# ASSIST-VLA: ACCELERATING VISION-LANGUAGE-ACTION INFERENCE VIA MULTI-LEVEL REDUNDANCY REDUCTION

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

## ABSTRACT

Vision-language-action (VLA) models enable robots to translate visual observations and language instructions into actions, but repeated multimodal inference incurs substantial latency during closed-loop control. We introduce ASSIST-VLA, a training-free framework that accelerates VLA inference by reducing redundancy at three levels: control cycles, visual inputs, and model structure. At the control-cycle level, consistency across successive action plans guides the reuse of cached cognition representations, allowing selected steps to bypass visual encoding and language-backbone inference. At the visual-input level, instruction-conditioned semantic relevance and local spatiotemporal cues guide token pruning to retain task-relevant information. At the model-structure level, we draw on the J-space perspective, which analyzes internal representations through their influence on model outputs, and extend it to VLA action outputs. By combining Jacobian mappings with channel activation statistics, we assess the influence of MLP channels on predicted actions and structurally prune low-influence channels. Together, these components reduce the frequency and cost of full multimodal inference. On four manipulation tasks in SimplerEnv with CogACT-Base, ASSIST-VLA reduces average end-to-end inference latency from $229.1$ ms to $158.5$ ms per control step, achieving a $1.45\times$ speedup with an average task success rate of $67.8\%$, compared with $67.1\%$ for the original model. These results show that reducing redundancy across multiple levels can accelerate closed-loop VLA inference while maintaining task performance.

## 1 INTRODUCTION

Vision-language-action (VLA) models integrate visual perception, language understanding, and action generation, enabling robots to follow natural-language instructions. RT-1 (Brohan et al., 2022) demonstrated multi-task control through large-scale robot demonstration learning, while RT-2 (Brohan et al., 2023) transferred knowledge from vision-language pretraining to robot action prediction. Subsequent models, including Octo (Octo Model Team et al., 2024), OpenVLA (Kim et al., 2024), and $\pi_{0.5}$ (Physical Intelligence et al., 2025), leverage diverse training data and policy architectures to improve generalization across tasks, objects, and environments, advancing generalist robot policies.

Despite their strong semantic reasoning and generalization capabilities, VLA models remain computationally expensive for online deployment. At every control cycle, a policy must process new visual observations and language instructions, and perform multimodal inference to generate actions. In particular, the internal LLM often accounts for the dominant inference cost, which is jointly affected by the number of visual tokens and the model parameter scale. Reducing redundant computation from both visual inputs and model parameters is therefore crucial for real-time VLA deployment.

Existing efficient-inference methods reduce VLA computation through visual-token compression, cross-timestep computation reuse, and model compression. Visual-token methods shorten the input sequence through selection, pruning, or merging (Ryoo et al., 2021; Rao et al., 2021; Liang et al., 2022; Bolya et al., 2023; Chen et al., 2024), while VLA-specific approaches further use task-related information to select visual tokens (Wang et al., 2026; Ma et al., 2026). Cache-based methods reuse intermediate computation across consecutive control steps (Xu et al., 2025). EfficientVLA (Yang



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

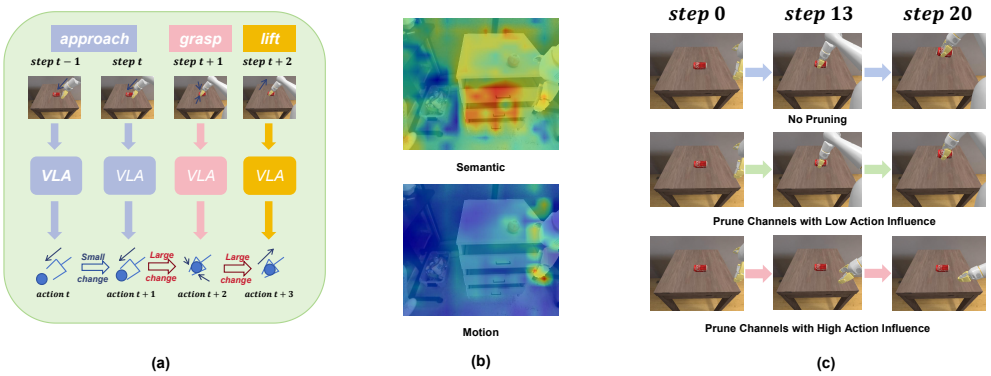



Figure 1: Multi-level redundancy in closed-loop VLA inference: (a) Action-plan consistency across control steps reveals temporal redundancy. (b) Semantic and motion scores identify task-relevant targets and gripper interactions. (c) Low-action-influence MLP channels can be pruned with less execution deviation than high- influence channels.

et al., 2025) combines visual-token selection, language-module compression, and action-head feature reuse, while quantization and structural pruning reduce parameter-level computation (Frantar et al., 2023; Xiao et al., 2023; Lin et al., 2024; Frantar & Alistarh, 2023; Sun et al., 2024; Ma et al., 2023; Xu et al., 2026). Despite covering multiple acceleration directions, most existing methods target a single module or local redundancy source and lack a unified framework that coordinates control-cycle, visual-input, and model-structure redundancy around final action decisions.

Our key observation is that, as illustrated in Figure 1, **redundancy in closed-loop VLA inference arises at the control-cycle, visual-input, and model-structure levels.** At the action level, consecutive action plans are highly consistent during stable phases but change substantially during transitions such as grasping and lifting, enabling action consistency to reduce unnecessary full cognition inference. At the visual level, semantic saliency focuses on the instruction-specified target, while motion saliency highlights the gripper and its interaction region, providing complementary signals for selecting critical visual tokens. At the model-parameter level, under the same pruning budget, removing low-action-influence MLP channels leaves execution behavior nearly unchanged from the unpruned model, whereas removing high-action-influence channels causes clear deviations, indicating removable structural redundancy. These observations show that closed-loop VLA inference contains exploitable redundancy at the control-cycle, visual-input, and model-structure levels, motivating a coordinated acceleration strategy across the entire inference pipeline.

Motivated by these observations, we propose **ASSIST-VLA**, short for **A**ction-autocorrelation, **S**emantic-**S**patiotemporal, and **I**nfluence-aware **S**tructural **T**rimming for efficient **VLA** inference. First, ASSIST-VLA uses action autocorrelation to determine whether the cognition representation from the previous control cycle can be reused, reducing unnecessary full multimodal inference. Second, semantic-spatiotemporal visual-token pruning combines instruction-conditioned semantic relevance, inter-frame similarity, and intra-frame local similarity to compress the visual sequence while preserving task targets and dynamic control cues. Finally, we draw on the J-space perspective (Gurnee et al., 2026), which uses Jacobian mappings to analyze internal representations according to their downstream effects on model outputs, and transfer this output-oriented perspective from language models to VLA control. Specifically, we relate the path from intermediate representations to verbalizable representations and word outputs to the path from intermediate representations through cognition representations to action outputs. This correspondence enables ASSIST-VLA to evaluate the influence of MLP channels on final actions and structurally remove low-influence channels, reducing parameter count and matrix computation.

Our contributions are summarized as follows:

- **Multi-level redundancy modeling and collaborative acceleration.** We model redundancy at three levels: action autocorrelation enables cognition-representation reuse, inter-frame and intra-frame local similarity guides visual-token pruning, and a J-space-inspired

criterion evaluates the influence of MLP neurons on action outputs, thereby lowering the overall computational cost of closed-loop inference.

- **A unified three-module acceleration framework.** We propose ASSIST-VLA, a unified framework that combines action-autocorrelation-based cognition-representation reuse, semantic-spatiotemporal visual-token pruning, and action-influence-aware structural MLP pruning to reduce redundancy at the control-cycle, visual-input, and model-structure levels.
- **Empirical validation of the efficiency–performance trade-off.** Across four manipulation tasks in SimplerEnv with CogACT-Base, ASSIST-VLA achieves a $1.45\times$ end-to-end inference speedup while maintaining an average success rate of $67.8\%$, compared with $67.1\%$ for the baseline. Leave-one-out ablations further quantify the contributions of the three modules to inference efficiency and task performance, validating coordinated multi-level acceleration for closed-loop VLA inference.

## 2 Related Work

### 2.1 Vision-Language-Action Models

Vision-language-action (VLA) models map visual observations and language instructions to robot actions. Early approaches, such as RT-2 (Brohan et al., 2023) and OpenVLA (Kim et al., 2024), discretize actions into tokens and autoregressively predict them with a vision-language backbone. Later architectures increasingly decouple multimodal representation learning from action generation and use diffusion- or flow-matching-based action modules to predict continuous action sequences, as in Octo (Octo Model Team et al., 2024) and $\pi_{0.5}$ (Physical Intelligence et al., 2025). CogACT (Li et al., 2024a) further adopts this design by using a vision-language backbone to produce a cognition representation that conditions a diffusion action head. Despite their different action decoders, these models repeatedly perform visual encoding, language-backbone inference, and action generation during closed-loop control. Efficient deployment therefore requires reducing redundant computation throughout this pipeline while preserving action-relevant information.

### 2.2 Efficient VLA Inference

Efficient VLA inference reduces the cost of visual encoding, multimodal processing, and action generation while preserving task performance. Existing methods mainly focus on visual-token compression, cross-timestep computation reuse, and model-size reduction.

**Visual sequence compression.** Visual-token selection and merging reduce downstream inference cost by shortening the input sequence. TokenLearner (Ryoo et al., 2021), DynamicViT (Rao et al., 2021), and EViT (Liang et al., 2022) explore adaptive visual-token selection, while ToMe (Bolya et al., 2023) reduces sequence length by merging similar tokens. FastV (Chen et al., 2024) further applies visual-token pruning to large vision-language models. For VLA systems, SpecPrune-VLA (Wang et al., 2026) uses action-related signals for self-speculative visual pruning, while SAFE-Pruner (Ma et al., 2026) combines semantic and future-aware information to select informative tokens. ASSIST-VLA likewise targets visual redundancy by combining task semantics with local spatiotemporal information to select informative tokens during cognition refreshes.

**Computation reuse and action generation.** Temporal correlation in continuous control provides opportunities to reuse intermediate computation. VLA-Cache (Xu et al., 2025) reuses intermediate states associated with visual tokens, while EfficientVLA (Yang et al., 2025) combines visual-token selection, language-module compression, and feature reuse in diffusion action heads for multi-stage training-free acceleration. OpenVLA-OFT (Kim et al., 2025) improves action-generation efficiency through fine-tuning with action-chunk prediction and parallel decoding. ASSIST-VLA performs reuse at the level of the complete cognition representation: historical action-plan consistency determines whether an eligible control step can bypass visual encoding and language-backbone computation while retaining action-head execution.

**Model compression.** Quantization and pruning reduce inference cost through lower-precision representations or smaller model structures. GPTQ (Frantar et al., 2023), SmoothQuant (Xiao et al.,

2023), and AWQ (Lin et al., 2024) investigate low-bit quantization. SparseGPT (Frantar & Alistarh, 2023) and Wanda (Sun et al., 2024) explore unstructured sparsification, while LLM-Pruner (Ma et al., 2023) targets structural pruning in large language models. For VLA models, QVLA (Xu et al., 2026) uses action sensitivity to guide mixed-precision quantization, and GLUESTICK (Jabbour et al., 2025) studies pruning-induced performance degradation and training-free recovery. ASSIST-VLA instead assesses the influence of intermediate MLP channels on action outputs and reconstructs the corresponding projection matrices to obtain a compact model that operates jointly with online visual selection and cognition reuse.

### 2.3 J-Space and the Jacobian Lens

The J-space perspective analyzes internal representations through their downstream effects on model outputs, rather than relying only on their magnitudes or distances in the original representation space. For an intermediate representation $\mathbf{h}$ and output $\mathbf{y}$, this local effect can be written as

$$\Delta \mathbf{y} \approx \mathbf{J} \Delta \mathbf{h}, \qquad \mathbf{J} = \frac{\partial \mathbf{y}}{\partial \mathbf{h}}. \tag{1}$$

Here, $\Delta \mathbf{h}$ denotes a perturbation of the intermediate representation, $\Delta \mathbf{y}$ is the resulting output change, and $\mathbf{J}$ is the local Jacobian mapping from representation space to output space. The Jacobian lens introduced by Gurnee et al. (Gurnee et al., 2026) uses layer-wise average Jacobians to map intermediate representations into an output-relevant space, supporting the study of verbalizable information and global-workspace properties in language models. Inspired by this perspective, ASSIST-VLA extends the analysis to VLA control by composing Jacobian mappings from intermediate representations through cognition features to action outputs, thereby measuring MLP-channel influence on actions and guiding the construction of a compact language backbone.

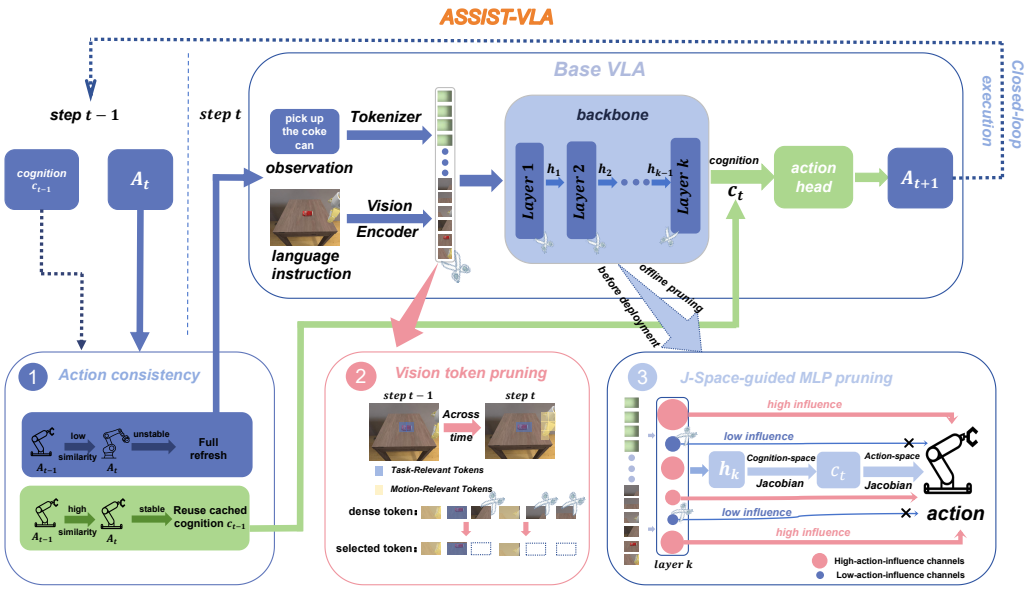



Figure 2: Overview of ASSIST-VLA, a unified training-free framework for efficient closed-loop VLA inference. It combines: (1) action-consistency-guided cognition reuse, where stable steps follow the green path to reuse cached cognition representations and bypass visual encoding and language-backbone inference; (2) semantic-spatiotemporal visual-token pruning, which retains task- and motion-relevant tokens on refresh steps; and (3) action-influence-aware structural MLP pruning, which constructs a compact language backbone before deployment.

# 3 METHOD

## 3.1 OVERALL ARCHITECTURE

Figure 2 illustrates the overall architecture of ASSIST-VLA. We follow CogACT's decoupled design, in which a vision encoder and a multimodal language backbone transform visual observations and language instructions into a cognition representation that conditions a diffusion action head. Repeated visual encoding, multimodal reasoning, and action generation at every control cycle create substantial online overhead. ASSIST-VLA reduces this redundancy at three levels: **(1)** before deployment, J-space-guided structural pruning uses cognition-space and action-space Jacobian mappings to evaluate the influence of MLP channels on final action outputs, removes low-action-influence channels, and constructs a compact language backbone; **(2)** during online inference, the cognition-reuse module determines whether to refresh the cognition representation based on historical action-plan consistency and a bounded refresh interval, and selects visual tokens using task semantics and local spatiotemporal information on refresh steps; **(3)** on reuse steps, the system directly uses the cached representation, bypassing visual encoding and language-backbone computation. The compact language backbone produces an updated cognition representation on refresh steps, and both paths execute the action head to generate the current action, whose execution produces the observation for the next control step. Together, these modules reduce the frequency of full multimodal inference, the visual sequence length, and the MLP computation required during refresh steps, thereby lowering the end-to-end cost of closed-loop VLA inference.

## 3.2 ACTION-AUTOCORRELATION COGNITION REUSE

Closed-loop robot control repeatedly invokes a VLA model at a high frequency, although the underlying control intent does not change at the same rate. For example, when the robot approaches a target or performs a continuous translation, consecutive action plans typically exhibit similar motion directions and magnitudes. ASSIST-VLA exploits this action autocorrelation to identify stable control phases and reuse the cognition representation produced by the most recent full inference.

At each full-refresh step, the model predicts an action plan $\mathbf{A}_t$ of length $K$. We assess the stability of the current control state by comparing the current action plan with recent historical plans, its temporally aligned predecessor, and the action to be executed next:

$$\rho_t = \mathcal{G}\Big(\mathrm{Sim}(\mathbf{A}_t, \mathcal{H}_t), \mathrm{Sim}(\mathbf{A}_t^{0:K_p}, \mathcal{H}_t^{0:K_p}), \mathrm{Sim}(\mathbf{A}_t^{0:K-1}, \mathbf{A}_{t-1}^{1:K}), \Delta\mathbf{a}_t, \Delta a_{t,\mathrm{grip}}\Big). \tag{2}$$

Here, $\mathcal{H}_t$ denotes a short history of action plans. The first three terms capture global-plan, near-term-prefix, and temporally aligned consistency, respectively. The remaining terms measure changes in the immediate action and gripper command.

The aggregation function $\mathcal{G}(\cdot)$ first converts these heterogeneous signals to a common stability scale. Cosine similarities are directly used as consistency scores, whereas action distances and gripper changes are transformed according to predefined variation ranges into bounded scores, so that smaller differences receive higher stability values. The system then combines the global-plan, local-prefix, temporal-alignment, and immediate-action signals using fixed weights to obtain the stability score for the current full-refresh step. To reduce the influence of fluctuations in an individual action prediction, this score is exponentially smoothed with the score from the previous full-refresh evaluation. Thus, the reuse decision is based on multiple consistency signals and their evolution across refresh steps rather than on a single average similarity. The concrete similarity definitions, weights, variation ranges, and thresholds are provided in Appendix A.2.

The system enters the cognition-reuse state only when the smoothed stability score and individual consistency conditions remain satisfied across consecutive refresh evaluations. When reuse is permitted, ASSIST-VLA uses the cached cognition representation to generate the next action plan, bypassing visual encoding and the full language-backbone forward pass. Otherwise, the system reselects visual tokens, runs the compact model, and updates the cached cognition representation. The reuse interval is bounded to prevent staleness, while an instruction change clears the cache and triggers a new full multimodal pass.

### 3.3 SEMANTIC-SPATIOTEMPORAL VISUAL TOKEN PRUNING

Visual observations often contain task-irrelevant background content, as well as information that is highly redundant across consecutive frames and within local spatial regions. Feeding all visual tokens into the multimodal LLM increases attention computation and language-model inference cost. In contrast, task targets, the robot end effector, and object-interaction regions usually occupy only a small portion of the image but are crucial for action decisions. ASSIST-VLA therefore selectively compresses visual tokens whenever the cognition representation needs to be refreshed.

The visual encoder produces $N$ visual tokens. For each token, we evaluate its task-semantic relevance, inter-frame variation, and local spatial distinctiveness, and combine these signals into a unified importance score:

$$S_{t,i} = \lambda_{\text{sem}} S_{t,i}^{\text{sem}} + \lambda_{\text{temp}} S_{t,i}^{\text{temp}} + \lambda_{\text{local}} S_{t,i}^{\text{local}}. \tag{3}$$

Here, $S_{t,i}^{\text{sem}}$ measures the semantic relevance between the $i$-th visual token and the language instruction, prioritizing target objects and task-critical regions. $S_{t,i}^{\text{temp}}$ measures the variation between the current token and the token at the corresponding position in the previous frame, preserving dynamic information associated with robot motion, object displacement, and interaction. $S_{t,i}^{\text{local}}$ measures the difference between a token and its spatial neighborhood, suppressing highly repetitive background regions. The weights $\lambda_{\text{sem}}$, $\lambda_{\text{temp}}$, and $\lambda_{\text{local}}$ correspond to the semantic, temporal, and local terms, respectively, and balance these three sources of information.

Given the importance scores, ASSIST-VLA selects the highest-scoring tokens under a token budget and retains their original spatial position encodings. Keeping positional information preserves the spatial layout needed to distinguish target objects, the robot end effector, and interaction regions.

This pruning procedure is performed only when the cognition representation requires a refresh. During control cycles that reuse a cached cognition representation, the system does not process a new visual sequence. By shortening the visual sequence, ASSIST-VLA lowers the input size and attention cost of the subsequent multimodal LLM while preserving information relevant to task semantics and closed-loop control.

### 3.4 ACTION-INFLUENCE-AWARE STRUCTURAL PRUNING

The MLP blocks in multimodal LLMs typically contain high-dimensional intermediate layers, while different intermediate channels contribute unequally to the final robot actions. Conventional pruning methods usually rely on weight magnitude or activation statistics, which do not directly reflect the influence of a channel on action outputs. Motivated by the J-space perspective, we argue that what matters for closed-loop control is not the magnitude of an intermediate representation in its native space, but the effect of that representation after it is transported, via an average Jacobian, into an output-relevant space. The Jacobian lens (Gurnee et al., 2026) uses a layer-wise average Jacobian to map intermediate activations into the subspace of representations that the model is poised to verbalize. In a VLA model, we rewrite this mapping from the language output space to the action output space, and thereby measure how an intermediate channel influences the action the model is poised to execute. ASSIST-VLA therefore evaluates MLP intermediate channels from an action-oriented perspective before deployment, using representative robot-control trajectories for calibration.

For the $j$-th intermediate channel of the $\ell$-th MLP layer, we jointly consider its activation strength on the calibration data and the influence of this channel after it is written into subsequent representations by the down-projection and then transported to the final action by action-related Jacobians. The channel importance is defined as

$$S_{\ell,j} = E_{\ell,j} \left\| \widehat{\mathbf{A}} \widehat{\mathbf{J}}_{\ell} \mathbf{w}_{\ell,j}^{\text{down}} \right\|_2^2. \tag{4}$$

Here, $E_{\ell,j}$ denotes the average activation energy of the channel over the calibration trajectories, and $\mathbf{w}_{\ell,j}^{\text{down}}$ denotes the corresponding column of the down-projection matrix. $\widehat{\mathbf{J}}_{\ell}$ is the average Jacobian from the layer representation to the cognition representation, characterizing how the channel enters the subsequent workspace. $\widehat{\mathbf{A}}$ is the average Jacobian from the cognition representation to the action output, characterizing how workspace variations change the final action. Their composition forms an action lens $\widehat{\mathbf{A}}\widehat{\mathbf{J}}_{\ell}$, which evaluates intermediate channels directly in an action- related J-space.

These quantities are estimated by perturbing intermediate representations and measuring the resulting action variations on calibration trajectories. The score therefore favors channels that are both frequently activated and strongly coupled to the final action along the action J-space, rather than channels that are merely large in the original activation space.

ASSIST-VLA globally ranks channels across candidate MLP layers and removes the lowest-scoring channels to meet a predefined sparsity target. A layer-wise pruning cap prevents excessive compression in any single layer, while retained intermediate widths are aligned with hardware-friendly block sizes. The gate, up, and down projection matrices are then physically reconstructed from the retained channels, directly reducing the intermediate width. Unlike unstructured pruning, which only sets selected weights to zero, physical dimension reduction lowers parameter storage, memory usage, and dense matrix computation. The compact model is generated once before deployment and combined with visual-token pruning and cognition-representation reuse during online control.

# 4 EXPERIMENTS

## 4.1 EXPERIMENTAL SETUP

We evaluate ASSIST-VLA on CogACT-Base Li et al. (2024a) without additional training. All compared methods share the same pretrained weights and the same remaining inference configuration; they differ only in the acceleration modules.

**Baselines.** We compare against the original CogACT-Base model and three representative training-free VLA acceleration methods: FastV Chen et al. (2024), which prunes visual tokens according to shallow-layer attention; VLA-Cache Xu et al. (2025), which reuses key–value states of static visual tokens across consecutive frames; and EfficientVLA Yang et al. (2025), which combines visual-token pruning, language-layer compression, and action-head feature caching. To our knowledge, none of these methods provides a public CogACT implementation, so we reimplement them on CogACT-Base following the descriptions in the original papers.

**Benchmark.** We follow the official visual-matching protocol of SimplerEnv Li et al. (2024b) and evaluate four tasks: *Pick Coke Can*, *Open/Close Drawer*, *Place Apple in Closed Drawer*, and *Put Carrot on Plate*. These tasks cover grasping, articulated manipulation, long-horizon placement, and cross-embodiment rearrangement, yielding diverse closed-loop evaluation scenarios.

Action-influence scores for structural pruning are calibrated using trajectories of coke-can grasping, drawer opening and closing, apple placement into drawers, and *Move Near*. This calibration set covers grasping, articulated interaction, placement, and continuous translation, and is used to estimate the coupling between intermediate channels and the final action. *Put Carrot on Plate* is not used for calibration. Details of the calibration data and procedures are provided in Appendix A.4.

**Metrics.** We report task success rate and end-to-end per-step latency. Success rate is computed under the official episode protocol. Latency is measured on a single NVIDIA RTX A6000 GPU, from receiving the current observation to emitting an executable action, and we report the steady-state mean after warmup.

Table 1: Main results on SimplerEnv. FLOPs denotes remaining computation relative to CogACT. Bold and underline indicate the best and second-best results, respectively.

| Method | Success Rate (%)↑ | | | | Avg.↑ | Latency (ms)↓ | FLOPs (%)↓ | Speedup↑ | Params (B)↓ |
|---|---|---|---|---|---|---|---|---|---|
| | PickCan | Drawer | DrawerApple | Carrot | | | | | |
| CogACT | **93.7** | **73.6** | 50.9 | **50.0** | <u>67.1</u> | 229.1 | 100.0 | 1.00× | 7.63 |
| FastV | <u>93.3</u> | 69.0 | <u>60.0</u> | 36.7 | 64.8 | 198.6 | 42.0 | 1.15× | 7.63 |
| VLA-Cache | 92.0 | 70.5 | 51.6 | **50.0** | 66.0 | 177.6 | 80.1 | 1.29× | 7.63 |
| EfficientVLA | <u>93.3</u> | 68.2 | 53.8 | **50.0** | 66.3 | <u>168.5</u> | <u>28.9</u> | <u>1.36×</u> | **4.86** |
| **ASSIST-VLA** | 88.0 | <u>73.3</u> | **61.7** | <u>48.3</u> | **67.8** | **158.5** | **27.7** | **1.45×** | <u>6.55</u> |

### 4.2 MAIN RESULTS

Table 1 compares ASSIST-VLA with unaccelerated CogACT-Base Li et al. (2024a) and three training-free acceleration methods reimplemented from the original papers, under the official SimplerEnv visual-matching protocol. Besides task success rate, we report end-to-end per-step latency, parameter count, remaining FLOPs relative to the baseline, and wall-clock speedup.

Overall, ASSIST-VLA reduces per-step latency from $229.1$ ms to $158.5$ ms ($1.45\times$) while achieving a $67.8\%$ average success rate, compared with $67.1\%$ for the baseline. FastV Chen et al. (2024) and VLA-Cache Xu et al. (2025) retain the original parameter count and achieve only $1.15\times$ and $1.29\times$ speedups, with average success rates of $64.8\%$ and $66.0\%$, respectively. EfficientVLA Yang et al. (2025) compresses the model to $4.86$B parameters, achieving a $1.36\times$ speedup and a $66.3\%$ average success rate. In comparison, ASSIST-VLA improves the speedup to $1.45\times$ and raises the average success rate by $1.5$ percentage points. Although it retains more parameters, ASSIST-VLA reduces more FLOPs ($72.3\%$ vs. $71.1\%$) and achieves faster end-to-end inference. These results show that shortening the visual sequence and skipping full multimodal forwards during stable control can further reduce latency beyond parameter compression alone. The gap between FLOPs reduction and wall-clock speedup reflects the fixed costs of visual encoding, feature projection, and action generation, which do not decrease proportionally with language-model computation.

Per-task results further illustrate this efficiency–performance trade-off. ASSIST-VLA stays close to the baseline on Open/Close Drawer ($73.3\%$ vs. $73.6\%$) and improves Place Apple in Closed Drawer from $50.9\%$ to $61.7\%$. On Put Carrot on Plate, which is held out from structural-pruning calibration, it reaches $48.3\%$ versus the baseline $50.0\%$, and remains substantially above FastV ($36.7\%$). The main performance loss appears on Pick Coke Can, where success drops from $93.7\%$ to $88.0\%$. This decrease may be related to the task's reliance on persistent localization of a small target, which can make it sensitive to visual compression and cognition reuse. We isolate the contribution of each module in the following ablation.

### 4.3 ABLATION STUDY

To analyze the role of each acceleration module, we conduct a leave-one-out ablation of cognition reuse (C), visual-token pruning (V), and MLP structural pruning (M). All configurations use the same pretrained CogACT weights and follow the official SimplerEnv evaluation protocol. In Table 2, a checkmark indicates that the corresponding module is enabled, and the cache hit rate is reported only for configurations with C. We use the complete ASSIST-VLA configuration as the reference and examine how removing each module affects inference latency, speedup, and task performance.

**Effect of cognition reuse.** Cognition reuse provides the largest marginal acceleration benefit in the complete configuration. Removing C while retaining V and M increases the per-step latency from $158.5$ ms to $188.7$ ms and reduces the speedup from $1.45\times$ to $1.21\times$. This result shows that cross-cycle computation reuse can further reduce inference overhead even after compressing the visual sequence and model structure. The complete configuration achieves a cache hit rate of $22.4\%$, indicating that cached cognition representations can be reused during a subset of stable control cycles, thereby bypassing visual encoding and the complete multimodal LLM forward pass. Meanwhile, removing C increases the average success rate from $67.8\%$ to $72.7\%$. Thus, the configuration without cognition reuse favors task performance, whereas the complete configuration trades some task performance for greater inference efficiency.

**Effect of visual-token pruning.** Visual-token pruning reduces multimodal inference overhead by shortening the input sequence. Removing V increases per-step latency from $158.5$ ms to $171.1$ ms and reduces the speedup from $1.45\times$ to $1.34\times$. Meanwhile, the average success rate decreases from $67.8\%$ to $61.3\%$, with DrawerApple dropping from $61.7\%$ to $32.4\%$. These results indicate that, within the current module combination, visual pruning not only reduces inference overhead but also helps maintain task performance. This is consistent with its design objective of prioritizing task-relevant semantic information and control-relevant dynamic information while reducing interference from redundant background content. The role of visual pruning therefore extends beyond token-count reduction to selectively preserving visual inputs that better support task-specific control.

Table 2: Leave-one-out ablation of cognition reuse (C), visual-token pruning (V), and action-influence-aware structural pruning (M). Speedup is relative to CogACT-Base. Bold and underline indicate the best and second-best results, respectively.

| Method | C | V | M | Success Rate (%)↑ | | | | Avg.↑ | Speedup↑ | Hit rate (%) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | PickCan | Drawer | DrawerApple | Carrot | | | |
| Baseline | | | | **93.7** | 73.6 | 50.9 | <u>50.0</u> | 67.1 | 1.00× | — |
| w/o C | | ✓ | ✓ | 89.0 | <u>76.4</u> | **71.3** | **54.2** | **72.7** | 1.21× | — |
| w/o V | ✓ | | ✓ | <u>93.3</u> | 69.4 | 32.4 | <u>50.0</u> | 61.3 | <u>1.34×</u> | <u>21.5</u> |
| w/o M | ✓ | ✓ | | 87.7 | **80.1** | <u>70.4</u> | 37.5 | <u>68.9</u> | 1.28× | 20.5 |
| ASSIST-VLA | ✓ | ✓ | ✓ | 88.0 | 73.3 | 61.7 | 48.3 | 67.8 | **1.45×** | **22.4** |

**Effect of MLP structural pruning.** MLP structural pruning further reduces model computation during cognition refreshes. Removing M increases per-step latency from $158.5$ ms to $178.4$ ms and reduces the speedup from $1.45\times$ to $1.28\times$. In other words, with C and V enabled, adding M reduces per-step latency by $19.9$ ms, while the average success rate decreases by only $1.1$ percentage points, from $68.9\%$ to $67.8\%$. This result demonstrates that physically compacting the intermediate MLP dimensions provides additional inference efficiency beyond visual compression and cognition reuse, while keeping average task performance close to that of the corresponding unpruned configuration.

Overall, the three modules play complementary roles in the current configuration: C provides the largest marginal acceleration benefit, V reduces input computation while helping maintain task performance, and M further reduces model computation at each cognition refresh. The complete configuration achieves the highest speedup among the evaluated ablation settings while maintaining an average success rate close to the unaccelerated baseline, demonstrating the value of jointly optimizing the control-cycle, input-sequence, and model-structure levels.

## 5 Conclusion

Closed-loop vision–language–action models repeatedly perform visual encoding and multimodal inference at high frequency, despite redundancy in control intent, task-relevant visual information, and the contributions of internal model components to action outputs. ASSIST-VLA reduces inference overhead at three levels without additional training: reusing cognition representations during stable control, compressing visual tokens using semantic and spatiotemporal information, and structurally pruning MLP channels according to their influence on final actions. Cognition reuse limits perception lag through periodic and event-triggered refreshes; visual pruning prioritizes task-relevant semantic information and control-relevant dynamic information; and structural pruning physically compacts intermediate model dimensions before deployment. The main experiments and ablation results demonstrate the complementary roles of these three modules in the evaluated configuration. On CogACT-Base and SimplerEnv, ASSIST-VLA reduces end-to-end per-step latency from $229.1$ ms to $158.5$ ms, achieving a $1.45\times$ speedup while attaining a $67.8\%$ average success rate, close to the unaccelerated baseline of $67.1\%$. These results highlight the potential of jointly optimizing the control-cycle, input-sequence, and model-structure levels for efficient VLA inference.

### AI use statement

In this work, we used generative AI tools to search for related work, draft sections of the paper, and edit the manuscript for readability. We have not used generative AI tools to generate synthetic datasets, develop theoretical models, formulate or prove mathematical claims, interpret experimental results, or clean and reformat datasets; the remaining required-disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to assist with translation, create and modify scientific figures, and write plotting code. We have reviewed all AI-assisted text, figures, code, and claims. All reported numbers are taken from our own evaluation logs, and the authors take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

### Ethics statement

This work studies training-free inference acceleration of vision-language-action models in simulation (SimplerEnv) and does not involve human subjects or the release of new datasets. The method does not collect private data and is evaluated only on public robotic manipulation benchmarks. We do not claim readiness for unsupervised real-world deployment; any real-robot use would still require standard safety controls. We foresee no additional ethical concerns beyond those already associated with robot learning research.

### Reproducibility Statement

Appendix A provides detailed implementation specifications for ASSIST-VLA, including cognition-cache thresholds and update rules, visual-token scoring and selection, and structural-pruning calibration and configuration. We report the calibration-data composition, sampling procedures and random seeds, action-influence scoring formulas, pruning budgets, and retained MLP dimensions for each layer. The appendix also describes action generation and the execution order of the combined method. We will publicly release the source code for ASSIST-VLA, together with the configuration files and scripts needed to reproduce the method.

## References

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. In *International Conference on Learning Representations*, 2023.

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.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision*, 2024.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *Proceedings of the International Conference on Machine Learning*, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations*, 2023.

Wes Gurnee, Nicholas Sofroniew, Adam Pearce, Mateusz Piotrowski, Isaac Kauvar, Runjin Chen, Anna Soligo, Paul Bogdan, Euan Ong, Rowan Wang, Ben Thompson, David Abrahams, Subhash Kantamneni, Emmanuel Ameisen, Joshua Batson, and Jack Lindsey. Verbalizable representations form a global workspace in language models. *Transformer Circuits Thread*, 2026. URL `https://transformer-circuits.pub/2026/workspace/index.html`.

Jason Jabbour, Dong-Ki Kim, Max Smith, Jay Patrikar, Radhika Ghosal, Youhui Wang, Ali Agha, Vijay Janapa Reddi, and Shayegan Omidshafiei. Don't run with scissors: Pruning breaks vla models but they can be recovered. *arXiv preprint arXiv:2510.08464*, 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. In *Robotics: Science and Systems*, 2025.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024a.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishika Lunawat, Isabel Sieh, Soroush Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. In *Conference on Robot Learning*, 2024b.

Youwei Liang, Chongjian Ge, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. Not all patches are what you need: Expediting vision transformers via token reorganizations. In *International Conference on Learning Representations*, 2022.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. In *Proceedings of Machine Learning and Systems*, 2024.

Shilin Ma, Chubin Zhang, Changyuan Wang, Yuji Wang, Yue Wu, Zixuan Wang, Jingqi Tian, Zheng Zhu, and Yansong Tang. Safe-pruner: Semantic attention-guided future-aware token pruning for efficient vision-language-action manipulation. *European Conference on Computer Vision*, 2026. To appear.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, 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.

Physical Intelligence, 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.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *Advances in Neural Information Processing Systems*, 2021.

Michael S. Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. Tokenlearner: What can 8 learned tokens do for images and videos? In *Advances in Neural Information Processing Systems*, 2021.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*, 2024.

Hanzhen Wang, Jiaming Xu, Yushun Xiang, Jiayi Pan, Yongkang Zhou, Yong-Lu Li, and Guohao Dai. Specprune-vla: Accelerating vision-language-action models via action-aware self-speculative pruning. *Proceedings of the International Conference on Machine Learning*, 2026. To appear.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the International Conference on Machine Learning*, 2023.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. Vla-cache: Efficient vision-language-action manipulation via adaptive token caching. *Advances in Neural Information Processing Systems*, 2025.

Yuhao Xu, Yantai Yang, Zhenyang Fan, Yufan Liu, Yuming Li, Bing Li, and Zhipeng Zhang. Qvla: Not all channels are equal in vision-language-action model's quantization. In *International Conference on Learning Representations*, 2026.

Yantai Yang, Yuhao Wang, Zichen Wen, Zhongwei Luo, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. Efficientvla: Training-free acceleration and compression for vision-language-action models. *arXiv preprint arXiv:2506.10100*, 2025.

# A METHOD IMPLEMENTATION AND REPRODUCIBILITY

This appendix specifies the implementation of ASSIST-VLA, including action generation, cognition caching, visual-token selection, and offline structural pruning. All settings below refer to the configuration used by the combined method. The same settings are retained for an enabled component when another component is disabled. No additional fine-tuning is performed after structural pruning.

## A.1 BASE MODEL AND ACTION GENERATION

ASSIST-VLA uses CogACT-Base with a Llama-2-7B language backbone and a DiT-B action head. The language backbone has 32 Transformer layers, hidden dimension $d = 4096$, and an original MLP intermediate dimension of $m = 11008$ in every layer. Structural pruning changes the MLP intermediate dimensions while preserving the number of Transformer layers and the residual-stream dimension.

The input image is resized to $224 \times 224$. Full visual encoding and projection produce $N = 256$ visual tokens on a $16 \times 16$ grid, each projected to the 4096-dimensional language embedding space. A full multimodal forward produces the cognition feature used to condition the action head.

Table 3: Base-model and online action-generation settings.

| Setting | Value |
|---|---|
| Base model | CogACT-Base |
| Language backbone | Llama-2-7B |
| Transformer layers | 32 |
| Hidden dimension | 4096 |
| Original MLP intermediate dimension | 11008 |
| Image resolution | $224 \times 224$ |
| Original visual tokens | 256 |
| Action head | DiT-B |
| Predicted action horizon | 16 |
| Action dimension | 7 |
| Action sampler | DDIM |
| DDIM steps | 10 |
| DDIM $\eta$ | 0 |
| Classifier-free guidance scale | 1.5 |
| Action scaling factor | 1.0 |
| Online action-noise seed | 1 |

**Language input.** The task instruction is lowercased and inserted into the prompt "What action should the robot take to {instruction}?" using the CogACT prompt builder. The implementation appends token IDs $[29871, 2]$ to the tokenized prompt. The semantic visual-selection branch separately tokenizes the lowercased task instruction without special tokens, as detailed in Section A.3.

**Action representation and decoding.** Each prediction is a plan $P_t \in \mathbb{R}^{H\times 7}$ with $H = 16$. The seven components represent translation in three dimensions, rotation in three dimensions, and the gripper command. The action head uses 10 DDIM steps with $\eta = 0$ and a classifier-free guidance scale of 1.5. The resulting normalized actions are clipped to $[-1, 1]$, and the gripper component is binarized at a threshold of 0.5. For dimensions selected by the normalization mask in the checkpoint, the action is decoded as

$$a_k = \frac{\widetilde{a}_k + 1}{2} \left(q_{99,k} - q_{01,k}\right) + q_{01,k}, \tag{5}$$

where $q_{01,k}$ and $q_{99,k}$ are the stored first and ninety-ninth percentile statistics. Dimensions outside this mask retain their normalized values. The Google Robot configuration uses the statistics indexed by `fractal20220817_data`; the WidowX configuration uses `bridge_orig`.

The cache compares decoded action plans before temporal ensembling and before conversion to the robot control interface. Thus, its distance thresholds apply to this decoded seven-dimensional repre-

sentation. A 16-step plan is predicted at each policy invocation, but only one temporally ensembled action is emitted for the current control step.

**Action noise.** Online inference uses a CPU random-number generator initialized with seed 1. Gaussian noise is generated in FP32 and then transferred to the action head's device and data type. A full refresh stores the sampled noise together with the cognition feature. A cache-reuse step uses this stored noise instead of drawing a new sample. Resetting an episode or the instruction-dependent inference state reinitializes the generator and clears the stored noise.

**Temporal action ensembling.** Historical plans are aligned to the current control step. Let $b_1, \ldots, b_n$ denote the aligned action predictions, with $b_n$ from the newest plan. The ensemble is

$$c_j = \frac{b_j^\top b_n}{\|b_j\|_2 \|b_n\|_2 + 10^{-7}}, \tag{6}$$

$$w_j = \frac{\exp(\alpha_{\text{ens}} c_j)}{\sum_{k=1}^{n} \exp(\alpha_{\text{ens}} c_k)}, \qquad \alpha_{\text{ens}} = 0.1, \tag{7}$$

$$a_t^{\text{ens}} = \sum_{j=1}^{n} w_j b_j. \tag{8}$$

The maximum number of plans is 2 for Google Robot and 7 for WidowX. Both refresh and reuse steps use this operation. The robot interface uses a sticky-gripper repeat count of 10 for Google Robot and 1 for WidowX. Ensembling history is cleared on reset.

### A.2 Cognition Cache: Scores, State Transitions, and Updates

The cache stores the latest full-forward cognition feature, its action-sampling noise, the previous action plan, and a bounded history of action plans. Stability is evaluated from newly generated full-forward plans. The history holds at most six plans; global and prefix consistency use its most recent three entries, or all available entries if fewer than three exist.

**Plan-consistency measurements.** Define the flattened cosine similarity

$$\text{cs}(X, Y) = \frac{\text{vec}(X)^\top \text{vec}(Y)}{\|\text{vec}(X)\|_2 \|\text{vec}(Y)\|_2 + 10^{-8}}. \tag{9}$$

The implementation returns zero if either norm is below $10^{-8}$. Let $P_t$ be a new full-forward plan, $\mathcal{H}_t$ the comparison history, and $P^-$ its newest member. Using zero-based action indices and half-open slices, the six measurements are

$$g_t = \frac{1}{|\mathcal{H}_t|} \sum_{P \in \mathcal{H}_t} \text{cs}(P_t, P), \tag{10}$$

$$p_t = \frac{1}{|\mathcal{H}_t|} \sum_{P \in \mathcal{H}_t} \text{cs}(P_t[0:4], P[0:4]), \tag{11}$$

$$s_t = \text{cs}(P_t[0:15], P^-[1:16]), \tag{12}$$

$$f_t = \text{cs}(P_t[0], P^-[1]), \tag{13}$$

$$d_t = \|P_t[0] - P^-[1]\|_2, \tag{14}$$

$$\delta_t = |P_t[0,6] - P^-[1,6]|. \tag{15}$$

The first-action cosine and distance include all seven action dimensions. Global and prefix comparisons use matching sequence indices, whereas $s_t$, $f_t$, $d_t$, and $\delta_t$ account for the one-step shift of the previous plan.

**Smoothed stability score.** The distance measurements are converted to bounded scores:

$$q_d(t) = \text{clip}\left(1 - \frac{d_t}{0.06}, 0, 1\right), \qquad q_g(t) = \text{clip}\left(1 - \frac{\delta_t}{0.075}, 0, 1\right). \tag{16}$$

Table 4: Cognition-cache thresholds and state parameters.

| Setting | Value |
|---|---|
| Global consistency | $g_t \geq 0.92$ |
| Prefix consistency | $p_t \geq 0.92$ |
| Shift consistency | $s_t \geq 0.85$ |
| First-action cosine | $f_t \geq 0.85$ |
| First-action distance | $d_t \leq 0.06$ |
| Gripper difference | $\delta_t \leq 0.075$ |
| History capacity / comparison window | 6 / 3 |
| Prefix length | 4 |
| Score smoothing coefficient | 0.50 |
| Smoothed-score threshold | 0.90 |
| Stable-entry count | 2 |
| Unstable-exit count | 1 |
| Base / maximum refresh interval | 2 / 2 |
| Previous-plan mixing weight | 0.75 |

The combined score is

$$r_t = 0.25g_t + 0.25p_t + 0.20s_t + 0.15f_t + 0.10q_d(t) + 0.05q_g(t). \tag{17}$$

At a full-forward step with a nonempty history, update

$$\overline{r}_t = 0.5r_t + 0.5\overline{r}_{\mathrm{prev}}, \tag{18}$$

where $\mathrm{prev}$ denotes the previous stability-update event. The first evaluable score initializes $\overline{r}_t = r_t$. A qualifying stability signal requires all six inequalities in Table 4 and $\overline{r}_t \geq 0.90$ simultaneously.

**State transitions.** An empty history starts in the cold state. The first full-forward plan establishes history without incrementing the stable streak. Each subsequent full-forward evaluation updates the score and streaks before appending its plan. Two consecutive qualifying evaluations enter the stable state. One nonqualifying evaluation resets the stable streak and exits stability. A reused plan is appended to the history, but does not update the score, its exponential average, or either streak. Consequently, history used at a later refresh can contain both fresh and reused plans.

**Reuse decision and maximum interval.** A reuse step requires caching to be enabled, a stored cognition feature, an available previous plan, and a stable cache state. Let $n_t$ be the number of reuse steps since the last full refresh. With refresh interval $I = 2$, reuse additionally requires $n_t < I - 1$. A refresh sets $n_t = 0$, and reuse increments it by one. Thus, at most one consecutive reuse step is allowed.

The adaptive-interval interface is enabled with score levels $(0.90, 0.94)$ and streak levels $(2, 4)$, but both the base and maximum intervals are 2. These levels therefore do not extend the effective interval beyond 2. Reuse is decided from the previously stored state before current-step action generation; a reuse step does not run a new visual check. An instruction change or episode reset clears the cognition cache, plan history, stability state, visual history, and action-ensemble history.

**Action generation on a reuse step.** The action head runs with the cached cognition feature and cached diffusion noise to obtain $\widehat{P}_t$. Let $P_{t-1}$ be the previously stored, pre-ensemble plan. The reused plan is

$$P_t[k] = \begin{cases} 0.75P_{t-1}[k+1] + 0.25\widehat{P}_t[k], & 0 \leq k < 15, \\ \widehat{P}_t[15], & k = 15. \end{cases} \tag{19}$$

After mixing, all gripper components are binarized at 0.5. The resulting plan is stored and passed to temporal ensembling. The action head and action postprocessing therefore remain active on reuse steps.

### A.3 Visual Scores and Token Selection

Selection operates on the full projected visual sequence before the language backbone. The vision encoder and projector process the complete image on every full-refresh step.

**Token budget and history.** For $N$ input visual tokens, the retained budget is

$$K = \min\left(N, \max\left(64, \lfloor 0.25N \rfloor\right)\right). \tag{20}$$

Thus, $N = 256$ yields $K = 64$. The first processed image after reset retains all 256 tokens and establishes visual history. Subsequent refreshes retain 64. History stores the full projected sequence before selection and uses only the most recent processed observation. Cache-reuse steps neither execute visual encoding nor update this history, so the reference observation need not be the immediately preceding control-step observation.

**Score normalization.** Scoring converts projected features to FP32. For a vector of scores $x$, define

$$\mathcal{N}(x)_i = \frac{x_i - \min_j x_j}{\max_j x_j - \min_j x_j + 10^{-8}}. \tag{21}$$

If its range is below $10^{-8}$, the normalized vector is zero. This normalization is applied separately at each stage specified below.

**Instruction relevance.** Let $v_i \in \mathbb{R}^{4096}$ denote the current projected visual token and $e_j$ a token embedding of the lowercased task instruction. Instruction tokenization uses `add_special_tokens=False`. If this produces no tokens, the implementation falls back to the prompt embeddings. The semantic score is

$$S = \mathcal{N}(s^{\text{sem}}), \qquad s_i^{\text{sem}} = \max_j \cos(v_i, e_j). \tag{22}$$

**Local temporal correlation.** Let $v_j^-$ denote a token from the most recent visual history. Compute $C_{ij} = \cos(v_i, v_j^-)$. On the $16 \times 16$ grid, $\mathcal{N}_2(i)$ contains positions within a square radius of 2, including the center and clipped at image boundaries. Define

$$b_i = \max_{j \in \mathcal{N}_2(i)} C_{ij}, \qquad \mu_i = \frac{1}{|\mathcal{N}_2(i)|} \sum_{j \in \mathcal{N}_2(i)} C_{ij}, \qquad a_i = C_{ii}, \tag{23}$$

$$m_i = \max(b_i - a_i, 0), \qquad d_i^{\text{vis}} = \max(b_i - \mu_i, 0). \tag{24}$$

The local contrast of the displacement response is

$$\overline{m}_i = \frac{1}{|\mathcal{N}_2(i)|} \sum_{j \in \mathcal{N}_2(i)} m_j, \qquad u_i = \max(m_i - \overline{m}_i, 0). \tag{25}$$

Normalize and combine these responses as

$$M = \mathcal{N}\big(\mathcal{N}(m) + \mathcal{N}(u)\big), \tag{26}$$

$$D = \mathcal{N}(d^{\text{vis}}), \tag{27}$$

$$T = \mathcal{N}(0.7M + 0.3D). \tag{28}$$

The within-frame neighborhood operation averages the displacement-response map. This branch does not require a separate same-frame visual-embedding similarity matrix. The final selection score is

$$Q_i = 0.5S_i + 0.5T_i. \tag{29}$$

**Candidates and spatial seeds.** Select the 64 highest-scoring semantic positions and the 16 highest-scoring temporal positions. Initial semantic seeds are the top 16 positions under $S$. Motion seeds are selected greedily in descending order of $U = \mathcal{N}(u)$: after selecting a position, suppress its radius-1 neighborhood, stop after 16 seeds, and reject scores below $10^{-6}$. Expand both types of seeds by a radius-1 neighborhood, corresponding to a clipped $3 \times 3$ region. Let the expanded semantic and motion sets be $\mathcal{E}_S$ and $\mathcal{E}_T$, respectively. The candidate pool is

$$\mathcal{C} = \text{Top}_{64}(S) \cup \text{Top}_{16}(T) \cup \mathcal{E}_T. \tag{30}$$

Table 5: Visual-selection configuration for a 256-token input. Candidate and seed ratios are relative to the retained budget $K = 64$.

| Setting | Value |
|---|---|
| Retention ratio / minimum token count | 0.25 / 64 |
| Local-correlation radius | 2 |
| Semantic / temporal score weights | 0.5 / 0.5 |
| Displacement / distinctiveness weights | 0.7 / 0.3 |
| Visual-history length used for scoring | 1 |
| Semantic candidate ratio / count | 1.0 / 64 |
| Motion candidate ratio / count | 0.25 / 16 |
| Semantic seed ratio / initial count | 0.25 / 16 |
| Motion seed ratio / maximum initial count | 0.25 / 16 |
| Seed expansion / peak suppression radius | 1 / 1 |
| Minimum motion-peak score | $10^{-6}$ |
| Protected-seed ratio / budget | 0.35 / 22 |
| Motion-contrast bonus for seed filling | 0.35 |

**Protected-seed allocation.** The protected budget is $\lfloor 0.35K \rfloor = 22$. The equal requested seed counts allocate 11 slots to each branch. Within $\mathcal{E}_S$, keep up to 11 positions ranked by $S$; within $\mathcal{E}_T$, keep up to 11 ranked by $U$. Merge and deduplicate these sets. If the result has fewer than 22 entries, fill from unused positions in $\mathcal{E}_S \cup \mathcal{E}_T$ in descending order of $Q_i + 0.35U_i$, up to the protected budget or the available pool size.

**Final selection and positions.** Insert protected positions first, then fill from $\mathcal{C}$ in descending order of $Q$, skipping duplicates, until 64 positions have been selected. A fallback can fill from a $4 \times 4$ partition of the patch grid and then from global scores if the candidate pool is insufficient; the current 64-position semantic pool already provides enough candidates under the standard input configuration. Finally, sort retained indices in their original spatial order.

Position IDs preserve the original multimodal layout. BOS has position 0, a retained visual token with zero-based original index $i$ has position $i + 1$, and the text following the original visual block starts at position 257. Visual positions are not compacted to $1, \ldots, 64$, and text positions are not shifted forward after selection.

### A.4 Structural-Pruning Calibration

Offline calibration has two distinct data uses: a 7000-context pool for randomized directional-response measurements, and a fixed 500-context subset for MLP activation statistics. These quantities are not interchangeable.

**Context collection.** The pool contains 14 task buckets with 500 contexts each. Contexts are sampled uniformly along collected trajectories, with a target of ten sampled positions per trajectory and a collection run limit of 160 steps. Before sampling, the usable trajectory ends at the first recorded completion, truncation, predicted termination, or environment success event after the initial step. A run of four actions with norm below 0.01 further limits the usable segment to the position before that stationary run. Sampling uses equally spaced positions over the usable segment, rounds indices, and removes duplicates. Each record stores the task bucket, trajectory identifier, step, instruction, trajectory length, and sampled-step list.

**Instruction coverage.** A task bucket can contain different subtask instructions along a trajectory. The pool contains 33 distinct instructions, including 28, 53, and 66 contexts with the instructions to place an apple into the top, middle, and bottom drawer, respectively. The 500-context activation subset contains 3, 3, and 5 such contexts. Thus, apple-placement instructions participate in calibration. No Put Carrot on Plate contexts are used in this calibration pool.

**Fixed activation subset.** Sort context metadata paths lexicographically and retain records with an associated image. With `numpy.random.default_rng(0)`, draw 500 indices uniformly

Table 6: Offline calibration buckets and the fixed activation subset. Counts refer to calibration contexts, not evaluation episodes.

| Task bucket | Context pool | Activation subset |
|---|---|---|
| `pick_coke` | 500 | 33 |
| `pick_coke_vertical` | 500 | 42 |
| `pick_coke_standing` | 500 | 38 |
| `move_near` | 500 | 29 |
| `move_near_v1` | 500 | 30 |
| `open_top_drawer` | 500 | 39 |
| `open_middle_drawer` | 500 | 33 |
| `open_bottom_drawer` | 500 | 35 |
| `close_top_drawer` | 500 | 21 |
| `close_middle_drawer` | 500 | 38 |
| `close_bottom_drawer` | 500 | 40 |
| `drawer` | 500 | 36 |
| `place_middle_drawer` | 500 | 44 |
| `place_bottom_drawer` | 500 | 42 |
| Total | 7000 | 500 |

without replacement from the 7000 records, then sort the selected indices before processing. This procedure reproduces the saved activation-subset identifiers.

**Task–progress balancing for directional measurements.** Group contexts by task bucket and one of eight progress bins. If a context is the $r$-th entry, indexed from zero, in a trajectory's list of $n > 1$ sampled steps, use

$$p = \frac{r}{n-1}, \qquad b = \min(7, \lfloor 8p \rfloor). \tag{31}$$

When the sampled-step list is unavailable, use the recorded step divided by the trajectory length minus one, with a denominator of at least one and progress clipped to $[0, 1]$. Repeatedly shuffle the nonempty task–bin groups and sample one context from each group until the required number of trials is reached. This balances trials over nonempty groups rather than drawing in proportion to their sizes.

For an exact implementation of the random streams, initialize the group-order generator with `default_rng([seed,104729])`. For each trial, initialize its within-group generator with the seed, trial index, and the CRC32 value of the string formed as `task:bin`. Draw one index uniformly from that group's contexts.

### A.5 Action-Influence Scores

**Layer-to-cognition and cognition-to-action maps.** Let $h_\ell \in \mathbb{R}^{4096}$ denote the cognition-token residual representation at the output of Transformer layer $\ell$, and let $c \in \mathbb{R}^{4096}$ be the final cognition feature. Perturb only this token while keeping the other token representations at that boundary fixed. The remaining language computation defines $c = f_\ell(h_\ell)$, with

$$J_\ell = \frac{\partial c}{\partial h_\ell} \in \mathbb{R}^{4096 \times 4096}. \tag{32}$$

The action head defines $y = g(c) \in \mathbb{R}^{112}$ by flattening its $16 \times 7$ output before clipping, action unnormalization, or robot-interface postprocessing. Its Jacobian is

$$A = \frac{\partial y}{\partial c} \in \mathbb{R}^{112 \times 4096}. \tag{33}$$

**Random-direction finite differences.** For each sampled context and trial, generate a Gaussian direction and normalize it:

$$z_i \sim \mathcal{N}(0, I_d), \qquad u_i = \frac{z_i}{\max(\|z_i\|_2, 10^{-12})}, \qquad d = 4096. \tag{34}$$

Table 7: Directional-response calibration settings. The trial count for $\widehat{J}_\ell$ applies to each of the 32 layers.

| Setting | $\widehat{J}_\ell$ | $\widehat{A}$ |
|---|---|---|
| Input dimension | 4096 | 4096 |
| Output dimension | 4096 | 112 |
| Context pool | 7000 | 7000 |
| Progress bins | 8 | 8 |
| Finite-difference scheme | Forward | Forward |
| Perturbation magnitude | 0.1 | 0.1 |
| Directional trials | 131072 per layer | 32768 |
| Direction/sampling seed | 2 | 4 |
| Saved matrix data type | FP16 | FP32 |

Use a forward difference with $\epsilon = 0.1$:

$$\Delta_i^{(\ell)} = \frac{f_\ell(h_{\ell,i} + \epsilon u_i) - f_\ell(h_{\ell,i})}{\epsilon}, \tag{35}$$

$$\widehat{J}_\ell = \frac{d}{N_J} \sum_{i=1}^{N_J} \Delta_i^{(\ell)} u_i^\top, \qquad N_J = 131072, \tag{36}$$

$$\Delta_i^{(A)} = \frac{g(c_i + \epsilon u_i) - g(c_i)}{\epsilon}, \tag{37}$$

$$\widehat{A} = \frac{d}{N_A} \sum_{i=1}^{N_A} \Delta_i^{(A)} u_i^\top, \qquad N_A = 32768. \tag{38}$$

The factor $d$ corresponds to isotropic unit directions, for which $\mathbb{E}[u_i u_i^\top] = I_d/d$. Each matrix aggregates responses across sampled contexts. The implemented channel score composes these aggregated matrices, rather than computing and averaging a separate end-to-end Jacobian for every context.

Directions use `default_rng([seed,trial_index])`. The implementation first consumes one integer draw in $[0, 2^{31} - 1)$, then draws the Gaussian vector, converts it to FP32, and normalizes it. Preserving this draw order reproduces the direction sequence. Action-response pairs use identical initial diffusion noise for the base and perturbed inputs. The saved layer matrices are converted to FP32 before channel scoring.

**Cognition-token activation energy.** For the MLP input $x_\ell$ at the cognition token, write

$$z_\ell = \mathrm{SiLU}(W_{\mathrm{gate},\ell} x_\ell) \odot (W_{\mathrm{up},\ell} x_\ell), \tag{39}$$

$$o_\ell = W_{\mathrm{down},\ell} z_\ell. \tag{40}$$

Capture $z_\ell$, the input to the down projection, at the cognition-token position only. For channel $j$, compute

$$E_{\ell,j} = \frac{1}{500} \sum_{n=1}^{500} \left( z_{\ell,j}^{(n)} \right)^2. \tag{41}$$

Activation collection uses evaluation mode with gradients disabled and the vision-language model cast to BF16. Captured activations are saved in FP16 and converted to FP32 for the squared-activation average. Energy is not averaged over all image and text token positions.

**Action probes and fixed weights.** The score uses four probes $\mathcal{Q} = \{q_x, q_y, q_z, q_{r_z}\}$ in the flattened 112-dimensional action space. For each probe, the selected action dimension has the same positive coefficient at all 16 horizons, and all other entries are zero. Unit normalization makes each nonzero coefficient $1/\sqrt{16} = 1/4$. The stored scoring weights are

$$\begin{aligned} r_x &= 0.2366164178, \quad & r_y &= 0.2787458003, \\ r_z &= 0.3091658950, \quad & r_{r_z} &= 0.5133790970. \end{aligned} \tag{42}$$

Table 8: Final MLP intermediate dimensions. Layers are zero-indexed; each layer in a grouped row has the stated dimensions. The original width is 11008 in every layer.

| Layer indices | Removed channels per layer | Retained width per layer |
|---|---|---|
| 0–7 | 4352 | 6656 |
| 8 | 2688 | 8320 |
| 9 | 1920 | 9088 |
| 10 | 1280 | 9728 |
| 11 | 896 | 10112 |
| 12–14 | 640 | 10368 |
| 15–18 | 512 | 10496 |
| 19 | 640 | 10368 |
| 20 | 1024 | 9984 |
| 21 | 1536 | 9472 |
| 22 | 2432 | 8576 |
| 23 | 2560 | 8448 |
| 24 | 3968 | 7040 |
| 25 | 4224 | 6784 |
| 26–31 | 4352 | 6656 |

These weights are applied directly without normalization to sum to one. The configuration does not include separate probes for rotations about $x$ and $y$ or for the gripper.

Let $w_{\ell,j}$ be column $j$ of $W_{\mathrm{down},\ell}$. The implemented channel score is

$$I_{\ell,j} = E_{\ell,j} \sum_{q \in \mathcal{Q}} r_q \left( q^\top \widehat{A} \widehat{J}_\ell w_{\ell,j} \right)^2 . \tag{43}$$

All channels in all 32 MLPs are scored using this expression in FP32.

### A.6 Global Pruning and Hardware-Aligned Dimensions

**Global and per-layer budgets.** The unpruned model contains $32 \times 11008 = 352256$ MLP intermediate channels. A global removal ratio of 0.25 removes 88064 channels and retains 264192. The per-layer removal cap is 0.40, and removal counts must be multiples of 128. The effective maximum number removed from a layer is therefore

$$R_{\max} = 128 \left\lfloor \frac{\lfloor 0.40 \times 11008 \rfloor}{128} \right\rfloor = 4352, \tag{44}$$

or approximately 39.5% of that layer's original intermediate width.

**Score-ordered blocks.** Within each layer, stably sort channels by increasing $I_{\ell,j}$. Partition this sorted list into blocks of 128 and score a block by the arithmetic mean of its channel scores:

$$B_{\ell,b} = \frac{1}{128} \sum_{j \in \mathcal{B}_{\ell,b}} I_{\ell,j}. \tag{45}$$

These blocks are contiguous in score order, not necessarily in the original channel indices. Each layer contributes at most its first 34 blocks as candidates. Sort candidate blocks by the tuple $(B_{\ell,b}, \ell, b)$, using layer and block indices to break ties, and select a total of $88064/128 = 688$ blocks. For each layer, selected blocks must form a prefix of its score-ordered block list. Block means are accumulated in FP64 from the FP32 channel scores.

**Compact weight reconstruction.** For retained indices $\mathcal{K}_\ell$, reconstruct the dense projections as

$$W'_{\mathrm{gate},\ell} = W_{\mathrm{gate},\ell}[\mathcal{K}_\ell, :], \tag{46}$$

$$W'_{\mathrm{up},\ell} = W_{\mathrm{up},\ell}[\mathcal{K}_\ell, :], \tag{47}$$

$$W'_{\mathrm{down},\ell} = W_{\mathrm{down},\ell}[:, \mathcal{K}_\ell]. \tag{48}$$

Retained channels preserve their original index order. Inference uses these compact dense layers. Each removed channel eliminates $3 \times 4096 = 12288$ projection weights, for a total of $88064 \times 12288 = 1082130432$ removed weights. The three MLP projection matrices across all layers therefore contain 3246391296 weights after pruning, compared with 4328521728 before pruning. The 25% removal ratio refers to these MLP channels and projection weights, not to the entire VLA parameter count.

### A.7 Combined Online Execution

The combined implementation executes the following sequence at each control step:

1. Process episode or instruction changes and reset the corresponding cache, visual history, action history, and noise generator when required.
2. Determine reuse eligibility from the stored stability state, available cached cognition and plan, and refresh-interval counter.
3. On a refresh, process the current image with the full vision encoder and projector. Retain all visual tokens on the first processed frame; otherwise select 64 using Section A.3. Preserve original position IDs and run the language backbone with the compact MLP weights.
4. Generate a fresh action plan from the new cognition feature, update cognition and noise caches, evaluate plan stability against the existing history, append the fresh plan, and reset the reuse counter.
5. On a reuse step, run the action head with cached cognition and noise, apply Equation 19, append the resulting plan without a stability update, and increment the reuse counter. Visual encoding, projection, and language inference are skipped on this path.
6. Apply temporal action ensembling and robot-interface postprocessing, and emit the action for the current control step.

Structural pruning is applied offline before online inference. During execution, visual selection reduces the sequence entering the language backbone on refresh steps, while cognition reuse avoids that entire multimodal computation on eligible steps. Both paths retain action-head execution and the same downstream action processing.

## B Additional Analysis of Multi-Level Redundancy

We provide additional evidence for the three sources of redundancy targeted by ASSIST-VLA. Figure 3 analyzes temporal action consistency, complementary visual-token selection criteria, and the distribution of MLP-channel influence on final actions.

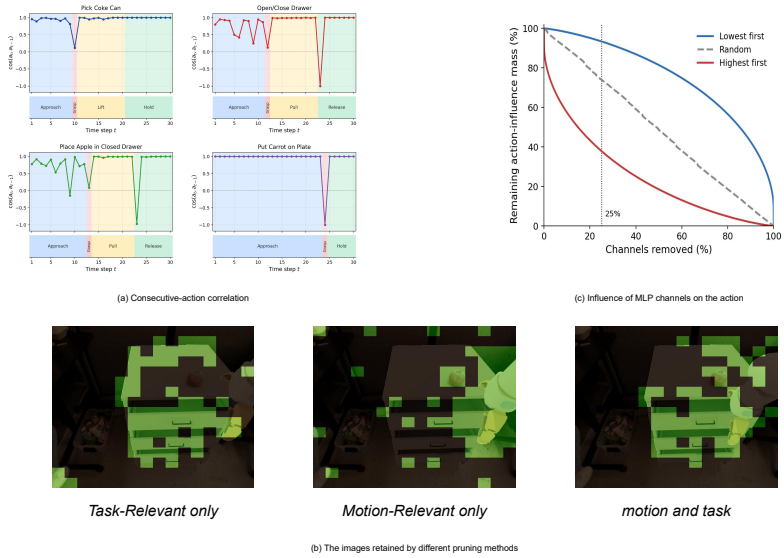



Figure 3: Additional analysis of multi-level redundancy in closed-loop VLA inference. (a) Cosine similarity between consecutive action plans across four SimplerEnv tasks. Action plans remain highly consistent during stable manipulation phases and change sharply near stage transitions. (b) Visual tokens retained by task-relevance-only pruning, motion-relevance-only pruning, and their combination. Task relevance concentrates on the instruction-specified target, while motion relevance captures the robot gripper and interaction region; combining both retains complementary task and motion cues. (c) Remaining action-influence mass after removing MLP channels in ascending, random, or descending order of action influence. Removing the lowest-influence $25\%$ of channels retains substantially more action-influence mass than random or highest-influence removal.