_t \,\|\, \hat{x}_0^{\mathrm{c}}]\big)\big)$
if $d < \delta$ **then**
  return denormalized $\hat{x}_0^{\mathrm{c}}$
end if
{Stage B: low-noise refinement}
Sample $\epsilon' \sim \mathcal{N}(0, I)$; set $x_{\tau_r}^{\mathrm{ref}} = (1 - \tau_r)\hat{x}_0^{\mathrm{c}} + \tau_r \epsilon'$
$\hat{x}_0^{\mathrm{f}} \leftarrow \textsc{EulerIntegrate}(x_{\tau_r}^{\mathrm{ref}}, \tau_r, 0, K_f)$
return denormalized $\hat{x}_0^{\mathrm{f}}$

Subroutine $\textsc{EulerIntegrate}(x_{\mathrm{start}}, \tau_{\mathrm{start}}, \tau_{\mathrm{end}}, K)$:
  $\Delta\tau \leftarrow (\tau_{\mathrm{start}} - \tau_{\mathrm{end}})/K$; $x \leftarrow x_{\mathrm{start}}$; $\tau \leftarrow \tau_{\mathrm{start}}$
  for $k = 1$ **to** K **do** $x \leftarrow x - \Delta\tau \, v_\theta(x, Z_t, s_t, \tau)$; $\tau \leftarrow \tau - \Delta\tau$
  return x

Stage A requires K_c+1 evaluations of v_θ, including the one-step shortcut at τ_m, and Stage B adds K_f evaluations when invoked. We report the expected logical number of decoder evaluations per action chunk as a hardware-independent compute measure. At batch size one, each chunk executes either $K_c + 1$ or $K_c + 1 + K_f$ decoder calls; the reported budget averages over chunks. For larger batches, realized wall-clock cost depends on whether routed samples are compacted before Stage B. Appendix B.7 gives the full accounting, and Appendix D.2 reports the accuracy–compute curve obtained by varying δ without retraining.

B Additional Implementation Details

B.1 Simulation Evaluation Protocol

For the standard LIBERO benchmark, we evaluate on LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long. Each suite contains 10 tasks, and we run 50 evaluation rollouts per task, resulting in 500 trials per suite. We report the aggregate trial-level success rate over all trials in a suite:

$$\mathrm{SR}_{\mathrm{suite}} = \frac{1}{10 \times 50} \sum_{m=1}^{10} \sum_{n=1}^{50} \mathbf{1}[\mathrm{success}_{m,n}] \,.$$

A rollout is counted as successful if the LIBERO task-specific BDDL goal predicate is satisfied before the time limit; otherwise it is counted as a failure. We use rollout limits of 800 environment steps for Spatial, Object, and Goal, and 900 environment steps for Long, with 10 initial stabilization steps.

All main LIBERO results of DepthFlow-VLA are reported as the mean over three random seeds. For each seed, we evaluate 50 rollouts per task, resulting in 500 trials per LIBERO suite. Unless otherwise stated, we evaluate the final checkpoint after the full training schedule in Table 10, rather than selecting a checkpoint based on validation or test performance. All ablation variants use the same per-task rollout protocol unless otherwise specified. The corresponding seed-level raw counts are reported in Appendix B.2.

For SimplerEnv-WidowX, we train a separate DepthFlow-VLA checkpoint on BridgeData-V2 and evaluate the four standard tasks (Spoon, Carrot, Stack, Eggplant) following the official SimplerEnv evaluation protocol, with 24 closed-loop rollouts per task per seed. Results are averaged over three seeds, and the per-method average reported in Table 2 is computed as the simple mean of the four displayed task scores.

B.2 Seed-Level Simulation Counts

Table 6 reports the seed-level closed-loop success counts of DepthFlow-VLA on the standard LIBERO benchmark. For each suite, we evaluate 10 tasks with 50 rollouts per task, yielding 500 trials per seed. The reported success rate is the mean and sample standard deviation over three seeds. This table provides the raw counts underlying the aggregate DepthFlow-VLA results in Table 1.

Table 6: Seed-level closed-loop success counts of DepthFlow-VLA on LIBERO. Each suite contains 10 tasks with 50 rollouts per task, yielding 500 trials per seed. Success Rate is reported as mean $\pm$ sample standard deviation over three seeds.

Benchmark / Task	**Seed 1**	**Seed 2**	**Seed 3**	**Success Rate (%)**
LIBERO-Spatial	492/500	491/500	493/500	98.4 ± 0.2
LIBERO-Object	497/500	495/500	499/500	99.4 ± 0.4
LIBERO-Goal	490/500	487/500	490/500	97.8 ± 0.3
LIBERO-Long	485/500	489/500	490/500	97.6 ± 0.5

Table 7 reports the seed-level closed-loop success counts of DepthFlow-VLA on SimplerEnv-WidowX. For each task, we evaluate 24 rollouts per seed and report the mean and sample standard deviation over three seeds. The average reported in Table 2 is computed as the simple mean of the four task-level success rates.

Table 7: Seed-level closed-loop success counts of DepthFlow-VLA on SimplerEnv-WidowX. Each task is evaluated with 24 rollouts per seed. Success Rate is reported as mean ± sample standard deviation over three seeds.

Benchmark / Task	Seed 1	Seed 2	Seed 3	Success Rate (%)
Spoon	21/24	21/24	21/24	87.5±0.0
Carrot	16/24	15/24	17/24	66.7±4.2
Stack	15/24	17/24	19/24	70.8±8.3
Eggplant	23/24	23/24	23/24	95.8±0.0

Table 8: Same-checkpoint inference ablation on SimplerEnv-WidowX. Task columns report aggregate success counts over 72 trials per task. Avg. is the success rate over all 288 trials, and N_{dec} is the expected logical decoder budget per action chunk.

Mode	Spoon	Carrot	Stack	Eggplant	Avg.	Stage-B Ratio	N_{dec}
Stage A	58/72	44/72	47/72	67/72	75.0	0%	6.0
Random	59/72	45/72	48/72	67/72	76.0	24%	7.7
DepthFlow-VLA	63/72	48/72	51/72	69/72	**80.2**	24%	7.7
Uniform refinement	64/72	49/72	52/72	69/72	81.3	100%	13.0

B.3 Same-Checkpoint Inference Ablation on SimplerEnv-WidowX

Table 8 compares four inference modes using the same three BridgeData-V2-trained checkpoints, the same initial-state keys, and the same 24-rollout-per-task protocol used for the main WidowX result. Stage-A-only decoding disables Stage B; random routing invokes Stage B under the same 24% budget as the learned router; and uniform refinement invokes Stage B for every action chunk. No model is retrained for this comparison. All four modes use the fully trained DepthFlow-VLA checkpoints. Stage A is therefore an inference mode with Stage B disabled, rather than a separately trained backbone-only or standard flow-matching baseline.

The aggregate success counts for Stage A, random routing, DepthFlow-VLA, and uniform refinement are 216/288, 219/288, 231/288, and 234/288, respectively. Adaptive routing improves aggregate success from 75.0% to 80.2%, whereas randomly spending the same Stage-B budget reaches only 76.0%. Uniform refinement reaches 81.3%, only 1.1 points above adaptive routing while invoking Stage B on every action chunk. Relative to Stage-A-only decoding, adaptive and uniform refinement yield 15 and 18 additional successful rollouts, respectively. Adaptive routing therefore retains 83.3% of the aggregate full-refinement success gain while using 7.7 rather than 13.0 expected decoder evaluations. This rollout-level gain retention is distinct from chunk-level router recall, which is measured in Table 4.

B.4 Vanilla Flow-Matching Control

Table 9 summarizes the vanilla flow-matching and DepthFlow controls. For each benchmark, we train a standard single-stage flow-matching control using the same backbone initialization, training data, optimizer, batch size, action normalization, and number of training steps as DepthFlow-VLA. It omits hard-sample weighting, routing supervision, and the Stage-B objective. At inference, we evaluate the vanilla control with standard full-trajectory Euler decoding using 6 or 13 decoder evaluations.

At the six-call decoder budget, the Stage-A mode of the fully trained DepthFlow checkpoint exceeds the separately trained vanilla control by 1.1 points on LIBERO and 2.8 points on WidowX; the latter vanilla result is 208/288 successful rollouts (72.2%). Increasing vanilla inference depth from 6 to 13 produces gains of 0.6 and 1.4 points, respectively, but does not recover the performance of coarse-anchored refinement. Because vanilla FM uses standard full-trajectory integration whereas Stage A uses the midpoint shortcut, this comparison reflects the combined effect of the DepthFlow training and coarse-decoding design rather than a loss-only effect. Adaptive routing then adds 2.4 points on LIBERO and 5.2 points on WidowX over the Stage-A mode of the same trained checkpoints, isolating the inference-time contribution of routing and refinement.

Table 9: Vanilla flow-matching and DepthFlow controls. The vanilla rows use standard full-trajectory Euler decoding at the indicated budgets. WidowX values aggregate 288 trials. N_{dec} for DepthFlow-VLA is reported as LIBERO / WidowX because the observed routing ratios are 22% / 24%.

Model / Mode	N_{dec}	**LIBERO Avg.**	**WidowX Avg.**
Vanilla FM (K=6)	6.0	94.8	72.2
Vanilla FM (K=13)	13.0	95.4	73.6
DepthFlow checkpoint, Stage A	6.0	95.9	75.0
DepthFlow-VLA	**7.5 / 7.7**	**98.3**	**80.2**
Uniform refinement	13.0	98.5	81.3

B.5 Training Hyperparameters

Table 10 lists the optimization and decoding hyperparameters. The same optimizer configuration and decoder settings are used across all simulation experiments unless otherwise noted.

Table 10: Training and decoding hyperparameters used in DepthFlow-VLA.

Hyperparameter	**Value**
Optimizer	AdamW
Learning rate	2×10^{-4}
Betas	$(0.9, 0.95)$
Weight decay	0
Per-GPU batch size	64
Number of GPUs	4
Global batch size	256
Training steps	150,000
Warmup steps	0
Learning-rate scheduler	None
VLM learning-rate multiplier	0.1
Precision	bfloat16
Image resize	128×128
Action normalization	Yes
Data shuffling	Yes
Action horizon H	10
Stage-A split time τ_m	0.5
Stage-B refine time τ_r	0.15
Stage-A steps K_c	5
Stage-B steps K_f	7
Routing threshold δ	0.7
Depth loss weight λ_1	0.1
Stage-B loss weight λ_2	0.05

B.6 Compute Resources

All simulation experiments were run on 4×H100 GPUs with the budget listed in Table 10. Real-world adaptation and evaluation used the same hardware.

B.7 Decoder-Compute Accounting

We report the expected logical number of action-decoder evaluations per action chunk as a hardware-independent measure of inference compute. The vision-language backbone is evaluated once per chunk in all variants and is therefore common to the comparisons.

Table 11: Batch-size-one mean latency on a single H100. Stage A, DepthFlow-VLA, and uniform refinement include the vision–language backbone and all required decoder evaluations. Routing + re-corruption is the aggregate non-decoder overhead under the default adaptive setting.

Component / Mode	**Mean (ms)**
Vision–language backbone	18.4
One decoder evaluation	3.1
Routing + re-corruption overhead	0.4
Stage A (N_{dec}=6.0)	37.2
DepthFlow-VLA (N_{dec}=7.54)	**42.6**
Uniform refinement (N_{dec}=13.0)	59.2

Stage A uses K_c Euler evaluations followed by one additional evaluation for the one-step shortcut in Eq. 3. If a fraction ρ of action chunks is routed to Stage B, the expected number of evaluations of v_θ is

$$N_{\text{dec}} = (K_c + 1) + \rho K_f. \tag{21}$$

For the default setting (K_c=5, K_f=7, ρ=0.22), $N_{\text{dec}} = 7.54$, compared with 6 for Stage-A-only decoding and 13 for uniform Stage-B refinement.

For deeper single-stage controls that integrate directly to $\tau = 0$ without the Stage-A shortcut, the decoder cost is the stated number of Euler evaluations. The lightweight routing MLP and re-corruption operation do not invoke v_θ and are not included in N_{dec}. This architecture-level measure is independent of hardware and batching. We complement it with batch-size-one wall-clock measurements below.

B.8 End-to-End Latency Measurement

All latency measurements use the same LIBERO checkpoint and input shape on a single NVIDIA H100 GPU with batch size 1, bfloat16 precision, and action horizon H=10. We run 50 warm-up iterations followed by 500 timed iterations, calling `torch.cuda.synchronize()` before and after each timed region. Environment stepping and data loading are excluded. The timed region includes the vision–language backbone, routing and re-corruption when applicable, and all action-decoder evaluations. Adaptive timing uses the default threshold $\delta = 0.7$ and its observed 22% Stage-B ratio.

The measured totals closely follow the component accounting: Stage A is within 0.2 ms of $18.4 + 6 \times 3.1$, DepthFlow-VLA is within 0.5 ms of $18.4 + 7.54 \times 3.1 + 0.4$, and uniform refinement is within 0.5 ms of $18.4 + 13 \times 3.1$. Relative to uniform refinement, DepthFlow-VLA reduces mean end-to-end latency by 28.0% while retaining nearly the same success rate.

C Additional Evaluation on SimplerEnv-Fractal

Beyond the WidowX results in Sec. 4, we further evaluate DepthFlow-VLA on the Google Robot (Fractal) track of SimplerEnv (Li et al., 2024) as additional cross-embodiment evidence. This setting differs from WidowX in embodiment, action distribution, and visual domain.

Setup. We train DepthFlow-VLA on the public Fractal dataset (Brohan et al., 2022) and report variant-aggregated results on Pick, Move, and Open following the official SimplerEnv evaluation protocol. The vision-language backbone is initialized from the same public checkpoint used in the LIBERO and WidowX experiments; the action decoder and routing head are initialized randomly. Decoder hyperparameters $(\tau_m, \tau_r, K_c, K_f, \delta)$ are kept identical to the LIBERO and WidowX configurations. All results are averaged over three seeds. Baseline numbers are taken from prior reports under the same protocol.

Results. As shown in Table 12, DepthFlow-VLA obtains 69.5% average success, compared with 65.6% for RT-2-X and 65.1% for ThinkAct. On Open, it reaches 55.3%, 7.7 percentage points

Table 12: Results on SimplerEnv-Fractal (Google Robot). Variant-aggregated success rates (%) on Pick, Move, and Open and their mean. DepthFlow-VLA results are averaged over three seeds.

Method	Pick	Move	Open	Avg
RT-1-X (O'Neill et al., 2024)	49.0	32.3	29.4	36.9
RT-2-X (O'Neill et al., 2024)	82.3	**79.2**	35.3	65.6
Octo-Base (Team et al., 2024)	0.6	3.1	1.1	1.6
OpenVLA (Kim et al., 2024)	54.5	47.7	17.7	40.0
OpenVLA-OFT (Kim et al., 2025)	65.3	59.0	12.2	45.5
RoboVLM (Li et al., 2026)	75.6	60.0	10.6	48.7
TraceVLA (Zheng et al., 2024)	60.0	56.4	31.0	49.1
π_0 (Black et al., 2024)	75.2	63.7	25.6	54.8
π_0-FAST (Pertsch et al., 2025)	77.6	68.2	31.3	59.0
GR00T-N1 (Bjorck et al., 2025)	78.8	62.5	13.2	51.5
DD-VLA (Liang et al., 2025)	82.5	64.6	23.6	56.9
ThinkAct (Huang et al., 2025)	84.0	63.8	47.6	65.1
DepthFlow-VLA	**85.4**	67.8	**55.3**	**69.5**

Table 13: Training objective ablation on LIBERO. We evaluate hard-sample weighting, the depth-selection loss, and the Stage-B refinement loss. All numbers are success rates in %.

Variant	Spatial	Object	Goal	Long
w/o hard-sample weighting	96.5	98.8	97.2	96.0
w/o depth-selection loss	95.6	97.8	96.0	94.2
w/o Stage-B refinement loss	94.4	96.8	95.8	93.8
DepthFlow-VLA	98.4	99.4	97.8	97.6

above ThinkAct. Its Move score of 67.8% is below RT-2-X (79.2%) and close to π_0-FAST (68.2%). Together with the WidowX results, these evaluations demonstrate the use of the same coarse-to-refine decoder and decoding hyperparameters in separately trained policies across robot embodiments.

Under the shared threshold $\delta = 0.7$, Stage B is invoked for 23% of Fractal action chunks, corresponding to $6 + 0.23 \times 7 = 7.61$ expected decoder evaluations per chunk. Together with the 24% WidowX ratio, this shows that the router remains selective across both SimplerEnv embodiments rather than degenerating into uniform refinement.

D Additional Efficiency Analysis

D.1 Training Objective Ablation

Table 13 ablates each term of the training objective. Removing hard-sample weighting yields the smallest average drop among the three ablations. Removing the depth-selection loss causes a larger drop, especially on LIBERO-Long, while removing the Stage-B refinement loss causes the largest degradation and falls below the Stage-A-only baseline.

D.2 Routing Threshold Sweep

Lowering δ increases Stage-B usage and improves success with progressively diminishing returns. The default $\delta = 0.7$ reaches 98.3% with 7.54 expected decoder evaluations, compared with 98.5% and 13.0 evaluations under uniform refinement. Increasing Stage-B usage beyond 35% produces no additional gain at the reported precision. The default threshold therefore provides a practical knee of the observed accuracy–compute curve.

D.3 Paired-Benefit Diagnostic Details

For a fixed refinement budget of $B = 44$ chunks, let $\mathcal{S}^\star$ contain the B chunks with the largest positive paired benefits Δ_i. For any routing policy selecting a set $\mathcal{S}$ of the same size, we define offline

Table 14: Routing-threshold sweep on LIBERO. Each threshold is evaluated using the same three trained checkpoints, initial-state keys, and random seeds. δ=1 disables Stage B, while δ=0 refines every chunk. N_{dec} is computed from Eq. 21.

δ	**Stage-B Ratio**	**Avg. Success**	N_{dec}
1.0	0%	95.9	6.0
0.9	7%	97.4	6.5
0.8	14%	98.0	7.0
0.7	**22%**	**98.3**	**7.5**
0.6	28%	98.4	8.0
0.5	35%	98.5	8.5
0.4	46%	98.5	9.2
0.0	100%	98.5	13.0

oracle-gain recovery as

$$\text{OGR}(\mathcal{S}) = \frac{\sum_{i\in\mathcal{S}} \max(\Delta_i, 0)}{\sum_{i\in\mathcal{S}^\star} \max(\Delta_i, 0)}. \tag{22}$$

With negative benefits clipped to zero, OGR measures positive-gain recovery rather than net error reduction. It is computed from offline action errors, not closed-loop success. Figure 5 shows the corresponding routing-score distributions across the four LIBERO suites.

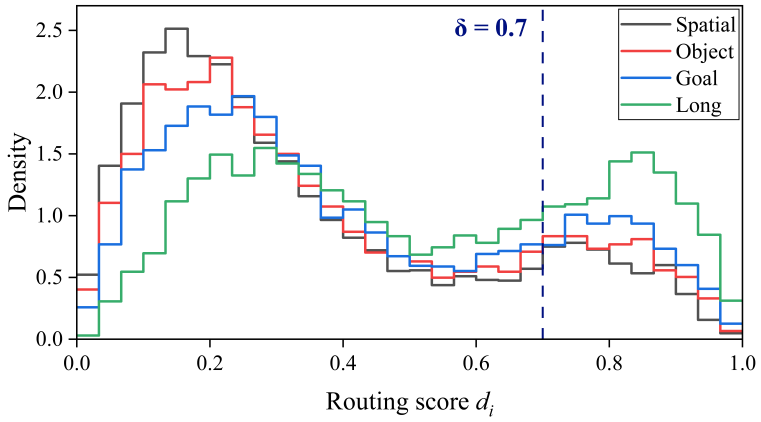

Figure 5: Routing-score distributions on the four LIBERO suites. The dashed line marks $\delta = 0.7$. Bimodality shows that the router produces separated operating scores, but does not by itself establish routing correctness; correctness is evaluated against paired offline benefit in Table 4.

D.4 Per-Task Routing Analysis on LIBERO-Long

We examine how Stage-B usage varies across the ten LIBERO-Long tasks. For this diagnostic, each task is evaluated over 50 rollouts. Its routing ratio is the fraction of action chunks with $d_i \geq \delta$ at the default threshold $\delta = 0.7$. We report this chunk-level ratio together with the task's rollout-level success rate. These quantities describe compute allocation and task outcome at different granularities.

As shown in Fig. 6, Stage-B usage varies across tasks. T9 and T10 have the highest routing ratios and lower success rates than most other tasks, whereas several tasks with near-saturated success use refinement less frequently. This pattern shows task-dependent compute allocation. Chunk-level routing quality is evaluated separately against paired refinement benefit in Table 4.

D.5 Additional Routing Baselines

Table 15 compares the learned router with two equal-budget alternatives. *Random routing* sends each chunk to Stage B with a fixed probability calibrated to the default refinement budget. *Heuristic*

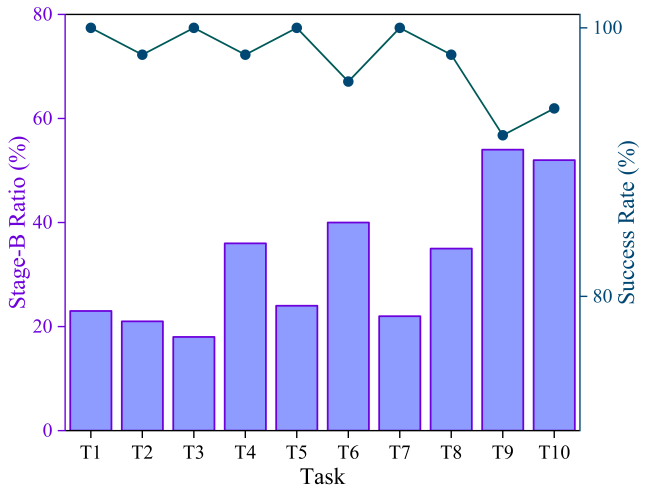

Figure 6: Per-task Stage-B routing ratios and success rates on LIBERO-Long. Bars show the fraction of action chunks routed to Stage-B refinement under the default threshold $\delta = 0.7$, and the line shows the corresponding task-level success rate. Routing ratios are computed at the action-chunk level, whereas success rates are computed at the rollout level. Tasks T1–T10 are listed in the default LIBERO-Long task order.

Table 15: Equal-budget routing baselines on LIBERO. All variants route 22% of action chunks and therefore use the same expected decoder budget.

Variant	**Avg**	**Avg. v_θ Evals**	**Stage-B Ratio**
Random routing	96.1	7.5	22%
Heuristic routing	96.7	7.5	22%
Adaptive routing	**98.3**	7.5	22%

routing replaces the learned score with a coarse-state proxy

$$s_i = \frac{1}{\sqrt{Hd_a}} \left\| \hat{x}_0^{\mathrm{c},i} - x_{\tau_m}^i \right\|_2,$$

which measures the update magnitude from the Stage-A midpoint to the coarse prediction. The top-s_i samples are routed to Stage B under the same 22% budget.

At the same 22% refinement budget, adaptive routing reaches 98.3% average success, compared with 96.1% for random routing and 96.7% for the coarse-state heuristic. Because the learned output is a continuous score, varying δ also provides the accuracy–compute trade-off reported in Table 14.

D.6 Sensitivity to Decoding Time and Step Counts

We first evaluate the sensitivity of DepthFlow-VLA to the two time parameters τ_m and τ_r. We choose $\tau_m = 0.5$ as a natural midpoint for Stage-A coarse decoding and set $\tau_r = 0.15$ to keep Stage-B within a low-noise correction regime. These choices reflect the intended roles of the two stages, and we verify below that performance remains competitive in a neighborhood of these values. As shown in Table 16, varying the Stage-A split time around the default value or changing the low-noise refinement time leads to only small fluctuations in average success rate. The default setting achieves a favorable accuracy–compute operating point, while nearby settings remain competitive, suggesting that the proposed coarse-to-refine decoder is not overly sensitive to a narrow choice of time parameters.

We further examine sensitivity to the inference-time integration budgets K_c and K_f while keeping the trained checkpoint fixed.

Table 17 shows that the default configuration lies in a flat region of the accuracy–compute surface. Reducing either integration budget mildly decreases success, whereas increasing K_c from 5 to 7

Table 16: Sensitivity to the Stage-A split time τ_m and Stage-B refinement time τ_r on LIBERO. We vary one parameter at a time while keeping the other fixed. N_{dec} is computed from Eq. 21.

τ_m	τ_r	**Avg. Success**	$\boldsymbol{N}_{\text{dec}}$	**Stage-B Ratio**
0.4	0.15	97.9	7.5	21%
0.5	0.15	98.3	7.5	22%
0.6	0.15	98.0	7.6	23%
0.5	0.10	97.8	7.5	22%
0.5	0.20	98.1	7.6	23%

Table 17: Sensitivity to Stage-A and Stage-B integration steps on LIBERO. Expected decoder evaluations account for the routing ratio of each configuration.

K_c	K_f	Avg Success	Avg. v_θ Evals	Stage-B Ratio
3	7	97.7	5.7	24%
5	7	98.3	7.5	22%
7	7	98.2	9.5	21%
5	5	98.0	7.1	22%
5	9	98.3	8.0	22%

or K_f from 7 to 9 provides no average gain despite requiring additional decoder evaluations. The default (K_c=5, K_f=7) therefore provides a favorable compute–accuracy operating point.

D.7 Real-World Setup Details

We provide additional details for the real-world evaluation in Sec. 4.4. As shown in Fig. 7, all experiments are conducted on the same UR5e tabletop platform with a three-finger gripper and a fixed RealSense D435i camera. The policy receives RGB observations, language instructions, and robot proprioceptive state as inputs, and predicts normalized action chunks that are executed in a receding-horizon manner.

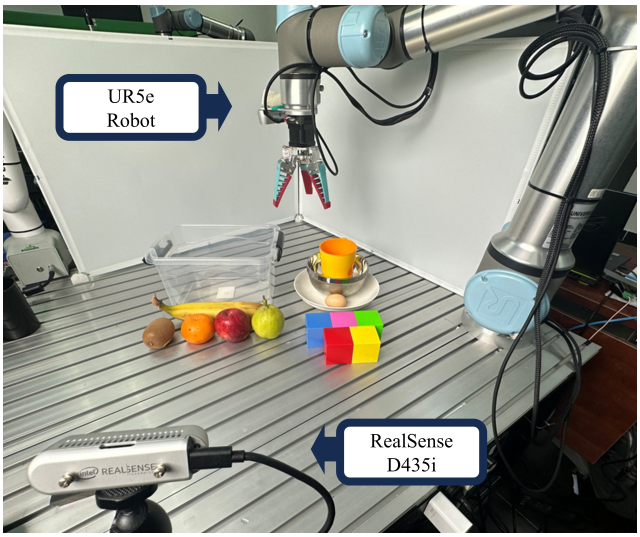

Figure 7: Real-world experimental setup. The platform consists of a UR5e robot arm equipped with a three-finger gripper and a fixed Intel RealSense D435i camera observing a tabletop workspace. The same setup is used for all three real-world tasks.

Demonstration collection and adaptation. For each task, we collect 100 demonstrations on the target platform through human teleoperation. Demonstrations are collected under randomized object

poses within a bounded tabletop workspace, and the same workspace is used at evaluation time. Demonstration lengths vary by task, and we do not filter demonstrations by a fixed trajectory length.

DepthFlow-VLA and Stage-A-only are adapted from the same LIBERO-trained DepthFlow-VLA checkpoint, with Stage B disabled only at inference for the Stage-A-only variant. The $\pi_{0.5}$ baseline is adapted from our independently trained LIBERO checkpoint. All methods use the same 100 task-specific demonstrations, number of optimization steps, action normalization convention, and evaluation reset distribution. Model-specific trainable components follow the original architecture of each method. Unless otherwise stated, real-world adaptation follows the same training recipe as simulation: the vision-language backbone is updated with a reduced learning-rate multiplier, while the action decoder and the routing head are trained with the base learning rate.

Tasks and language instructions. We evaluate three long-horizon tabletop manipulation tasks. Task 1 is five-fruit collection, where five fruits of different categories are placed at randomized positions within the workspace and the robot sequentially picks them up and places them into a basket; the trial proceeds only when each fruit is delivered into the basket without being dropped outside. Task 2 is two-step color block stacking, where five differently colored blocks are placed on the table and the robot must identify the target blocks by color, first stack the blue block on the green block and then stack the red block on the blue block; the remaining two blocks act as on-table distractors and are not part of the target stack. Task 3 is cup pouring with placement, where the robot grasps a cup containing several small balls, pours the balls into a bowl, and then places the empty cup onto a plate; the task therefore couples a pouring sub-action that requires controlled orientation with a subsequent precise placement.

Evaluation protocol and success criteria. For each method and task, we run 50 evaluation trials with randomized initial object poses. A Task 1 trial is counted as successful only if all five fruits are placed into the basket. A Task 2 trial is counted as successful only if the blue block is placed stably on the green block and the red block is then placed stably on the blue block after release. A Task 3 trial is counted as successful only if the small balls are poured into the bowl and the cup is finally placed stably on the plate. Trials involving a wrong target object, failed grasp, object drop, unstable final placement, unsafe collision, or timeout are counted as failures.

D.8 Additional Real-World Results

Table 18: Detailed real-world success rates (%) on the three tabletop manipulation tasks. Each entry reports the success rate and the corresponding 95% Clopper–Pearson confidence interval over 50 trials. Task 1, Task 2, and Task 3 correspond to five-fruit collection, two-step color block stacking, and cup pouring with placement, respectively.

Task	Stage-A Only	$\pi_{0.5}$	DepthFlow-VLA
Task 1	86 [73.3, 94.2]	94 [83.5, 98.7]	**96 [86.3, 99.5]**
Task 2	74 [59.7, 85.4]	84 [70.9, 92.8]	**88 [75.7, 95.5]**
Task 3	58 [43.2, 71.8]	70 [55.4, 82.1]	**78 [64.0, 88.5]**

Table 18 complements Fig. 4 with the per-task numbers underlying the bar chart. Task 1 denotes five-fruit collection, Task 2 denotes two-step color block stacking, and Task 3 denotes cup pouring with placement. Each entry reports the success rate together with the corresponding 95% Clopper–Pearson confidence interval over 50 trials, following the adaptation and evaluation protocol described in Sec. 4.4 and detailed in Appendix D.7.

E Additional Discussions

E.1 Limitations

Several limitations remain. First, although our simulation evaluation spans LIBERO and both SimplerEnv tracks (WidowX and Google Robot), the real-world study covers only three tabletop tasks on a single robot; broader validation across robots and environments is still needed. Second,

although the main LIBERO and SimplerEnv-WidowX results are reported over three seeds, some auxiliary studies are still evaluated under fixed protocols, and we do not perform formal hypothesis tests. In addition, given the limited scale of the real-world study, the 2–8 point gaps over $\pi_{0.5}$ should be interpreted as directional trends rather than statistically tight differences. Third, the router is trained from a detached, ground-truth-corrupted midpoint estimate and a batch-relative residual label, rather than from the exact Stage-A rollout state or direct paired refinement benefit. Although Table 4 shows that its score ranks offline Stage-B benefit substantially better than chance, the measured recall is 61.2% and the false-negative rate is 38.8%. The score should therefore be interpreted as a refinement-priority ranking rather than a calibrated probability of closed-loop improvement. The fixed threshold may also require recalibration under substantially different task distributions. Constructing the Stage-B targets also requires replaying Stage A for difficult samples, which increases training-time computation. The latency measurements in Table 11 use batch size 1 on a single H100; realized savings may differ with hardware, batching, routed-sample compaction, and kernel scheduling. Performance also varies across tasks: DepthFlow-VLA scores below FPC-VLA on WidowX Stack and RT-2-X on Fractal Move.

E.2 Broader Impacts

By reducing the number of action-decoder evaluations required at deployment, this work may improve the accessibility of embodied AI research on resource-constrained platforms. Conversely, more capable and efficient robotic policies raise safety concerns when deployed without appropriate oversight: prediction errors, distribution shifts, or failures of language grounding can lead to unsafe behaviors or unintended interactions with surrounding objects. Our experiments are confined to simulation and a controlled tabletop platform, and we make no claim of readiness for safety-critical or unsupervised real-world deployment. Practical use should therefore incorporate additional safety constraints, monitoring, and human oversight.