SPOT THE DIFFERENCE: EFFICIENT VLA INFERENCE VIA DIFFERENTIAL CODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) models inherit the perception and reasoning of large vision-language models (VLMs), but they pay for it on every policy query: a multi-billion-parameter VLM re-encodes each new observation, even though consecutive frames of a manipulation rollout are highly redundant. We propose Differential VLA, which applies a principle shared by differential coding in communications, predictive coding in cognitive science, and I-frame/P-frame video compression: *encode a reference once, and just spot the difference afterwards*. Differential VLA runs the VLM (System 2) on a reference frame only once every T control steps and caches its tokens. In between, a lightweight *differential encoder* lets the features of the current frame attend to those of the reference frame and compresses the residual into a handful of *difference tokens*, which are appended to the cached tokens that condition the diffusion-transformer action head (System 1). Built on GR00T N1.7 with $T = 64$, Differential VLA achieves an average success rate of $93.5\,\%$ on LIBERO, which is within 1.0 points of the baseline, while calling the VLM $8\times$ less often. An amortized query costs $1.8\times$ fewer FLOPs and runs $1.27\times$ faster than an equally optimized baseline. On SimplerEnv, Differential VLA stays within 6.6 points of the baseline while calling the VLM $16\times$ less often, and on two real-world tasks with an SO-101 arm, it matches the baseline's success rate with 1.22 to $1.27\times$ lower latency.

1 INTRODUCTION

Vision-language-action (VLA) models extend pretrained vision-language models (VLMs) to robot control by conditioning an action decoder on the VLM's representation of camera images and a language instruction (Kim et al., 2024; Black et al., 2024; Physical Intelligence et al., 2025a;b; 2026; Bjorck et al., 2025). The VLM supplies broad semantic knowledge, but it is also by far the most expensive component of the policy: a multi-billion-parameter transformer is re-run on every new observation, even though the instruction is fixed for the whole episode and the scene changes only slightly between two consecutive control steps. To reduce this cost, recent VLA architectures (Bjorck et al., 2025; Zhang et al., 2024; Bu et al., 2024), inspired by dual process theory (Kahneman, 2011), separate a slow, deliberative "System 2" (the VLM) from a fast, reactive "System 1" (the action head), which generates a chunk of future actions conditioned on the semantic representation produced by System 2. System 1 must be lightweight, as it is typically evaluated several times per chunk (Zhao et al., 2023; Chi et al., 2025; Shin et al., 2026), e.g., once per decoded action token or once per denoising step. Executing several actions of each chunk per query amortizes the cost of the VLM over multiple control steps.

In this work we take a different view of the problem: the observation stream of a robot is a *signal* with strong temporal correlation, and the question of how to process it cheaply has been studied for decades in other fields. Three ideas in particular motivate our method.

- **Differential coding in digital communications.** Differential pulse-code modulation (Cutler, 1952; Elias, 1955) does not transmit the whole sequence as it is. Instead, it transmits the difference between two consecutive samples, which has a far smaller dynamic range and therefore requires far fewer bits.

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.

- **Predictive coding in cognitive science.** Predictive coding theories of perception (Friston & Kiebel, 2009) posit that the brain maintains an internal model of the world and that the signals ascending the sensory hierarchy mainly carry *prediction errors*: only the part of the input that the current internal model fails to explain is propagated and processed further.
- **I-frames and P-frames in video compression.** Video codecs (Gall, 1991; Wiegand et al., 2003; Sullivan et al., 2012) encode a reference frame (*I-frame*) in full and represent the following frames (*P-frames*) by motion vectors, which predict each frame from the reference, and the residual with respect to that prediction. This is why video is compressible by orders of magnitude at little perceptual loss.

All three share the same principle: *encode a reference once at high cost, and just spot the difference afterwards, at low cost*. We argue that this is exactly the structure that a VLA's perception stack should have.

We propose Differential VLA, which applies this principle to the dual-system VLA (Fig. 1). The VLM plays the role of the I-frame encoder: it is evaluated on a *reference frame* o_{kT} only once every T control steps, and its output tokens are cached. For every *current frame* $o_{kT+\Delta t}$ in between, a lightweight *differential encoder* (Fig. 3) plays the role of the P-frame encoder. It embeds both frames with a small vision foundation model, lets every current-frame token attend to the reference-frame tokens (a learned, soft analogue of the brain's internal model or a video codec's motion estimation), and compresses the result into a handful of *difference tokens*. These are appended to the cached VLM tokens that the diffusion-transformer action head attends to. The action head therefore always receives both the rich but stale semantics of the reference frame and a cheap but fresh account of how the scene has changed since then.

Our contributions are as follows:

- We draw the connection between VLA inference and differential, predictive, and inter-frame coding. Guided by it, we propose Differential VLA, a dual-system VLA in which the VLM is evaluated only on sparse reference frames and the fast action head is driven by the reference tokens together with a learned, compact encoding of the change since the reference.
- We design a differential encoder based on reference-conditioned cross-attention over DINOv2 (Oquab et al., 2024) features, and a simple training scheme that samples random reference offsets so that training matches the reference-reuse pattern seen at inference.
- We instantiate Differential VLA on GR00T N1.7 and evaluate it on LIBERO, SimplerEnv, and two real-world tasks with an SO-101 arm. On LIBERO, Differential VLA calls the VLM $8\times$ less often than GR00T N1.7, reduces the amortized latency by $1.27\times$ relative to an equally optimized baseline, and stays within 1.0 points of its average success rate. On SimplerEnv, it stays within 6.6 points of the baseline while calling the VLM $16\times$ less often. On the real robot, it matches the baseline's success rate with 1.22 to $1.27\times$ lower latency. Ablations isolate the effect of the pooling strategy and the injection point of the difference tokens, and verify that the action head relies on the difference tokens.

2 Preliminaries and Motivation

Dual-system VLA. We build on NVIDIA's GR00T N1.7, the latest release of the GR00T family (Bjorck et al., 2025). At each control step t it outputs an action chunk $\boldsymbol{A}_t = (\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, \ldots, \boldsymbol{a}_{t+H-1})$ based on the camera images $\mathbf{O}_t = (\mathbf{O}_t^1, \mathbf{O}_t^2, \ldots, \mathbf{O}_t^V)$ from V views, a language instruction ℓ, and the proprioceptive state $\boldsymbol{q}_t$. It adopts a dual-system architecture, where System 2 is a Cosmos-Reason2-2B (Agarwal et al., 2025) VLM f_ϕ truncated to its first 16 language-model layers that maps the images and the instruction to a token sequence

$$\boldsymbol{Z}_t = f_\phi(\mathbf{O}_t, \ell) \in \mathbb{R}^{N \times D}, \qquad D = 2048,$$

and System 1 is a diffusion transformer (DiT; Peebles & Xie 2023) action head g_θ trained with flow matching (Lipman et al., 2023), composed of 32 transformer blocks, every other one of which cross-attends to $\boldsymbol{Z}_t$, alternately to its image tokens and to its text tokens. At inference, the action chunk is

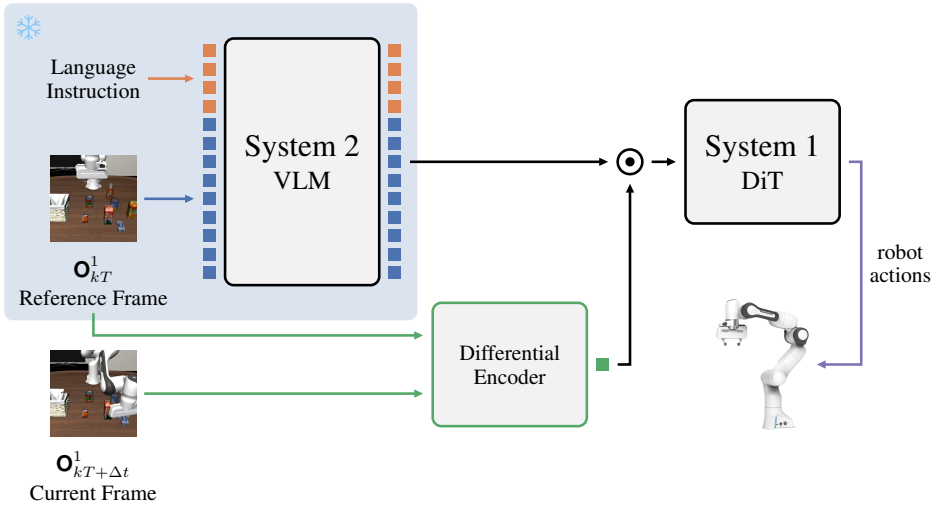


Figure 1: **Overview of Differential VLA.** The VLM (System 2) encodes the language instruction and the *reference frame* o_{kT} only once every T control steps, and its output tokens are cached. At every control step, the differential encoder compares the *current frame* $o_{kT+\Delta t}$ with the reference frame and produces a few compact difference tokens (green; one per view in our default setting), which are concatenated with the cached VLM tokens. The diffusion transformer (System 1) attends to this sequence while computing the action chunk.

generated through 4 denoising steps,

$$\boldsymbol{A}_t^0 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \qquad \boldsymbol{A}_t^{\tau+1} = \boldsymbol{A}_t^\tau + \frac{1}{4} g_\theta(\boldsymbol{A}_t^\tau, \tau, \boldsymbol{q}_t, \boldsymbol{Z}_t), \qquad \tau = 0, 1, 2, 3, \tag{1}$$

and the first n actions of the chunk are executed before the policy is queried again. The forward pass of f_ϕ dominates the cost of each query.

How redundant is the observation stream? To quantify the temporal redundancy that motivates our design, we encode every frame of the 379 LIBERO-10 demonstrations (Liu et al., 2023) with a pretrained DINOv2-small (Oquab et al., 2024) and measure, for frame pairs (o_t, o_{t+g}) at gap g, the average cosine similarity of the CLS tokens and the average chamfer similarity (Barrow et al., 1977) between the patch tokens, which is defined as

$$\text{chamfer}\left(\{\boldsymbol{x}_i\}_{i=1}^P, \{\boldsymbol{y}_j\}_{j=1}^P\right) = \frac{1}{2P}\left(\sum_{i=1}^P \max_j \text{sim}(\boldsymbol{x}_i, \boldsymbol{y}_j) + \sum_{j=1}^P \max_i \text{sim}(\boldsymbol{x}_i, \boldsymbol{y}_j)\right),$$

where $\{\boldsymbol{x}_i\}_{i=1}^P$ and $\{\boldsymbol{y}_j\}_{j=1}^P$ are patch token sequences of each frame and sim is the cosine similarity. Fig. 2 presents the results. Two observations stand out. First, the stream is highly redundant at short gaps: consecutive frames have a CLS similarity of 0.98 in the agent view. Second, the redundancy decays steadily with g, and much faster in the wrist view, whose chamfer similarity drops to 0.67 at $g = 64$. Therefore, a policy that simply reuses stale VLM tokens for T steps would be oblivious to a substantial fraction of what is currently happening in the scene, whereas one that re-encodes every frame would waste most of its compute on redundant content. Differential VLA occupies the middle ground: it reuses the expensive encoding and pays only for the change.

3 Differential Vision-Language-Action Models

In this section, we propose *Differential VLA*, an architecture that improves the computational efficiency of VLA inference by encoding a reference frame once and encoding afterwards only what has changed relative to it. We first provide an overview of the pipeline (Section 3.1), then describe its key component, the differential encoder (Section 3.2), and how its output is injected into the action head (Section 3.3). Finally, we discuss the training (Section 3.4) and inference (Section 3.5) procedures.

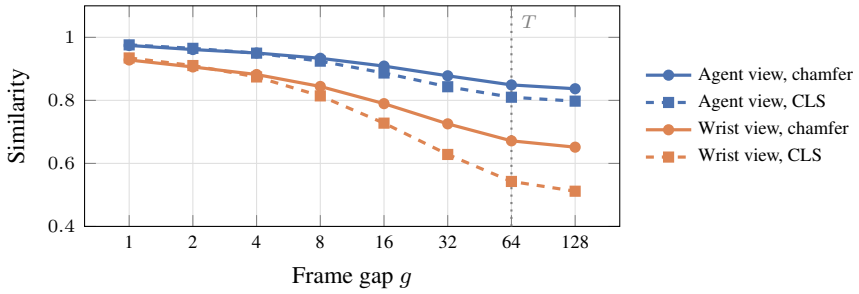


Figure 2: **Temporal redundancy of robot observations.** Similarity between DINOv2-small features of frames o_t and o_{t+g}, averaged over all frame pairs of the LIBERO-10 dataset. The solid lines denote the chamfer similarity between the patch token sequences. The dashed lines denote the cosine similarity between the CLS tokens. The gray dotted vertical line indicates the reference interval $T = 64$ used in our experiments.

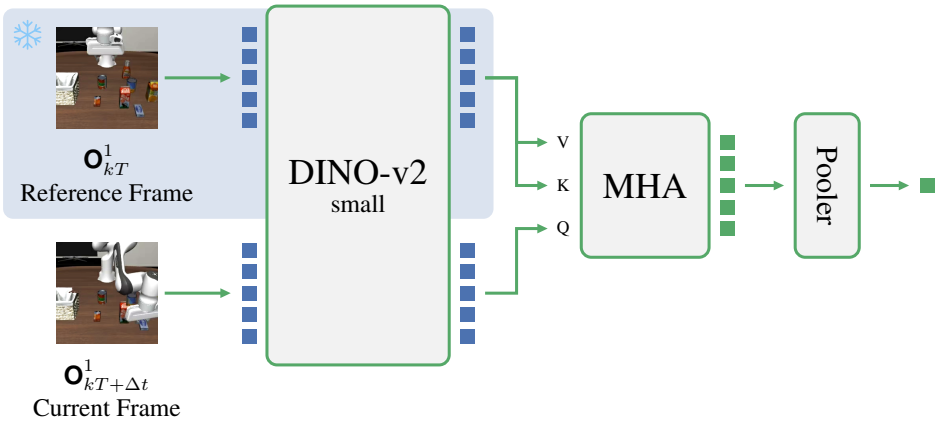


Figure 3: **Differential encoder.** The reference and current frames are embedded by a shared DINOv2-small. The current-frame tokens act as queries, and the reference-frame tokens as keys and values, of a multi-head cross-attention (MHA). The resulting tokens are compressed by a pooler into m difference tokens ($m = 1$ shown). The reference branch (shaded) is computed once per reference frame and is cached at inference.

3.1 Overview

Differential VLA encodes the observation stream at two rates. Once every T steps, that is, at $t = kT$, System 2 encodes the current frame, which becomes the *reference frame*, and its tokens $\bar{\boldsymbol{Z}}_k = f_\phi(\mathbf{O}_{kT}, \ell)$ are cached. At every step $t = kT + \Delta t$, $0 \le \Delta t < T$, a lightweight *differential encoder* h_ψ compares the current frame with the latest reference frame and summarizes what has changed into m *difference tokens* per view,

$$\boldsymbol{U}_t = h_\psi(\mathbf{O}_{kT+\Delta t}, \mathbf{O}_{kT}) \in \mathbb{R}^{mV \times D}.$$

Instead of $\boldsymbol{Z}_t$, the action head then conditions on $[\bar{\boldsymbol{Z}}_k; \boldsymbol{U}_t]$, which contains the rich but stale semantics of the reference frame together with a compact account of what has changed since. Because the differential encoder also caches its features of the reference frame, each step costs only one pass of its small image encoder over the current frame and one pass of the action head, while the cost of the VLM is amortized over T steps.

3.2 Differential Encoder

Fig. 3 shows the differential encoder, which is applied to each camera view independently with shared weights. We therefore describe it for a single view v from now on.

Frame features. We embed both frames with a shared DINOv2-small (Oquab et al., 2024) encoder E. For a 224×224 input with 14×14 patches, this gives $P = 257$ tokens (one CLS token and 16×16 patch tokens) of width $d = 384$:

$$\boldsymbol{F}_c^v = E(\mathbf{O}_{kT+\Delta t}^v) + \boldsymbol{P}, \qquad \boldsymbol{F}_r^v = E(\mathbf{O}_{kT}^v) + \boldsymbol{P}, \qquad \boldsymbol{F}_c^v, \boldsymbol{F}_r^v \in \mathbb{R}^{P \times d},$$

where $\boldsymbol{P} \in \mathbb{R}^{P \times d}$ is a learned positional embedding shared by both frames. We use a self-supervised vision encoder, rather than the VLM's own vision tower, because it is small enough to run at every step and its features are known to capture fine-grained spatial correspondence (Oquab et al., 2024; Zhang et al., 2023). We denote the rest of the differential encoder, that is, the cross-attention and the pooler described below, by $\tilde{h}_\psi$, so that the difference tokens of view v are $\boldsymbol{U}_t^v = \tilde{h}_\psi(\boldsymbol{F}_c^v, \boldsymbol{F}_r^v)$.

Reference-conditioned cross-attention. The simplest way to encode the change is to subtract the reference features from the current ones, $\boldsymbol{F}_c^v - \boldsymbol{F}_r^v$. This compares each current-frame patch with the reference-frame patch at the same location, which is only meaningful when the two patches contain the same objects, or in other words, when no object in the scene moves. We instead adopt multi-head attention (MHA; Vaswani et al. 2017) with the current-frame tokens as queries and the reference-frame tokens as keys and values, and subtract its output from the current-frame tokens:[1]

$$\tilde{\boldsymbol{F}}^v = \mathrm{LayerNorm}\left(\boldsymbol{F}_c^v - \mathrm{MHA}(Q = \boldsymbol{F}_c^v, K = \boldsymbol{F}_r^v, V = \boldsymbol{F}_r^v)\right).$$

This design is motivated by predictive coding. The cross-attention acts as the internal model: each current patch, as a query, retrieves the reference content it corresponds to. If we interpret this as a prediction of that patch, then subtracting it from the current patch yields the prediction error, that is, the part of the current patch that the reference frame fails to explain. Unlike $\boldsymbol{F}_c^v - \boldsymbol{F}_r^v$, each patch is thus compared with the reference content it actually corresponds to, wherever the content has moved, rather than with whatever occupies the same location. Moreover, even if there is newly revealed content in the current frame, its information is preserved by the $\boldsymbol{F}_c^v$ term.

Pooling into difference tokens. The P tokens are then compressed to m difference tokens of dimension D. In our default *concat-project* pooler, we flatten $\tilde{\boldsymbol{F}}^v$ and pass it through a single linear layer,

$$\boldsymbol{U}_t^v = \mathrm{reshape}_{m \times D}\left(\boldsymbol{W} \cdot \mathrm{vec}(\tilde{\boldsymbol{F}}^v) + \boldsymbol{b}\right), \qquad \boldsymbol{W} \in \mathbb{R}^{mD \times Pd},$$

which preserves the spatial layout of the change. We compare it with three alternatives in Section 4.6: using the output at the CLS position or the mean over all tokens, each followed by a linear projection ($m = 1$), and a learned-query attention pooler (Lee et al., 2019) with m learned queries. The tokens of all views are concatenated, $\boldsymbol{U}_t = [\boldsymbol{U}_t^1; \boldsymbol{U}_t^2; \ldots; \boldsymbol{U}_t^V]$. We use $m = 1$ by default, so that the entire change in a view is summarized by a single token.

3.3 Injecting the Difference Tokens

We append the difference tokens to the cached VLM tokens, $[\bar{\boldsymbol{Z}}_k; \boldsymbol{U}_t]$, and mark them as *image* tokens, so that the DiT blocks that cross-attend to image tokens can see the fresh difference tokens next to the stale reference-image tokens. The concatenated sequence passes through the same layer normalization as the VLM tokens before entering the action head. This *cross-attention injection* leaves the action head architecture unchanged. In Section 4.6, we compare it with two alternatives: *self-attention injection*, which prepends the difference tokens to the state and action tokens of the DiT, and *VLM update*, which edits the cached VLM tokens with a gated cross-attention where $\bar{\boldsymbol{Z}}_k$ attends to the current-frame features $\boldsymbol{F}_c^v$ of all views.

3.4 Training

We fine-tune from the pretrained GR00T N1.7 checkpoint. For each training sample at time t, we draw a reference offset $\Delta t \sim \mathcal{U}\{0, \ldots, T-1\}$ and load exactly two frames: the reference frame $\mathbf{O}_{t-\Delta t}$ and the current frame $\mathbf{O}_t$. If $t - \Delta t < 0$, the reference index is clamped to 0, which mirrors

[1] By negating the output projection of the MHA, we can always make this equivalent to a standard residual connection, which is how we implement it.

the beginning of an episode at test time, where the reference and the current frame coincide. The VLM receives only the reference frame and the instruction; the differential encoder receives both frames; and the action head receives the proprioceptive state and the noised action chunk, both taken at the current time t. We apply the same random image augmentation to the two frames of a pair, so that the augmentation is not mistaken for a change in the scene. The training objective is the standard flow matching loss, with $\boldsymbol{Z}_t$ replaced by $[f_\phi(\mathbf{O}_{t-\Delta t}, \ell); h_\psi(\mathbf{O}_t, \mathbf{O}_{t-\Delta t})]$. We do not adopt any auxiliary loss. Since the difference tokens are the action head's only source of visual information about the current frame, the flow matching loss alone provides a training signal for the differential encoder. The VLM is kept frozen, while everything else is trained.

3.5 Inference

Algorithm 1 in Appendix B gives the full inference procedure. The reference is refreshed every T environment steps rather than every T policy queries, so that Δt at test time follows the same range as in training even though the policy executes n actions per query. With $T = 64$ and $n = 8$ (our LIBERO setting), the VLM is evaluated on one out of every 8 policy queries, and the other 7 queries only run the differential encoder and the action head. The cache is cleared every time an episode resets.

4 Experiments

We evaluate Differential VLA to answer the following questions: (i) How close does it stay to a policy that runs the VLM at every query, and how much better is it than naively reusing stale VLM features? (ii) How much inference cost does it save? (iii) Does it transfer to real robots? (iv) Which design choices of the differential encoder matter?

4.1 Setup

All models are fine-tuned from GR00T N1.7[2]. Unless stated otherwise, Differential VLA uses DINOv2-small, concat-project pooling with $m = 1$ difference token per view, cross-attention injection, and $T = 64$. We train with AdamW at a learning rate of 10^{-4}, weight decay 10^{-5}, $5\,\%$ linear warm-up, and gradient accumulation, for 20k steps on 2 NVIDIA H200 GPUs. The full hyperparameter list can be found in Appendix C.

4.2 LIBERO

LIBERO (Liu et al., 2023) consists of four suites of 10 tasks each: Spatial, Object, Goal, and Long (LIBERO-10). We train one model per suite on the corresponding demonstrations with a batch size of 640 and evaluate each task for 100 episodes, twice on each of its 50 predefined initial states, executing $n = 8$ actions of each 16-step chunk, allowing up to 720 steps per episode. Two camera views (agent and wrist) are used. For the baseline, we evaluate the official LIBERO-fine-tuned GR00T N1.7 checkpoints under our protocol; the reported numbers are from the official Isaac-GR00T repository[3]. Due to compute constraints, the Object model of Differential VLA was trained without augmenting the inputs of the differential encoder. The VLM inputs were augmented in all of the runs. The LIBERO-10 model and all ablations (Section 4.6) use the full recipe of Section 3.4. Table 1 reports success rates together with the VLM call frequency.

Differential VLA reaches an average success rate of $93.5\,\%$, 1.0 points below the official checkpoint under our evaluation while calling the VLM $8\times$ less often, and 0.8 points below GR00T N1.7 with $n = 16$, which calls it $4\times$ more often than Differential VLA. Reusing the cached VLM tokens for the same interval without difference tokens fails almost completely ($0.8\,\%$), so the stale tokens alone are not enough to act on. Contrary to the expectation that long-horizon tasks suffer most from stale semantics, Differential VLA outperforms the baseline on LIBERO-10 by 3.3 points; the gap is instead concentrated on LIBERO-Goal ($87.7\,\%$ against $96.2\,\%$). Within LIBERO-Goal, the weakest

[2] https://huggingface.co/nvidia/GR00T-N1.7-3B

[3] https://github.com/NVIDIA/Isaac-GR00T/blob/51d4c89f72fda44cbf77285c6a8114b52676b8a1/examples/LIBERO/README.md

Table 1: **Success rates (%) on LIBERO.** Freq. is the average number of VLM calls per environment step.

Method	Freq.	Spatial	Object	Goal	10 (Long)	Avg.
N1.7 (reported)	1/8	97.7	98.5	97.5	94.4	97.0
N1.7 (official ckpt., our eval.)	1/8	95.1	98.8	96.2	88.0	94.5
N1.7 ($n = 16$)	1/16	93.8	97.6	95.4	90.2	94.3
N1.7 + cache ($T = 64$)	1/64	0.2	0.0	3.0	0.0	0.8
Differential VLA (T=64)	1/64	95.7	99.4	87.7	91.3	93.5

Table 2: **Real-world results on SO-101** (24 paired trials per task). Latency is the mean server-side inference time per policy query on one RTX 4090, over all queries of all trials.

	Block placing		Cup stacking	
Method	Success	Latency (ms)	Success	Latency (ms)
GR00T N1.7	17/24	54.9	8/24	55.7
Differential VLA (T=128)	16/24	44.9	8/24	43.8

task by far is putting the bowl on the plate, at 53 %, while the other nine tasks are at 82 % or above (Table 14). In its failures, the policy either reaches for the plate instead of the bowl or carries on as if it held the bowl after a failed grasp. The same failures occur when the VLM is run on every query ($T = 8$), which suggests that they stem from fine-tuning rather than from stale VLM tokens.

4.3 SimplerEnv

On the WidowX setting of SimplerEnv (Li et al., 2024), with policies trained on Bridge-Data V2 (Walke et al., 2023), Differential VLA ($T = 64$) reaches an average success rate of 54.7 %, against 61.3 % for the official GR00T N1.7 checkpoint under our evaluation, while calling the VLM 16× less often. The gap comes mostly from the spoon, carrot, and drawer-opening tasks, whereas Differential VLA succeeds far more often on putting the eggplant in the sink (45.5 % against 1.0 %). Appendix F gives the setup and per-task results.

4.4 Real-World Experiments

Setup. We deploy Differential VLA on a low-cost SO-101 arm (6 DoF including the gripper) using the LeRobot framework (Cadene et al., 2024), with a fixed bird's-eye-view camera above the workspace and a wrist camera, on two tasks (Fig. 4): *block placing*, which requires picking up a blue cube among distractors and placing it on a red target, and *cup stacking*, which requires stacking three cups into a single stack. For each task we collect 50 teleoperated demonstrations, fine-tune both methods with the same recipe, and evaluate them on the same 24 systematically enumerated initial configurations, so the trials are paired. The policy executes $n = 16$ actions per query, and we set $T = 128$ so that, as on LIBERO, the VLM runs on one out of every 8 policy queries. The policy is served from a separate workstation, and we report the average server-side inference latency over all queries of all trials. Appendix H gives the details of the tasks, training, and evaluation.

Differential VLA matches the baseline on both tasks (Table 2), 16/24 against 17/24 on block placing and 8/24 against 8/24 on cup stacking, while lowering the mean inference latency from 54.9 ms to 44.9 ms on block placing and from 55.7 ms to 43.8 ms on cup stacking, a 1.22× and 1.27× reduction that agrees with the amortized speed-up measured in Section 4.5. Appendix H breaks the latency down by query type.

Table 3: **Inference cost per policy query.** Latency is the end-to-end median, including preprocessing, over 3200 queries on one RTX 4090, with the 95th percentile in parentheses. The amortized rows are the mean cost of a query over 8 queries, $(C_2 + 7C_1)/8$. FLOPs are analytic estimates (Appendix D). Speed-up is relative to the PyTorch baseline, which runs the VLM on every query and is optimized to the same degree as Differential VLA.

Method	Query type	GFLOPs	Latency (ms)	Speed-up
GR00T N1.7	every query	1033	31.5 (31.8)	1.0×
Differential VLA (PyTorch)	System 2	1062	38.1 (38.4)	0.83×
	System 1	490	23.0 (23.2)	1.37×
	amortized (T=64, n=8)	561	24.9	1.27×

4.5 Inference Cost

We report inference cost in two ways: FLOPs and wall-clock latency. Both are shown in Table 3. FLOPs are counted analytically with the primitives of NVIDIA's Cosmos FLOP estimator. Appendix D gives the breakdown. Latency is measured end to end, including preprocessing, on a single RTX 4090 with batch size 1, over 3200 queries after 64 warm-up queries. For Differential VLA we report System-2 queries (which additionally run the VLM) and System-1 queries separately, as well as the amortized cost at $T = 64$ and $n = 8$, where one in every 8 queries is a System-2 query.

Skipping the VLM removes 572 GFLOPs from a query, so an amortized query costs $1.8\times$ fewer operations than a baseline query and runs $1.27\times$ faster end to end. We believe that the discrepancy between the two speed-ups is largely due to per-query costs that do not scale with FLOPs, such as data preprocessing and kernel-launch overhead in eager mode. All of the numbers are based on the eager PyTorch (Ansel et al., 2024) runs. Appendix E discusses compiled execution.

4.6 Ablation Studies

We ablate the main design choices of the differential encoder on the LIBERO-10 suite, changing one factor at a time from the default configuration (Table 4).

Pooling. We compare the CLS token, mean pooling, a learned-query attention pooler, and the default concat-project pooler. The CLS and mean poolers discard the spatial layout of the change, whereas concat-project preserves it at the cost of a large projection matrix. CLS, mean, and concat-project pooling perform within 0.4 points of each other, which is within the evaluation noise, while the learned-query pooler is 2.1 points lower. The spatial layout thus does not need to be preserved explicitly, and the much cheaper CLS and mean poolers are viable alternatives.

Injection. We compare cross-attention injection (default), self-attention injection into the DiT token sequence, and the VLM update, which edits the cached VLM tokens directly. Cross- and self-attention injection perform comparably, whereas the VLM update drops the success rate to $80.0\,\%$: it is better to keep the cached VLM tokens intact and supply the change as separate tokens than to edit the cached tokens.

Are the difference tokens used? To verify that the action head relies on the difference tokens rather than on the stale VLM tokens alone, we replace them on System-1 queries at test time in two ways. *Zero* feeds all-zero tokens, an input never seen in training. *Identity* feeds the tokens computed with the reference frame as the current frame, $\tilde{h}_\psi(\boldsymbol{F}_r^v, \boldsymbol{F}_r^v)$, which encode "no change"; the action head sees this input in training whenever the reference offset is clamped to 0, so it isolates the information about the change from the effect of an unfamiliar input. With either replacement, the success rate drops to $0\,\%$ on each of the first five LIBERO-10 tasks, so the action head does rely on the difference tokens, even when it is given tokens it has seen in training.

Table 4: **Ablations on LIBERO-10.** The default configuration is shaded.

Factor	Variant	Success (%)
Pooling	CLS	91.6
	Mean	91.2
	Learned query	89.2
	Concat-project	91.3
Injection	Self-attention	92.2
	Cross-attention	91.3
	VLM update	80.0

5 Related Work

Dual-system VLAs (Bjorck et al., 2025; Zhang et al., 2024; Bu et al., 2024) pair a slow VLM (System 2) with a fast action module (System 1). Zhang et al. (2024) and Bu et al. (2024) run the low-level policy at a higher rate than the VLM, conditioned on its latest latent output and the current observation, whereas Differential VLA conditions System 1 on a learned encoding of the *change* relative to the frame that System 2 has seen. Efficient VLA inference methods use smaller backbones (Wen et al., 2024; Shukor et al., 2025), exit the VLM early (Yue et al., 2024), prune its layers and visual tokens (Yang et al., 2025), or reuse cached visual tokens (Xu et al., 2025), but still evaluate (part of) the VLM at every query. Differential VLA evaluates the VLM only once every T control steps, that is, at one in every T/n queries, and is complementary to these techniques. Differential coding, the design principle behind Differential VLA, has also been used to accelerate video models (O'Connor & Welling, 2017; Habibian et al., 2021; Wu et al., 2018) and to tokenize video for understanding and generation (Wang et al., 2026). However, to our knowledge, Differential VLA is the first to apply this principle to VLAs. Appendix A discusses related work in more detail.

6 Limitations and Conclusion

Limitations. First, we have implemented and evaluated Differential VLA only on NVIDIA's GR00T N1.7 (Bjorck et al., 2025). The idea does not depend on this particular dual-system design, but whether it carries over to other VLA architectures remains to be verified experimentally. Second, Differential VLA uses a fixed reference interval T. Scenes that change abruptly, such as when an object is dropped or a human intervenes, may call for refreshing the reference earlier, and adaptive reference frame updates are a natural extension. Third, the wrist view decorrelates much faster than the agent view (Fig. 2), so after a long interval its difference tokens must summarize a large change in a single token. Allocating the difference tokens unevenly across views, for example more to the wrist view than to the agent view, may address this.

Conclusion. We introduced Differential VLA, a VLA that encodes a reference frame with its expensive VLM only once every T steps and represents the frames in between by a few learned difference tokens. Drawing on differential coding, predictive coding, and modern video compression techniques, this design lets the fast action head act on up-to-date visual information while amortizing the cost of the VLM. On LIBERO and on two real-world tasks, Differential VLA stays close to the success rate of GR00T N1.7 while calling the VLM $8\times$ less often and lowering the amortized inference latency by up to $1.27\times$ relative to an equally optimized baseline.

Two settings would make this trade-off even more favorable. First, since the per-step cost of the differential encoder and the action head does not depend on the VLM, the relative savings grow with the size of the VLM, making larger and more capable VLMs practical for closed-loop control. Second, Differential VLA naturally supports deploying System 2 on a remote server while System 1 and the lightweight differential encoder run on the robot: the robot has to send a reference frame to the server, and wait for the network round trip, only once every T steps. Offloading the VLM in this way could substantially reduce the hardware cost of the robot.

AI USE STATEMENT

We used generative AI tools (an LLM-based coding assistant) to help implement and debug parts of the code base, and to draft and edit parts of the manuscript text, including the LaTeX layout of tables, and to assist with the literature search. All research ideas, experimental design, and results are by the authors. We have reviewed all AI-assisted work: every citation was checked against the original publication, and all reported numbers come from our own experiment logs or from the cited sources. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The model architecture is described in Section 3, the training and evaluation protocols in Section 4.1 and in the per-benchmark subsections of Section 4, and all hyperparameters are listed in Appendix C. The inference procedure is given in Algorithm 1. We will release the code, built on the open-source GR00T code base, and the fine-tuned checkpoints.

REFERENCES

Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, Prithvijit Chattopadhyay, Yongxin Chen, Yin Cui, Yifan Ding, Daniel Dworakowski, Jiaojiao Fan, Michele Fenzi, Francesco Ferroni, Sanja Fidler, Dieter Fox, Songwei Ge, Yunhao Ge, Jinwei Gu, Siddharth Gururani, Ethan He, Jiahui Huang, Jacob Samuel Huffman, Pooya Jannaty, Jingyi Jin, Seung Wook Kim, Gergely Klár, Grace Lam, Shiyi Lan, Laura Leal-Taixé, Anqi Li, Zhaoshuo Li, Chen-Hsuan Lin, Tsung-Yi Lin, Huan Ling, Ming-Yu Liu, Xian Liu, Alice Luo, Qianli Ma, Hanzi Mao, Kaichun Mo, Arsalan Mousavian, Seungjun Nah, Sriharsha Niverty, David Page, Despoina Paschalidou, Zeeshan Patel, Lindsey Pavao, Morteza Ramezanali, Fitsum Reda, Xiaowei Ren, Vasanth Rao Naik Sabavat, Ed Schmerling, Stella Shi, Bartosz Stefaniak, Shitao Tang, Lyne Tchapmi, Przemek Tredak, Wei-Cheng Tseng, Jibin Varghese, Hao Wang, Haoxiang Wang, Heng Wang, Ting-Chun Wang, Fangyin Wei, Xinyue Wei, Jay Zhangjie Wu, Jiashu Xu, Wei Yang, Yen-Chen Lin, Xiaohui Zeng, Yu Zeng, Jing Zhang, Qinsheng Zhang, Yuxuan Zhang, Qingqing Zhao, and Artur Zólkowski. Cosmos world foundation model platform for physical AI. *CoRR*, abs/2501.03575, 2025.

Jason Ansel, Edward Z. Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, Chi-Keung Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In Rajiv Gupta, Nael B. Abu-Ghazaleh, Madan Musuvathi, and Dan Tsafrir (eds.), *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, pp. 929–947. ACM, 2024. doi: 10.1145/3620665.3640366.

Harry G. Barrow, Jay M. Tenenbaum, Robert C. Bolles, and Helen C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In Raj Reddy (ed.), *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, USA, August 22-25, 1977*, pp. 659–663. William Kaufmann, 1977.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith LLontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng,

and Yuke Zhu. GR00T N1: an open foundation model for generalist humanoid robots. *CoRR*, abs/2503.14734, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. *CoRR*, abs/2410.24164, 2024.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael S. Ryoo, Grecia Salazar, Pannag R. Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong T. Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: robotics transformer for real-world control at scale. In Kostas E. Bekris, Kris Hauser, Sylvia L. Herbert, and Jingjin Yu (eds.), *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023.

Qingwen Bu, Hongyang Li, Li Chen, Jisong Cai, Jia Zeng, Heming Cui, Maoqing Yao, and Yu Qiao. Towards synergistic, generalized, and efficient dual-system for robotic manipulation. *CoRR*, abs/2410.08001, 2024.

Remi Cadene, Simon Alibert, Alexander Soare, Quentin Gallouedec, Adil Zouitine, Steven Palma, Pepijn Kooijmans, Michel Aractingi, Mustafa Shukor, Dana Aubakirova, Martino Russi, Francesco Capuano, Caroline Pascal, Jade Choghari, Khalil Meftah, Maxime Ellerbach, Jess Moss, and Thomas Wolf. Lerobot: State-of-the-art machine learning for real-world robotics in pytorch. `https://github.com/huggingface/lerobot`, 2024.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *Int. J. Robotics Res.*, 44(10-11):1684–1704, 2025. doi: 10.1177/02783649241273668. URL `https://doi.org/10.1177/02783649241273668`.

Cassius C Cutler. Differential quantization of communication signals, July 29 1952. US Patent 2,605,361.

Peter Elias. Predictive coding-i. *IRE Trans. Inf. Theory*, 1(1):16–24, 1955.

Karl Friston and Stefan Kiebel. Predictive coding under the free-energy principle. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1521):1211–1221, 05 2009. ISSN 0962-8436. doi: 10.1098/rstb.2008.0300.

Didier Le Gall. MPEG: A video compression standard for multimedia applications. *Commun. ACM*, 34(4):46–58, 1991.

Dibya Ghosh, Homer Rich Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Quan Vuong, Ted Xiao, Pannag R. Sanketi, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In Dana Kulic, Gentiane Venture, Kostas E. Bekris, and Enrique Coronado (eds.), *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, 2024.

Amirhossein Habibian, Davide Abati, Taco S. Cohen, and Babak Ehteshami Bejnordi. Skip-convolutions for efficient video processing. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pp. 2695–2704. Computer Vision Foundation / IEEE, 2021.

Daniel Kahneman. *Thinking, fast and slow*. Farrar, Straus and Giroux, New York, 2011. ISBN 9780374275631 0374275637.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Paul Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Conference on Robot Learning, 6-9 November 2024, Munich, Germany*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2024.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3744–3753. PMLR, 2019.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Conference on Robot Learning, 6-9 November 2024, Munich, Germany*, volume 270 of *Proceedings of Machine Learning Research*, pp. 3705–3728. PMLR, 2024.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: benchmarking knowledge transfer for lifelong robot learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

William Lotter, Gabriel Kreiman, and David D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Peter O'Connor and Max Welling. Sigma delta quantized networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alexander Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew E. Wang, Anikait Singh, Animesh Garg, Aniruddha Kembhavi, Annie Xie, Anthony Brohan, Antonin Raffin, Archit Sharma, Arefeh Yavary, Arhan Jain, Ashwin Balakrishna, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Blake Wulfe, Brian Ichter, Cewu Lu, Charles Xu, Charlotte Le, Chelsea Finn, Chen Wang, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Christopher Agia, Chuer Pan, Chuyuan Fu, Coline Devin, Danfei Xu, Daniel Morton, Danny Driess, Daphne Chen, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dinesh Jayaraman, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Ethan Paul Foster, Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guanzhi Wang, Hao Su, Haoshu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I. Christensen, Hiroki Furuta, Homer Walke, Hongjie Fang, Huy Ha, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jaimyn Drake, Jan Peters, Jan Schneider, Jasmine Hsu, Jeannette Bohg, Jeffrey T. Bingham, Jeffrey Wu, Jensen Gao, Jiaheng Hu, Jiajun Wu, Jialin Wu, Jiankai Sun, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jimmy Wu, Jingpei Lu, Jingyun Yang, Jitendra Malik, João Silvério, Joey Hejna, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Jordi Salvador, Joseph J. Lim, Junhyek Han, Kaiyuan Wang, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg,

Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Black, Kevin Lin, Kevin Zhang, Kiana Ehsani, Kiran Lekkala, Kirsty Ellis, Krishan Rana, Krishnan Srinivasan, Kuan Fang, Kunal Pratap Singh, Kuo-Hao Zeng, Kyle Hatch, Kyle Hsu, Laurent Itti, Lawrence Yunliang Chen, Lerrel Pinto, Li Fei-Fei, Liam Tan, Linxi Jim Fan, Lionel Ott, Lisa Lee, Luca Weihs, Magnum Chen, Marion Lepert, Marius Memmel, Masayoshi Tomizuka, Masha Itkina, Mateo Guaman Castro, Max Spero, Maximilian Du, Michael Ahn, Michael C. Yip, Mingtong Zhang, Mingyu Ding, Minho Heo, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J. Joshi, Niko Sünderhauf, Ning Liu, Norman Di Palo, Nur Muhammad (Mahi) Shafiullah, Oier Mees, Oliver Kroemer, Osbert Bastani, Pannag R. Sanketi, Patrick Tree Miller, Patrick Yin, Paul Wohlhart, Peng Xu, Peter David Fagan, Peter Mitrano, Pierre Sermanet, Pieter Abbeel, Priya Sundaresan, Qiuyu Chen, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Martín-Martín, Rohan Baijal, Rosario Scalise, Rose Hendrix, Roy Lin, Runjia Qian, Ruohan Zhang, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante-Gomez, Sean Kirmani, Sergey Levine, Shan Lin, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham D. Sonawani, Shuran Song, Sichun Xu, Siddhant Haldar, Siddharth Karamcheti, Simeon Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Subramanian Ramamoorthy, Sudeep Dasari, Suneel Belkhale, Sungjae Park, Suraj Nair, Suvir Mirchandani, Takayuki Osa, Tanmay Gupta, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Thomas Kollar, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Trinity Chung, Vidhi Jain, Vincent Vanhoucke, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiaolong Wang, Xinghao Zhu, Xinyang Geng, Xiyuan Liu, Liangwei Xu, Xuanlin Li, Yao Lu, Yecheng Jason Ma, Yejin Kim, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Yilin Wu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yue Cao, Yueh-Hua Wu, Yujin Tang, Yuke Zhu, Yunchu Zhang, Yunfan Jiang, Yunshuang Li, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zehan Ma, Zhuo Xu, Zichen Jeff Cui, Zichen Zhang, and Zipeng Lin. Open x-embodiment: Robotic learning datasets and RT-X models : Open x-embodiment collaboration. In *IEEE International Conference on Robotics and Automation, ICRA 2024, Yokohama, Japan, May 13-17, 2024*, pp. 6892–6903. IEEE, 2024.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision. *Trans. Mach. Learn. Res.*, 2024, 2024.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*, pp. 4172–4182. IEEE, 2023.

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. *CoRR*, abs/2501.09747, 2025.

Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachy Groom, Hunter Hancock, Karol Hausman, Gashon Hussein, Brian Ichter, Szymon Jakubczak, Rowan Jen, Tim Jones, Ben Katz, Liyiming Ke, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Yao Lu, Vishnu Mano, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Charvi Sharma, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Alex Swerdlow, James Tanner, Marcel Torne, Quan Vuong, Anna Walling, Haohuan Wang, Blake Williams, Sukwon Yoo, Lili Yu, Ury Zhilinsky, and Zhiyuan Zhou. $\pi^*_{0.6}$: a VLA that learns from experience. *CoRR*, abs/2511.14759, 2025a.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z.

Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *CoRR*, abs/2504.16054, 2025b.

Physical Intelligence, Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, Vedant Choudhary, Foster Collins, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Maitrayee Dhaka, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachlan Groom, Haroun Habeeb, Hunter Hancock, Karol Hausman, Gashon Hussein, Victor Hwang, Brian Ichter, Connor Jacobsen, Szymon Jakubczak, Rowan Jen, Tim Jones, Gregg Kammerer, Ben Katz, Liyiming Ke, Mairbek Khadikov, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Brendon LeCount, Sergey Levine, Xinyu Li, Adrian Li-Bell, Vladislav Lialin, Zhonglin Liang, Wallace Lim, Yao Lu, Enyu Luo, Vishnu Mano, Nandan Marwaha, Aikys Mongush, Liam Murphy, Suraj Nair, Tyler Patterson, Karl Pertsch, Allen Z. Ren, Gavin Schelske, Charvi Sharma, Baifeng Shi, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Jiaming Tang, Jimmy Tanner, Shalom Tekeste, Marcel Torne, Kyle Vedder, Quan Vuong, Anna Walling, Haohuan Wang, Jason Wang, XuDong Wang, Chris Whalen, Samuel Whitmore, Blake Williams, Charles Xu, Sukwon Yoo, Lili Yu, Wuming Zhang, Zhuoyang Zhang, and Ury Zhilinsky. $\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities. *CoRR*, abs/2604.15483, 2026.

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

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andrés Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Rémi Cadène. Smolvla: A vision-language-action model for affordable and efficient robotics. *CoRR*, abs/2506.01844, 2025.

Gary J. Sullivan, Jens-Rainer Ohm, Woojin Han, and Thomas Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Trans. Circuits Syst. Video Technol.*, 22(12): 1649–1668, 2012.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017.

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, Abraham Lee, Kuan Fang, Chelsea Finn, and Sergey Levine. Bridgedata V2: A dataset for robot learning at scale. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1723–1736. PMLR, 2023.

Feng Wang, Yichun Shi, Ceyuan Yang, Qiushan Guo, Jingxiang Sun, Alan L. Yuille, and Peng Wang. Vtok: A unified video tokenizer with decoupled spatial-temporal latents. *CoRR*, abs/2602.04202, 2026.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, Yaxin Peng, Feifei Feng, and Jian Tang. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *CoRR*, abs/2409.12514, 2024.

Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegaard, and Ajay Luthra. Overview of the H.264/AVC video coding standard. *IEEE Trans. Circuits Syst. Video Technol.*, 13(7):560–576, 2003.

Chao-Yuan Wu, Manzil Zaheer, Hexiang Hu, R. Manmatha, Alexander J. Smola, and Philipp Krähenbühl. Compressed video action recognition. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 6026–6035. Computer Vision Foundation / IEEE Computer Society, 2018.

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 Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025.

Yantai Yang, Yuhao Wang, Zichen Wen, Luo Zhongwei, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. Efficientvla: Training-free acceleration and compression for vision-language-action models. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025.

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 Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

Jianke Zhang, Yanjiang Guo, Xiaoyu Chen, Yen-Jen Wang, Yucheng Hu, Chengming Shi, and Jianyu Chen. Hirt: Enhancing robotic control with hierarchical robot transformers. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Conference on Robot Learning, 6-9 November 2024, Munich, Germany*, volume 270 of *Proceedings of Machine Learning Research*, pp. 933–946. PMLR, 2024.

Junyi Zhang, Charles Herrmann, Junhwa Hur, Luisa Polania Cabrera, Varun Jampani, Deqing Sun, and Ming-Hsuan Yang. A tale of two features: Stable diffusion complements DINO for zero-shot semantic correspondence. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In Kostas E. Bekris, Kris Hauser, Sylvia L. Herbert, and Jingjin Yu (eds.), *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023.

A Extended Related Work

Vision-language-action models. Building on large-scale robot datasets (Brohan et al., 2023; O'Neill et al., 2024), VLA models fine-tune pretrained VLMs to output actions, either as discrete tokens (Kim et al., 2024; Pertsch et al., 2025) or through a continuous action head based on diffusion or flow matching (Black et al., 2024; Bjorck et al., 2025), as in earlier diffusion-based robot policies (Chi et al., 2025; Ghosh et al., 2024). Dual-system architectures (Bjorck et al., 2025; Zhang et al., 2024; Bu et al., 2024) pair a slow VLM (System 2) with a fast action module (System 1). Zhang et al. (2024) and Bu et al. (2024) let a lightweight low-level policy run at a higher rate than the VLM, conditioned on its latest latent output together with the current observation. Differential VLA shares this asynchronous structure, but it conditions System 1 on an explicit, learned encoding of the *change* relative to the frame that System 2 has seen, instead of an independent encoding of the current observation.

Efficient VLA inference. Other works reduce the cost of VLA inference by using smaller backbones (Wen et al., 2024; Shukor et al., 2025), by adaptively exiting the VLM early (Yue et al., 2024), by pruning redundant language-model layers and task-irrelevant visual tokens while caching intermediate features of the action head across denoising steps (Yang et al., 2025), by compressing the

Algorithm 1 Differential VLA inference for one episode

Require: instruction ℓ, reference interval T, number of executed actions per query n

$s \leftarrow T$ ▷ steps since the last reference frame
while episode not done **do**
  observe $\mathbf{O}_t, \boldsymbol{q}_t$; $\boldsymbol{F}_c^v \leftarrow E(\mathbf{O}_t^v) + \boldsymbol{P}$ ▷ each view v; cheap, every query
  if $s \geq T$ **then** ▷ System 2
    $\bar{\boldsymbol{Z}} \leftarrow f_\phi(\mathbf{O}_t, \ell)$; $\boldsymbol{F}_r^v \leftarrow \boldsymbol{F}_c^v$; $s \leftarrow 0$
  end if
  $\boldsymbol{U}_t^v \leftarrow \tilde{h}_\psi(\boldsymbol{F}_c^v, \boldsymbol{F}_r^v)$; $\boldsymbol{U}_t \leftarrow [\boldsymbol{U}_t^1; \boldsymbol{U}_t^2; \cdots; \boldsymbol{U}_t^V]$ ▷ difference tokens
  $\boldsymbol{A}_t \leftarrow$ denoise with 4 steps as in Eq. (1) while feeding $[\bar{\boldsymbol{Z}}; \boldsymbol{U}_t]$ instead of $\boldsymbol{Z}_t$ ▷ System 1
  execute $\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, \ldots, \boldsymbol{a}_{t+n-1}$; $s \leftarrow s + n$; $t \leftarrow t + n$
end while

action representation (Pertsch et al., 2025), or by reusing the cached computation of visual tokens that did not change between steps (Xu et al., 2025). These approaches still evaluate at least part of the VLM at every policy query, whereas Differential VLA evaluates it only once every T control steps, or, equivalently, once every T/n queries. Moreover, Differential VLA is complementary to most of these techniques: they reduce the cost of each VLM call, whereas Differential VLA reduces the number of calls.

Differential and predictive coding. Differential coding (Cutler, 1952; Elias, 1955) transmits the error of a prediction instead of the signal itself, and its extension to video, inter-frame coding with I- and P-frames (Gall, 1991; Wiegand et al., 2003; Sullivan et al., 2012), is the basis of most modern video codecs. In neuroscience and cognitive science, predictive coding (Friston & Kiebel, 2009) posits that the brain propagates only the errors of predictions made by an internal generative model, an idea that has also inspired neural network architectures for video prediction (Lotter et al., 2017). In deep learning, temporal redundancy has been exploited to accelerate video models by processing only the changes between frames (O'Connor & Welling, 2017; Habibian et al., 2021), by merging redundant spatio-temporal tokens (Bolya et al., 2023), by operating directly on the compressed video (Wu et al., 2018), and by tokenizing videos as a key frame and its subsequent motion for both understanding and generation (Wang et al., 2026). To our knowledge, Differential VLA is the first to apply this principle to the perception stack of VLA policies, where the reference frame is encoded by a large VLM and the change is encoded by a lightweight differential encoder trained end-to-end with the control objective.

B Inference Algorithm

Algorithm 1 gives the inference procedure of Differential VLA described in Section 3.5. The cache stores the VLM tokens $\bar{\boldsymbol{Z}}$ and the DINOv2 tokens $\boldsymbol{F}_r^v$ of the reference frame, so the reference branch of the differential encoder is not recomputed. The counter s advances in environment steps rather than policy queries.

C Hyperparameters

Table 5 lists the hyperparameters of Differential VLA for the simulation benchmarks. Those of the real-world experiments are given in Appendix H.

D FLOP Accounting

Table 6 breaks down the per-query FLOP estimates of Table 3. We count every module with the primitives of NVIDIA's FLOP estimator for Cosmos models[4], which count two FLOPs per multiply-accumulate operation in every matrix multiplication, including the attention score and score-times-value products, together with biases, softmax, normalizations, and activations. For the VLM we use

[4] https://github.com/NVIDIA/cosmos-framework

Table 5: **Hyperparameters** of Differential VLA. Color jitter is applied to the inputs of the VLM and the differential encoder.

	LIBERO	SimplerEnv (WidowX)
Base model	GR00T N1.7 (3B)	
VLM	Cosmos-Reason2-2B (16 LLM layers), frozen	
Action head	32-layer DiT, width 1536, 4 denoising steps	
Action chunk length	16	8
Actions executed per query n	8	4
Camera views V	2 (agent, wrist)	1 (third-person)
Frame encoder	DINOv2-small ($d = 384$, $P = 257$), fine-tuned	
Cross-attention	1 layer, 6 heads, pos. emb., residual + LN	
Pooling	concat-project	
Difference tokens per view m	1	
Injection	cross-attention (as image tokens)	
Reference interval T	64	
Optimizer	AdamW	
Learning rate	10^{-4}, 5 % warm-up	
Weight decay	10^{-5}	
Batch size	640	1024
Training steps	20k	
State dropout	0.2	0.8
Color jitter (b, c, s, h)	0.3, 0.4, 0.5, 0.08	
Hardware	2× NVIDIA H200	

Table 6: **Per-query FLOP breakdown.** The VLM block runs only on System-2 queries; every other block runs on every query. The baseline runs the VLM and the action head on every query; its action head costs 461.0 GFLOPs, since its context lacks the 2 difference tokens.

Module	Shape	Tokens	GFLOPs
VLM backbone (System-2 queries only)			
Qwen3-VL vision tower	24 layers, h=1024, i=4096	2×256	330.6
LLM (first 16 of 28 layers)	h=2048, i=6144, GQA 16/8	149	241.6
Subtotal			*572.2*
Differential encoder (every query)			
DINOv2-small, 2 views	12 layers, d=384, i=1536	2×257	24.6
Cross-attention, 2 views	1 layer, 6 heads, d=384	2×257	0.8
Concat-project pooler	$2048 \times 98{,}688$	2×1	0.8
Subtotal			*26.2*
Action head (every query)			
VL self-attention	4 layers, width 2048	151	61.6
DiT, 4 denoising steps	32 layers, width 1536	41	401.9
Subtotal			*463.5*
Differential VLA, System-2 query	VLM + encoder + head		1061.9
Differential VLA, System-1 query	encoder + head		489.7
Differential VLA, amortized (T=64, n=8)			561.2
GR00T N1.7, every query	VLM + head		1033.2

the estimator's Qwen3-VL functions directly, with the vision tower attending within each image as in Qwen3-VL. All counts are for batch size 1 with the shapes the deployed model runs: two camera views, resized to 256×256 for the VLM and to 224×224 for DINOv2; a 149-token VLM sequence (128 image tokens after the 2×2 spatial merge, plus 21 text tokens, which is the text token count per frame averaged over the four LIBERO suites); 257 DINOv2 tokens per view, which the pooler flattens ($257 \times 384 = 98\,688$) and projects to the LLM width of 2048; and a 40-step action chunk (the checkpoint's fixed horizon, of which only the first 16 actions are used on LIBERO) denoised in 4 steps. Before the DiT, the action head passes the VLM tokens and, for Differential VLA, the 2 difference tokens through a 4-layer self-attention transformer. Because the difference tokens enter this transformer, it must run on every query instead of being cached with the VLM output. The DiT sequence consists of one state token and 40 action tokens, and its 16 cross-attention blocks attend to the same 151 tokens. The DiT row also includes the action encoder and decoder. The action-head shapes and the LLM truncation are those of the released GR00T N1.7 checkpoint, which our fine-tuned models inherit. The script that produces Table 6 is included in the supplementary material.

E Compiled Execution

The latencies in Table 3 are all measured in eager PyTorch, so that the comparison between Differential VLA and the baseline varies only the architecture. Ahead-of-time compilation changes the absolute numbers substantially. Compiling the baseline with TensorRT brings a query from 31.5 ms down to 20.9 ms (95th percentile 21.0 ms) on the same RTX 4090 with batch size 1 over 3200 queries, a $1.51\times$ speed-up, which is faster than the 24.9 ms that Differential VLA achieves amortized in eager mode.

We do not report a compiled number for Differential VLA. The TensorRT export path of GR00T N1.7 reimplements the action head with its conditioning fixed to the VLM tokens, so a compiled build would drop the difference tokens. Supporting Differential VLA requires extending the exporter to the differential encoder and the extended token sequence.

Table 7: **Success rates (%) on SimplerEnv, WidowX.** Freq. is the average number of VLM calls per environment step. Our eval. is the official checkpoint evaluated under our protocol.

Method	Freq.	0	1	2	3	4	5	6	Avg.
N1.7 (reported)	1/4	78.0	58.0	48.0	53.0	2.0	100.0	97.0	62.3
N1.7 (our eval.)	1/4	79.2	65.6	37.5	54.2	1.0	100.0	91.3	61.3
N1.7 (n=8)	1/8	78.1	69.8	37.5	28.1	1.0	100.0	99.0	59.1
N1.7 + cache (T=64)	1/64	1.0	5.2	0.0	1.0	0.0	38.0	49.0	13.5
Differential VLA (T=64)	1/64	53.1	31.3	29.2	47.9	45.5	76.0	100.0	54.7

This does not affect the $1.8\times$ reduction in operations, which is a property of the architecture, nor the $1.27\times$ latency reduction against a baseline optimized to the same degree. Against a compiled baseline, however, our current eager implementation is slower in wall-clock time, and closing this gap requires engineering work rather than a change to the method.

F SimplerEnv Results

SimplerEnv (Li et al., 2024) evaluates policies trained on real-robot data in simulated replicas of real setups. We use its WidowX setting, for which policies are trained on BridgeData V2 (Walke et al., 2023). We train with a batch size of 1024 for 20k steps, predict 8-step chunks, and execute $n = 4$ actions per query. Each task is evaluated for the multiple of its number of initial configurations that is closest to 100 episodes, with a limit of 300 steps; with parallel evaluation workers, a few runs exceed this count, and we keep all completed episodes. The baselines are as in Table 1: N1.7 (n=8) executes each 8-step chunk in full, and N1.7 + cache reuses the cached VLM tokens for $T = 64$ steps without difference tokens. For the baseline, we evaluate the official Bridge-fine-tuned GR00T N1.7 checkpoint under our protocol; the reported numbers are from the official Isaac-GR00T repository[5]. Results are shown in Table 7, where tasks are identified by their index. Appendix G lists the corresponding task descriptions.

Differential VLA reaches an average success rate of $54.7\,\%$, 6.6 points below the official checkpoint under our evaluation while calling the VLM $16\times$ less often. As on LIBERO, reusing the cached VLM tokens without difference tokens largely fails ($13.5\,\%$). Halving the VLM calls of the baseline by executing whole chunks already costs 2.2 points, mostly on putting the eggplant in the basket. The gap to the baseline is concentrated on the spoon, carrot, and drawer-opening tasks, whereas on putting the eggplant in the sink, where every baseline, including the reported one, succeeds at most $2\,\%$ of the time, Differential VLA reaches $45.5\,\%$.

G Task Indices

Tables 8 to 12 list the task index used in the per-task tables together with the corresponding task. For LIBERO the index is the position of the task in the suite's evaluation task list of the official Isaac-GR00T repository[6]; the task string is also the language instruction given to the policy. For SimplerEnv the index is the position of the task in the benchmark's task list, and the name is the SimplerEnv environment identifier.

H Real-World Experiment Details

Tasks. The two SO-101 tasks of Section 4.4 are shown in Fig. 4.

[5] https://github.com/NVIDIA/Isaac-GR00T/blob/51d4c89f72fda44cbf77285c6a8114b52676b8a1/examples/SimplerEnv/README.md

[6] https://github.com/NVIDIA/Isaac-GR00T/blob/51d4c89f72fda44cbf77285c6a8114b52676b8a1/examples/LIBERO/README.md

Table 8: **LIBERO-Spatial task indices.** The task string is the language instruction of the corresponding task.

Index	Task
0	pick up the black bowl between the plate and the ramekin and place it on the plate
1	pick up the black bowl next to the ramekin and place it on the plate
2	pick up the black bowl from table center and place it on the plate
3	pick up the black bowl on the cookie box and place it on the plate
4	pick up the black bowl in the top drawer of the wooden cabinet and place it on the plate
5	pick up the black bowl on the ramekin and place it on the plate
6	pick up the black bowl next to the cookie box and place it on the plate
7	pick up the black bowl on the stove and place it on the plate
8	pick up the black bowl next to the plate and place it on the plate
9	pick up the black bowl on the wooden cabinet and place it on the plate

Table 9: **LIBERO-Object task indices.** The task string is the language instruction of the corresponding task.

Index	Task
0	pick up the alphabet soup and place it in the basket
1	pick up the cream cheese and place it in the basket
2	pick up the salad dressing and place it in the basket
3	pick up the bbq sauce and place it in the basket
4	pick up the ketchup and place it in the basket
5	pick up the tomato sauce and place it in the basket
6	pick up the butter and place it in the basket
7	pick up the milk and place it in the basket
8	pick up the chocolate pudding and place it in the basket
9	pick up the orange juice and place it in the basket

Table 10: **LIBERO-Goal task indices.** The task string is the language instruction of the corresponding task.

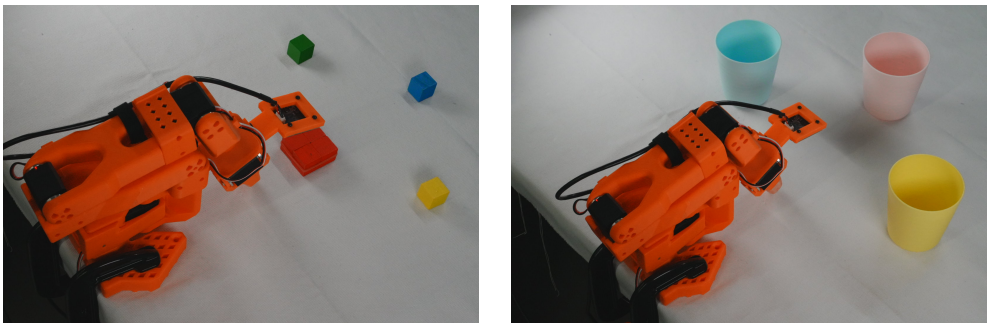
Index	Task
0	open the middle drawer of the cabinet
1	put the bowl on the stove
2	put the wine bottle on top of the cabinet
3	open the top drawer and put the bowl inside
4	put the bowl on top of the cabinet
5	push the plate to the front of the stove
6	put the cream cheese in the bowl
7	turn on the stove
8	put the bowl on the plate
9	put the wine bottle on the rack

Table 11: **LIBERO-Long (LIBERO-10) task indices.** The task string is the language instruction of the corresponding task.

Index	Task
0	put both the alphabet soup and the tomato sauce in the basket
1	put both the cream cheese box and the butter in the basket
2	turn on the stove and put the moka pot on it
3	put the black bowl in the bottom drawer of the cabinet and close it
4	put the white mug on the left plate and put the yellow and white mug on the right plate
5	pick up the book and place it in the back compartment of the caddy
6	put the white mug on the plate and put the chocolate pudding to the right of the plate
7	put both the alphabet soup and the cream cheese box in the basket
8	put both moka pots on the stove
9	put the yellow and white mug in the microwave and close it

Table 12: SimplerEnv task indices

Index	WidowX (Bridge) Task
0	spoon on towel
1	carrot on plate
2	stack cube
3	put eggplant in basket
4	put eggplant in sink
5	open drawer
6	close drawer



(a) Block placing (b) Cup stacking

Figure 4: **Real-world tasks on SO-101.** (a) The blue cube is to be placed on the red target, which is made of four red blocks fastened together. The green and yellow cubes are distractors. (b) The three cups are to be stacked into a single stack.

Table 13: **Language instructions of the real-world SO-101 tasks.**

Task	Instruction
Block placing	`Pick up the blue block and place it on the red blocks.`
Cup stacking	`Stack the three cups.`

- **Block placing**: pick up a blue wooden cube and place it on a red target. To keep the task feasible for the SO-101, the target is formed by fastening four red blocks together into a larger square. A green and a yellow cube are placed on the table as distractors. The task requires identifying the correct cube, grasping it, and placing it accurately relative to the target.
- **Cup stacking**: stack three cups placed on the table into a single stack. The cup placements constantly change throughout the episode, requiring the policy to keep track of progress between reference frames.

Training. Both methods are fine-tuned with the same recipe (batch size 128, 30k steps, cosine learning-rate schedule) on 4 RTX A6000 GPUs with a batch size of 32 per GPU. For Differential VLA, we accumulate gradients over 4 micro-batches of 8 to fit in memory.

Initial configurations. Rather than sampling the initial configurations of the SO-101 tasks of Section 4.4 at random, we enumerate them systematically. For block placing, we fix four candidate positions and place the four objects (the blue cube, the red target, and the two distractors) in all $4! = 24$ assignments. For cup stacking, we fix four sets of three candidate positions and place the three cups in every one of their $3! = 6$ orders, which gives $4 \times 6 = 24$ configurations. Both policies are run on every configuration, so the trials are paired, and within each pair we took care to reproduce the object placement as closely as possible so that both policies start from the same initial condition.

Language instructions. Table 13 gives the language instruction that is passed to the policy on each of the two SO-101 tasks of Section 4.4. The same string is used for every demonstration of the task and for every evaluation trial, so it is constant within a task and carries no per-episode information.

Latency breakdown. The per-query latency on the SO-101 tasks follows the same pattern as in Section 4.5: a System-1 query takes $41\,\mathrm{ms}$ to $42\,\mathrm{ms}$ and a System-2 query $63\,\mathrm{ms}$ to $64\,\mathrm{ms}$, so System-2 queries are slower than a baseline query and the saving comes entirely from the 7 out of 8 queries that skip the VLM. End-to-end round-trip latency, which adds $32\,\mathrm{ms}$ to $35\,\mathrm{ms}$ of network

Table 14: **Per-task success rates (%) on LIBERO.** Columns are the task indices of Tables 8 to 11. GR00T N1.7 is the official checkpoint under our evaluation, and Differential VLA uses $T{=}64$, as in Table 1.

Suite	Method	0	1	2	3	4	5	6	7	8	9	Avg.
Spatial	N1.7	100.0	100.0	100.0	99.0	95.0	62.0	100.0	100.0	96.0	99.0	95.1
	N1.7 ($n{=}16$)	96.0	99.0	99.0	98.0	95.0	60.0	100.0	100.0	94.0	97.0	93.8
	Differential VLA	100.0	98.0	99.0	100.0	93.0	77.0	100.0	96.0	95.0	99.0	95.7
Object	N1.7	98.0	99.0	99.0	98.0	97.0	99.0	100.0	100.0	98.0	100.0	98.8
	N1.7 ($n{=}16$)	98.0	100.0	96.0	95.0	98.0	99.0	97.0	96.0	99.0	98.0	97.6
	Differential VLA	100.0	100.0	98.0	98.0	100.0	100.0	100.0	98.0	100.0	100.0	99.4
Goal	N1.7	98.0	95.0	94.0	92.0	99.0	98.0	92.0	100.0	99.0	95.0	96.2
	N1.7 ($n{=}16$)	98.0	98.0	90.0	92.0	98.0	94.0	91.0	100.0	99.0	94.0	95.4
	Differential VLA	86.0	83.0	98.0	95.0	82.0	95.0	90.0	100.0	53.0	95.0	87.7
Long	N1.7	94.0	98.0	98.0	96.0	86.0	92.0	86.0	90.0	66.0	74.0	88.0
	N1.7 ($n{=}16$)	100.0	94.0	94.0	98.0	90.0	94.0	88.0	84.0	70.0	90.0	90.2
	Differential VLA	93.0	98.0	96.0	100.0	90.0	97.0	83.0	92.0	71.0	93.0	91.3

transfer and serialization common to both methods, drops from $87.5\,\mathrm{ms}$ to $79.0\,\mathrm{ms}$ on block placing and from $87.6\,\mathrm{ms}$ to $78.9\,\mathrm{ms}$ on cup stacking.

I Per-Task Results

Table 14 breaks the LIBERO results of Table 1 down by task. Task indices refer to Tables 8 to 11; each entry is the success rate over 100 episodes, twice on each of the task's 50 initial states.