Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Semester Project

Systems Group, Department of Computer Science, ETH Zurich

Inference Optimization of Vision-Language-Action Models through Speculative Decoding

by

Ayoub Agouzoul

Supervised by

Dr. Wenqi Jiang

September 2025–December 2025

D INFK

Contents

Abstract

Autoregressive Vision-Language-Action (VLA) models enable robot control by predicting action tokens from visual observations, while conditioning on language instructions, but their sequential decoding limits inference throughput and thus control frequency. This project investigates whether lossless speculative decoding can accelerate VLA inference by using MiniVLA as a fast draft model and OpenVLA as the target model. We implement a speculative decoder wrapper that supports batched verification with KV cache and resolves draft/target tokenizer incompatibilities via token remapping, and we optimize the draft model with a minimal generation path and selective `torch.compile`. On an NVIDIA H100 NVL, the optimized MiniVLA reaches 16–17 Hz versus 5.49 Hz for OpenVLA, yet speculative decoding yields net slowdowns: with $\gamma = 6$ and exact acceptance we measure an acceptance rate of 27.5% and only $0.46\times$ throughput relative to the OpenVLA baseline, while relaxed acceptance improves α but does not reach break-even even under perfect acceptance. A GenZ-based theoretical analysis explains these results: the short action sequence length and large multimodal prefill leave little amortization headroom, and break-even would require substantially higher acceptance and/or a lower end-to-end cost ratio than achieved by independently trained VLA draft/target pairs. Overall, our findings indicate that practical VLA inference acceleration is more likely to come from more tightly aligned draft heads or from architectural and system-level optimizations than from vanilla speculative decoding. All code is available at `https://github.com/TheAyos/openvla-mini-specdec`.

1 Introduction

Following the shift towards foundational models in the field of machine learning, Vision-Language-Action (VLA) models have emerged through unified robotic policies for embodied AI [1], enabling advanced robot control by conditioning on visual observations and natural language instructions. These models leverage the powerful representations learned by Vision-Language Models (VLMs), which have been pretrained on internet-scale data, to develop generalist robot policies that can handle diverse environments and tasks.

However, inference latency still represents a critical challenge for the deployment of VLA models. State-of-the-art autoregressive VLA models such as OpenVLA achieve inference rates of approximately 5 Hz on modern GPUs, significantly impairing smoothness and reactiveness in robot manipulation. This latency bottleneck stems from the token generation inherited from the underlying language model backbone, where each action token must be generated sequentially.

Speculative decoding [2, 3] has emerged as a promising technique for accelerating autoregressive generation in Large Language Models (LLMs). The core insight is that some tokens are "easier" to predict than others. This is exploited by using a smaller, faster "draft" model to speculatively generate multiple tokens, which are then verified in parallel by the larger "target" model. When the draft model's predictions align with what the target model would have produced, significant speedups can be achieved without altering the output distribution.

This project investigates the application of speculative decoding to accelerate VLA inference, using MiniVLA[4] as the draft model and OpenVLA[5] as the target model. The remainder of this report is organized as follows: Section 2 provides background on VLA models and speculative decoding. Section 3 describes the methodology and implementation approach. Section 4 presents experimental and theoretical results. Section 5 discusses findings, challenges, and insights. Section 6 concludes with a summary.

2 Background and Related Work

2.1 Vision-Language-Action Models

The development of VLA models has progressed rapidly since the introduction of Google's RT-1 [6], the first transformer-based architecture to demonstrate that large pretrained models could effectively map visual and language inputs to robot actions. RT-1 processes short image sequences along with natural language task descriptions to output actions across 11 dimensions (7-dimensional robot arm control vector, 3-dimensional robot base position and robot control mode), with each dimension uniformly discretized into 256 bins, at a throughput of 3 Hz. RT-1 showed that data diversity is more important than quantity for generalization, especially when training on demonstrations from different robot morphologies.

RT-2 [7] extended RT-1's training recipe by co-training on vision-language data alongside robot demonstrations, preserving prior world knowledge and thus improving semantic generalization. The RT-2-X variant, trained on the Open X-Embodiment dataset [1], achieved state-of-the-art performance by leveraging data from multiple robotic platforms. The Open X-Embodiment dataset harmonizes differences in sensory inputs, action spaces, and robot capabilities across 22 robots and 60 datasets, enabling cross-embodiment learning. Experiments demonstrated that models trained on this dataset can transfer skills across different embodiments and improve generalization compared to single-robot datasets. It is now adopted as the primary (pre-) training corpus for modern VLA models such as OpenVLA.

The Prismatic VLM framework [8] provides a systematic investigation of vision-language model architectures that both OpenVLA and MiniVLA are based on. This work explores combinations of vision encoders and language model backbones to identify which configurations balance computational efficiency with accuracy.

The following VLA models are considered in this project:

OpenVLA [5] An open-source 7B parameter VLA built on the Prismatic VLM architecture, combining a dual vision encoder consisting of DINOv2 and SigLIP with a Llama-2 7B language model. It is trained through imitation learning on a mixture of datasets from Open X-Embodiment and further fine-tuned on the respective evaluation datasets. Similar to RT-1 [6], OpenVLA discretizes continuous robot actions into 256 bins per dimension and represents each bin as a discrete action token. It then treats action prediction as an autoregressive prediction task over these tokens.

MiniVLA [4] A lightweight VLA designed to achieve competitive performance with a smaller computational footprint. Key design choices include replacing the `Llama-2-7B` backbone with `Qwen2.5-0.5B` and using the same dual vision encoder as OpenVLA. MiniVLA comes in other variants, supporting vector-quantized [9] action chunking for improved accuracy and multi-image support for history or camera inputs for additional points of view. We use the base variant for this project. The size difference between MiniVLA ($\sim$1B parameters) and OpenVLA ($\sim$7B parameters), combined with their architectural similarity, makes them a natural candidate for a target/draft model pair for speculative decoding.

SmolVLA [10] An efficient VLA model that can run on a single consumer GPU at decent speeds. It uses a flow matching "action expert" head to generate continuous actions through a denoising process. The policy is built on top of `SmolVLM-500M`, a compact vision-language backbone. The action expert is a lightweight Conditional Flow Matching Transformer [11] conditioned on intermediate backbone features via configurable layer skipping. This flexibility was initially interesting for this project as it could allow for an in-depth analysis of draft model design and its impact on speculative decoding performance, but it was ultimately not used because of its incompatibility with speculative decoding.

2.2 Speculative Decoding

2.2.1 Vanilla Speculative Decoding

Speculative decoding [2, 3] accelerates Transformer-based [12] autoregressive generation by leveraging (i) the ability of the target model to verify multiple draft tokens in parallel using a single forward pass (batching) and (ii) the draft model's speed relative to the target model. The main speculative decoding steps are:

1. **Drafting**: The draft model M_d autoregressively generates a sequence of γ candidate tokens.
2. **Verifying**: The target model M_t processes all γ draft tokens in a single forward pass, computing the probability of each token
3. **Rejection Sampling**: Draft tokens are accepted or rejected by comparing the probabilities assigned by M_d and M_t. In the case of rejection, the sampling distribution is adjusted to ensure that the final output distribution exactly matches that of the target model.

The theoretical speedup depends on:

- The acceptance rate α (probability that draft tokens are accepted),
- the cost ratio c (time for draft forward pass / time for target forward pass), and
- the number of draft tokens to generate during each drafting step γ.

2.2.2 EAGLE

EAGLE [13] introduces a different approach to speculative decoding by performing autoregression at the *feature level* rather than the token level, and by using tree attention to generate multiple tokens per forward pass through draft trees. The key insight is that predicting hidden states from the second-to-top-layer of the target model is more tractable than directly predicting tokens, as feature sequences are more structured than token sequences. EAGLE is a lightweight draft head consisting of a single-layer Transformer decoder with 0.24B–0.99B parameters (corresponding to 7B–70B target models) that takes the current feature sequence and an advanced token sequence as input to predict future features. The frozen language model head from the target model is then used to predict token probabilities. EAGLE achieves 2.7–3.5$\times$ speedup on LLaMA2-Chat 70B inference latency compared to autoregressive decoding, while preserving accuracy.

EAGLE-2 [14] extends this framework by introducing a dynamic draft tree that adapts how candidate token sequences are generated rather than using a draft tree with fixed shape. EAGLE-2 leverages the critical observation that the first version of the EAGLE draft model is *well-calibrated*, meaning that its confidence scores closely approximate the acceptance probabilities under the target model. This allows to dynamically grow and prune a draft tree to construct a set of promising draft paths which are then flattened into a sequence and verified by the target model using a custom attention mask. This provides lossless acceleration approximately 20–40% faster than EAGLE, achieving empirical speedup ratios of 3.05–4.26× thanks to an average acceptance length of 4–5.5 tokens per draft-verify step.

2.2.3 Spec-VLA

Spec-VLA [15] represents the first application of speculative decoding to VLA models. The authors show that exact speculative decoding yields only minor speedups for OpenVLA, due to low acceptance rates arising from the difficulty of action prediction. To address this, Spec-VLA replaces exact matching by *relaxed acceptance*, which is based on the relative distances between discretized action bins, allowing draft tokens to be accepted when they fall within a threshold distance of the target prediction. The implementation uses EAGLE-2's dynamic draft tree strategy with tree attention for parallel verification. Empirically, Spec-VLA improves acceptance length by 25–44% and achieves 1.22x–1.42x speedup compared to the autoregressive baseline on LIBERO benchmarks with minor degradation of success rates. However, this approach is *lossy* with respect to the target model's distribution and might require dataset-specific tuning of the relaxation threshold. These limitations, combined with the complexity of training EAGLE-style draft heads, motivated our investigation of vanilla speculative decoding as a baseline lossless approach.

2.3 Alternative Acceleration Approaches

Several concurrent and recent works have explored alternative approaches to VLA inference optimization that do not rely on speculative decoding.

Parallel Action Decoding and Action Chunking

Recent work on fine-tuning OpenVLA [16] demonstrated that replacing sequential autoregressive action prediction with parallel decoding can achieve significant throughput improvements. By predicting all action tokens simultaneously rather than one-by-one, the authors report a 3–4× increase in generation throughput while simultaneously improving task success rates by 15–20% when combined with action chunking. Further architectural changes include replacing the language model decoder with a direct MLP mapping to continuous action spaces, eliminating discretization artifacts. This approach achieved state-of-the-art performance on the LIBERO benchmark. One limitation of this approach is that it requires architectural modifications and retraining, whereas speculative decoding can theoretically accelerate existing models without modification.

Hardware-Level Optimization

Recent work [17] demonstrated that careful CUDA graph optimization can achieve 30 Hz inference on the π_0 [18] model, enabling reactive control at camera frame rates, allowing the policy to catch a falling pen. The primary optimization involves using CUDA graphs to eliminate kernel launch overhead due to Python code running on the CPU. Secondary optimizations include kernel customization and optimizations to further reduce kernel launches, with diminishing returns. However, CUDA graph optimization is not directly applicable to all VLA models, as it has many strict requirements, including static input shapes, no branching, and no dynamic control flow.

Continuous Speculative Decoding

Recent work on continuous speculative decoding (CSpD) for autoregressive image generation [19] extends speculative decoding to continuous output distributions. Unlike the discrete version which operates over discrete token probabilities, CSpD performs verification directly in continuous latent space by adapting the acceptance-rejection sampling scheme to diffusion denoising trajectories. To improve the initially very low acceptance rates, the method aligns noise schedules and intermediate representations between the draft and target models, yielding around 2× inference speedup with no degradation in output quality. Unlike parallel decoding or action chunking, this approach preserves the autoregressive structure and can be applied post hoc to existing continuous autoregressive models. Early attempts were made to apply Continuous Speculative Decoding to SmolVLA, but it was found to be incompatible with the model's design.

2.4 The LIBERO Benchmark

LIBERO [20] is a simulation benchmark for evaluating language-conditioned robotic manipulation policies across a wide range of tasks. It is designed to test generalization, compositional reasoning, and long-horizon planning. The benchmark includes the following task suites:

- LIBERO-Spatial: 10 tasks focusing on spatial reasoning and arrangement
- LIBERO-Object: 10 tasks centered on object manipulation
- LIBERO-Goal: 10 tasks requiring goal-directed planning
- LIBERO-10: 10 diverse long-horizon tasks

- LIBERO-90: A comprehensive set of 90 tasks (different than the other subsets) covering the full distribution of the benchmark

Success rates are computed as the percentage of evaluation rollouts in which the task is successfully completed within a fixed time horizon. LIBERO uses the RLDS format for data storage and provides standardized training and evaluation protocols.

2.5 GenZ: LLM Inference Performance Modeling

Analyzing LLM inference performance across diverse hardware platforms and model architectures is essential for understanding the computational requirements and bottlenecks of these systems. However, running experiments on real hardware is expensive and time-consuming, particularly when exploring large design spaces spanning multiple models, hardware configurations, and serving optimizations.

GenZ [21] (Generative LLM Analyzer) is an analytical tool for estimating LLM inference latency without actual hardware execution. The tool uses roofline modeling to compute upper bound estimates on latency for prefill (typically compute-bound) and decode (typically memory-bandwidth-bound at low batch sizes) phases. GenZ models each operator's runtime based on the number of floating-point operations (C_{op}) and memory accesses (M_{op}), combined with hardware-specific efficiency factors derived from profiling real systems.

The framework consists of three key components: (1) a *model profiler* that extracts operator dimensions and computation patterns from LLM architectures, (2) an *NPU characterizer* that models accelerator compute capabilities and memory bandwidth, and (3) a *platform characterizer* that simulates multi-dimensional network topologies for distributed inference. GenZ supports various LLM architectures including dense models, mixture-of-experts (MoE), and Mamba-based models, as well as serving optimizations such as tensor parallelism, pipeline parallelism, chunked prefill, and speculative decoding.

Validation against real hardware platforms (NVIDIA H100, A100, Intel Gaudi2, AMD MI300X, and SambaNova SN40L) demonstrates high accuracy, achieving a maximum geometric mean error of 5.82% across different architectures [21]. This makes GenZ suitable for theoretical analysis of inference performance without requiring access to physical hardware. We use GenZ in Section 4.5 to analyze the theoretical speedup potential of speculative decoding for OpenVLA and MiniVLA.

3 Methods

3.1 Initial Setup

Before implementing speculative decoding, we reproduced published benchmark results to establish baselines and verify our experimental setup. The throughput measurement script initializes a LIBERO task environment, performs 10 warmup iterations to stabilize the GPU state, measures wall-clock time for 100 action predictions, and reports throughput in actions per second (Hz). For accuracy evaluation on LIBERO, we followed OpenVLA's protocol. Each task is evaluated for 50 episodes to reduce variance, with a maximum of 400 steps per episode. Success is determined by task-specific completion criteria, and results are aggregated as success rate per task suite over a fixed seed.

3.2 Finetuning for Distribution Alignment

A fundamental requirement for effective speculative decoding is that the draft and target models produce similar token distributions. For otherwise, their output distributions may diverge significantly (e.g., when trained on different datasets), leading to low acceptance rates and no speedup or even a slowdown.

3.2.1 MiniVLA Finetuning Attempts

We first attempted to fine-tune MiniVLA on LIBERO-Spatial to create a more aligned draft-target pair. However, these attempts failed due to unexpected results. Training loss decreased to extremely low values (0.0003 at 162k steps), far below the original MiniVLA LIBERO-90 checkpoint (0.07 at 122k steps). And despite low training loss, evaluation success rate was 0% on both LIBERO-Spatial and LIBERO-90. This anomaly suggests issues with the default configurations for training and data augmentation. As the MiniVLA article and codebase did not provide further details on training recipes, and due to time and resource constraints, we were unable to diagnose the root cause.

3.2.2 OpenVLA Finetuning on LIBERO-90

We instead decided to fine-tune OpenVLA using LoRA on the same dataset as MiniVLA, LIBERO-90. For dataset preparation, we generated LIBERO-90 trajectories with no-op filtering, yielding 3,950 episodes, and converted the dataset to RLDS format using the `rlds_dataset_builder` [22] tool. We used a training configuration with LoRA rank 32, batch size 16×2 with gradient accumulation, learning rate 5×10^{-4}, and enabled image augmentation. Training ran for 125,000 steps on 4× H100 GPUs (approximately 24 hours). Token accuracy reached 95.5% and L1 loss was 0.0012.

3.3 Speculative Decoding Implementation

This section details the challenges encountered when implementing speculative decoding with OpenVLA/MiniVLA and the solutions developed to overcome them. The implementation, `VLASpeculativeDecoderBatchedLM`, represents a working speculative decoding system for VLAs that addresses several technical barriers.

3.3.1 Some Implementation Challenges

OpenVLA's forward pass does not support batched verification with cached key-value pairs. The underlying Prismatic VLM backbone's implementation enforces that cached generation only works with single tokens, and the unimodal forward path does not support `past_key_values`. This design prevents the standard speculative decoding verification step, which requires processing multiple draft tokens while utilizing the cached KV states from previous tokens. Inspired by Spec-VLA's implementation, we developed a solution that bypasses the multimodal path by extracting the embedding layer from the language model, converting draft token IDs directly to embeddings, and calling the language model directly with `inputs_embeds` and cached KV states to avoid the multimodal forward path's restrictions.

MiniVLA and OpenVLA use different language model backbones with incompatible tokenizers (e.g., different vocabulary sizes), as OpenVLA uses the last 256 tokens of Llama-2 vocabulary for action bins (tokens 31744–31999), while MiniVLA uses 256 extra tokens appended to Qwen2.5 vocabulary (tokens 151665–151920). This requires token remapping between the two vocabularies during the verification step. We implement bidirectional token mapping as follows:

- `_draft_token_to_target`: Maps MiniVLA action tokens to equivalent OpenVLA tokens by computing the action bin index and remapping to the target vocabulary range
- `_target_token_to_draft`: Maps OpenVLA tokens back to MiniVLA vocabulary for cache updates after rejection
- `_remap_logits_draft_to_target`: Remaps draft probability distributions to the target vocabulary space for proper rejection sampling

3.3.2 Relaxed Acceptance Criterion

Following Spec-VLA, we implement relaxed acceptance based on absolute action bin distance. For action tokens, adjacent bins represent similar continuous actions. A draft token predicting bin 127 when the target would produce bin 128 results in nearly identical robot behavior. The relaxed acceptance criterion accepts a draft token if:

$$|b_{\text{draft}} - b_{\text{target}}| \leq r \tag{1}$$

where b_{draft} and b_{target} are the action bin indices and r is the relaxation threshold. This trades exact distribution matching for higher acceptance rates, potentially improving speedup at the cost of slight deviation.

3.3.3 MiniVLA Inference Optimization

A critical discovery during speculative decoding implementation was that initial throughput measurements were significantly slower than expected. This was due to low acceptance rates and high cost ratio between draft and target models. To address this, we developed `MiniVLAFastPath`, a wrapper around MiniVLA's forward pass that attempts to use PyTorch optimizations to speed up inference.

The standard MiniVLA inference involves several inefficiencies that we address in the `MiniVLAFastPath` module to reduce rollout overhead.

The HuggingFace Transformers `generate()` function, designed for variable-length text generation, introduces unnecessary branching, especially for our use case with a fixed 7-token greedy generation. The "fast path" replaces `generate()` with a minimal autoregressive loop: a single prefill pass over the image and cached prompt with KV caching enabled, followed by decoding of the 7 action tokens using only the LLM backbone and `torch.argmax` for token selection without synchronization until final output.

Each inference call re-tokenizes the instruction prompt, despite the instruction being constant throughout a robot rollout (typically hundreds of steps). We thus implement instruction caching: because rollouts reuse the same instruction for hundreds of steps, the prompt is built and tokenized once, stored as device tensors, and only refreshed via `set_instruction()` when tasks change.

Additionally, selective `torch.compile` is supported, allowing compilation of the LLM backbone (recommended), optional compilation of the vision backbone, and configurable optimization modes (e.g.,"default" or "max-autotune-no-cudagraphs") provided by PyTorch. We present the results of different compilation strategies in Section 4. A key implementation detail is that `dynamic=True` must be specified for the LLM backbone because KV cache shapes change with prompt length and decode steps. Setting `fullgraph=False` is required for Qwen2.5 models due to graph breaks that prevent full-graph compilation.

4 Results

All experiments were conducted on an NVIDIA H100 NVL 96GB GPU. The primary software environment used PyTorch 2.2.2 with CUDA 13.0, flash-attn 2.5.5, and transformers 4.40.1.

4.1 Throughput Benchmarks

Table 1 presents throughput measurements for various VLA configurations. We observe that MiniVLA provides ∼1.6× speedup over OpenVLA with its smaller backbone, and that compilation on the LLM and vision backbones yields the best speedup, reaching ∼16 Hz.

Table 1: Throughput comparison of VLA models on NVIDIA H100 GPU. Compiled models use `torch.compile(mode=‘‘default’’)`.

Model	Throughput (Hz)	Relative Speed
OpenVLA-7B (base)	5.49	1.0×
MiniVLA (base)	8.97	1.63×
MiniVLA (compiled LLM only)	13.76	2.51×
MiniVLA (compiled Vision only)	9.68	1.76×
MiniVLA (compiled LLM+Vision)	**15.88**	**2.89**×

4.2 MiniVLA `torch.compile()` Ablation Study

To systematically evaluate the impact of different compilation strategies, we conduct an ablation study on the `MiniVLAFastPath` class using full LIBERO-90 rollouts. The benchmark ran complete rollouts with precise GPU timing using CUDA events. Table 2 and Figure 1 present the full results.

We observed early in testing that component-wise compilation is crucial. Compiling the full model with default settings provides no gain, but selectively compiling only the LLM backbone yields significant speedup (1.53×). The LLM backbone dominates inference time due to its autoregressive token generation for 7 action tokens, making it the main compilation target. Even though compiling the vision backbone alone provides only 1.08× speedup, combining LLM and vision compilation achieves the best overall result (1.77×), adding a marginal benefit. This suggests that PyTorch's optimizations are more effective on architecturally homogeneous components.

Interestingly, the `max-autotune-no-cudagraphs` mode, which explores more kernel configurations, achieved comparable but slightly lower performance than the default mode (Figure 3). For LLM-only compilation, max-autotune achieved 1.49× speedup compared to 1.53× for default mode. For LLM+Vision compilation, max-autotune achieved 1.73× compared to 1.77× for default mode. This suggests that inference is not mainly bottlenecked by the particular operation

Table 2: MiniVLAFastPath compilation ablation study on LIBERO-90. GPU timing measured with CUDA events during full task rollouts. All configurations succeeded on the test episode. The LLM+Vision (`default` mode) configuration achieves the best throughput. The `reduce-overhead` mode systematically fails on the Qwen2.5 backbone, as it attempts to use CUDA graphs.

Configuration	**Mean (ms)**	**Hz**	**Speedup**
Baseline (no compile)	111.5	8.97	1.00×
Default compile mode			
LLM only	72.7	13.76	1.53×
Vision only	103.3	9.68	1.08×
LLM + Vision	**63.0**	**15.88**	**1.77**×
max-autotune-no-cudagraphs mode			
LLM only	74.7	13.39	1.49×
Vision only	103.0	9.71	1.08×
LLM + Vision	64.5	15.51	1.73×
reduce-overhead mode			
Vision only	97.4	10.27	1.15×
Mixed modes			
LLM (default) + Vision (max-autotune)	63.1	15.85	1.77×
LLM (default) + Vision (reduce-overhead)	63.7	15.71	1.75×

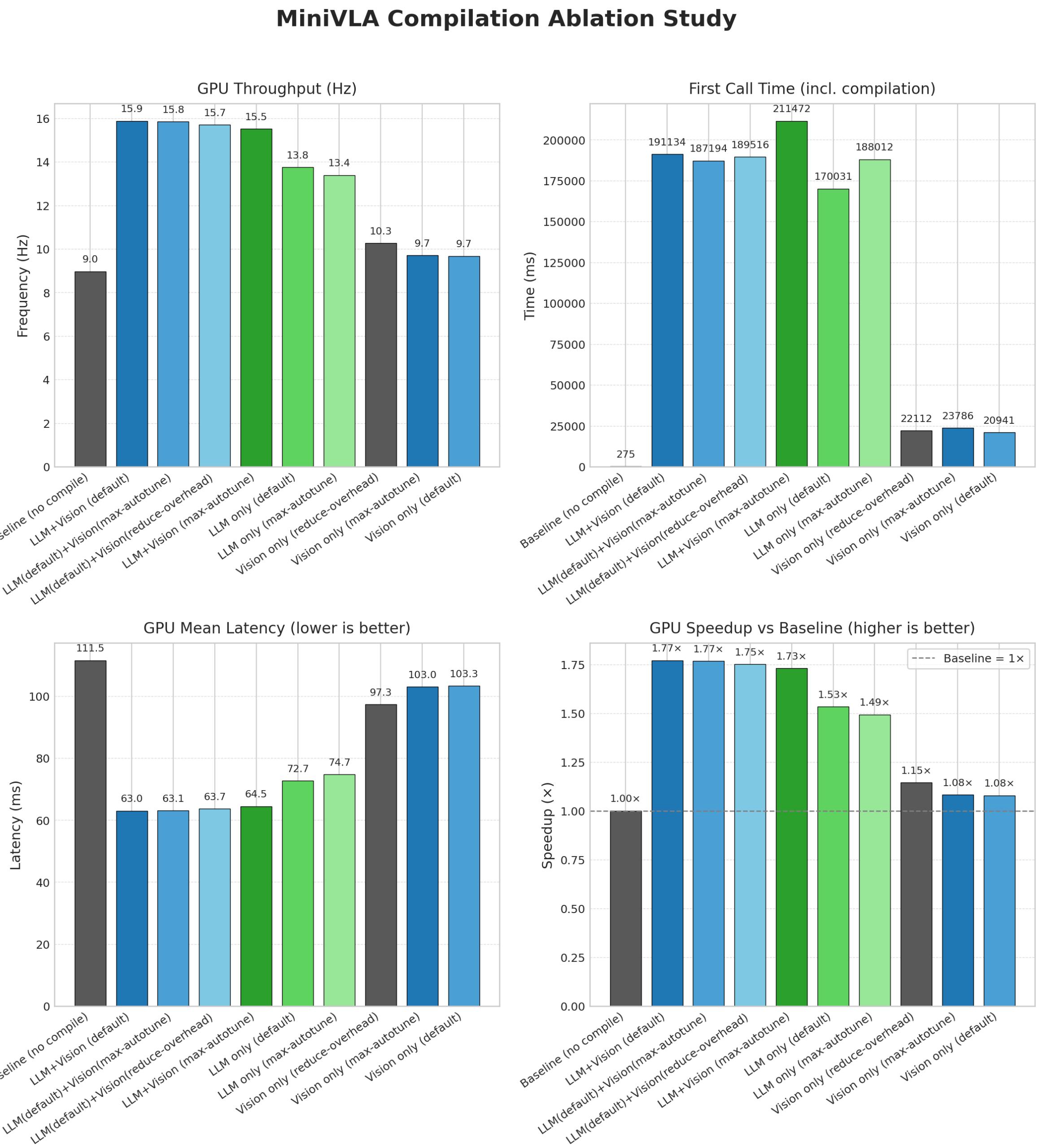

Figure 1: MiniVLAFastPath compilation ablation results. GPU throughput, compilation time, mean latency, and speedup vs baseline. The LLM+Vision (default) configuration achieves the best throughput at 15.88 Hz (1.77× speedup).

kernels that MiniVLA uses, which would explain why autotuning does not provide further benefits compared to the `default` compilation mode. We profiled the max-autotune mode and observed that although the compiled regions run faster, a quite high TorchDynamo cache overhead is introduced, which leads to periods where the GPU is idle and ultimately performs worse than the default mode. This can be clearly seen in Figure 2

Some remarks follow from these results for the speculative decoding approach. First, the cost ratio (as defined by Leviathan et al. [2]) can be computed from empirical measurements. With the optimized MiniVLA reaching 17.22 Hz (58 ms) and unoptimized OpenVLA at 5.49 Hz (182 ms), the current empirical cost ratio is $c = 58/182 \approx 0.32$. However, this is still higher than ideal for substantial speculative decoding speedup, as typical LLM speculative decoding benefits from cost ratios below 0.1. Moreover, the optimized draft model provides a meaningful throughput speedup ($\approx 3\times$ over target), and hints at the potential of other optimization techniques (e.g., custom kernels) to further improve draft model throughput and, ultimately, the speculative decoding performance.

4.3 LIBERO Benchmark Results

Table 3 presents success rates on LIBERO benchmark suites. Our OpenVLA-OFT reproduction matched or exceeded the paper results. SmolVLA reproductions showed lower performance than reported, possibly due to hyperparameter differences. OpenVLA LoRA finetuning on LIBERO-90 achieved a 71.4% success rate, comparable to MiniVLA. It is interesting to note that continued finetuning, beyond loss convergence, showed better success rates.

4.4 Speculative Decoding Results

We evaluated the `VLASpeculativeDecoderBatchedLM` implementation with systematic ablations over the speculation depth γ and relaxation threshold r. All experiments used the fine-tuned OpenVLA checkpoint (125k steps, 95.5% token accuracy) as target and the available MiniVLA LIBERO-90 checkpoint as draft, evaluated on LIBERO-90 task 0. Table 4 presents the baseline throughput measurements used to compute speedups.

4.4.1 Speculative Decoding Performance

With $\gamma = 6$ and no relaxation ($r = 0$), speculative decoding achieved:

- a throughput of 2.50 Hz (400 ms per action),
- a speedup of $0.46\times$ ($2.2\times$ slower than target baseline),
- an acceptance rate of 27.5%, and
- 0.96 tokens per target forward.

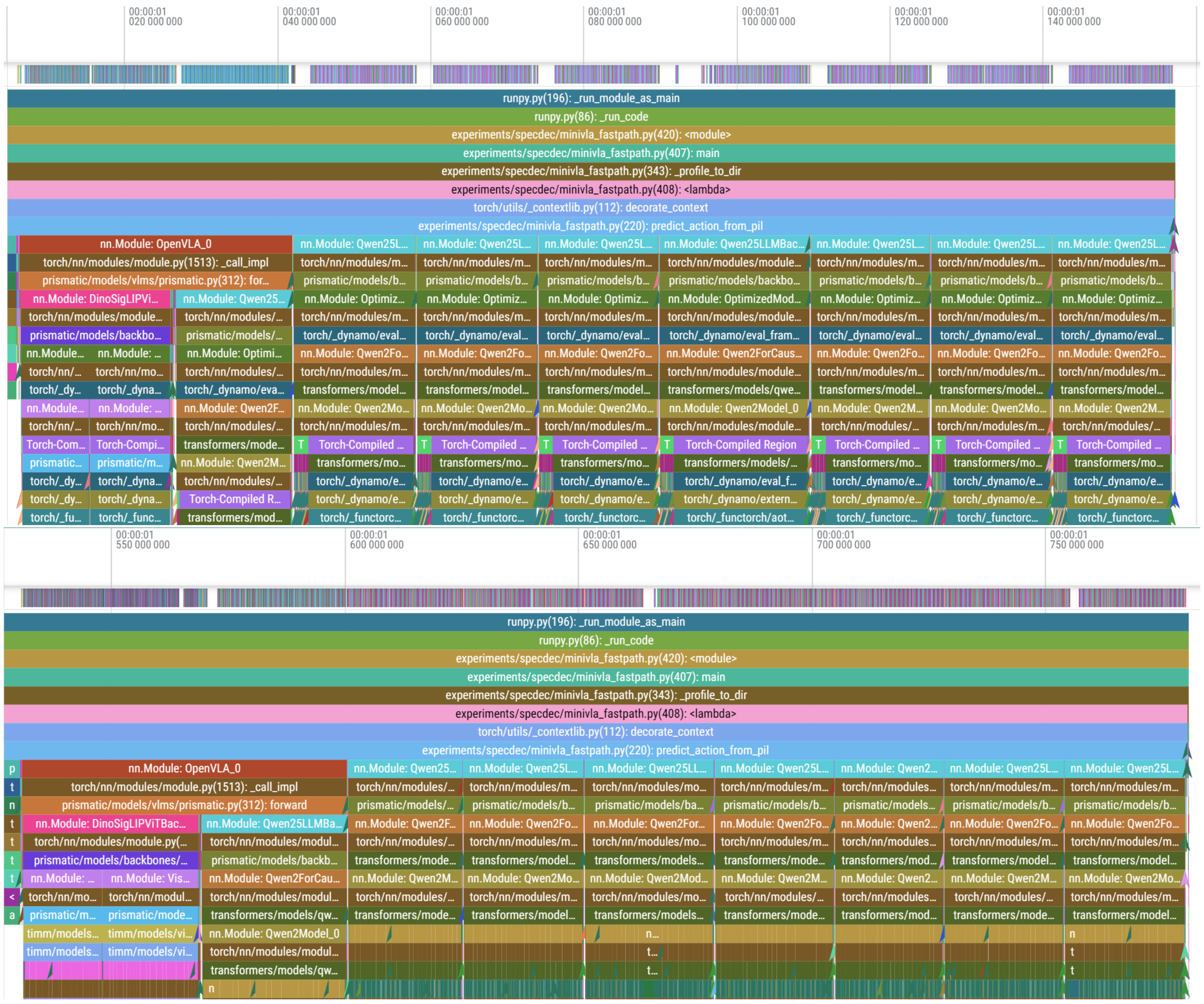
00:00:01 020 000 000
00:00:01 040 000 000
00:00:01 060 000 000
00:00:01 080 000 000
00:00:01 100 000 000
00:00:01 120 000 000
00:00:01 140 000 000
runpy.py(196): _run_module_as_main
runpy.py(86): _run_code
experiments/specdec/minivla_fastpath.py(420): <module>
experiments/specdec/minivla_fastpath.py(407): main
experiments/specdec/minivla_fastpath.py(343): _profile_to_dir
experiments/specdec/minivla_fastpath.py(408): <lambda>
torch/utils/_contextlib.py(112): decorate_context
experiments/specdec/minivla_fastpath.py(220): predict_action_from_pil
nn.Module: OpenVLA_0
torch/nn/modules/module.py(1513): _call_impl
nn.Module: Qwen25LLMBac…
Torch-Compiled Region
00:00:01 550 000 000
00:00:01 600 000 000
00:00:01 650 000 000
00:00:01 700 000 000
00:00:01 750 000 000
runpy.py(196): _run_module_as_main
runpy.py(86): _run_code
experiments/specdec/minivla_fastpath.py(420): <module>
experiments/specdec/minivla_fastpath.py(407): main
experiments/specdec/minivla_fastpath.py(343): _profile_to_dir
experiments/specdec/minivla_fastpath.py(408): <lambda>
torch/utils/_contextlib.py(112): decorate_context
experiments/specdec/minivla_fastpath.py(220): predict_action_from_pil
nn.Module: OpenVLA_0
torch/nn/modules/module.py(1513): _call_impl
prismatic/models/vlms/prismatic.py(312): forward
nn.Module: DinoSigLIPViTBac…
nn.Module: Qwen25LLMBa…
nn.Module: Qwen2Model_0

Figure 2: Profile traces of a complete MiniVLA forward pass on the first observation of LIBERO-90 task 0 after 5 warmup steps. Visualized using Perfetto [23]. Top: MiniVLA is compiled with torch.compile mode '`max-autotune-no-cudagraphs`'. The TorchDynamo cache overhead introduced by max-autotune is visible in the top row as the additional empty gaps in white, and in light green in the bottom row of the trace. Bottom: MiniVLA without compilation. Note: Top and Bottom do not use the same time scale.

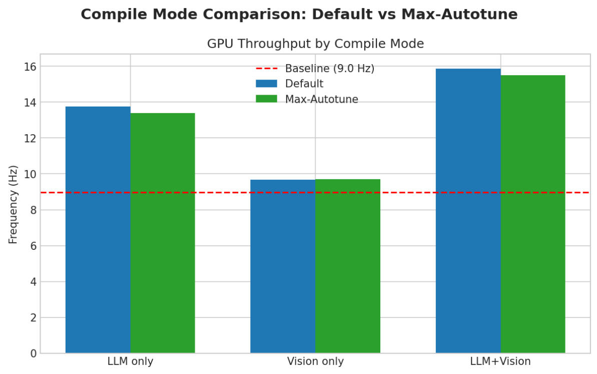

Figure 3: Comparison of default vs max-autotune compilation modes. Default mode consistently outperforms max-autotune across configurations including the LLM backbone. The red dashed line indicates baseline (uncompiled) throughput.

Table 3: LIBERO benchmark success rates (%). Bold values are paper-reported results and others are our reproductions. 50 episodes per task. Columns 2-6 represent LIBERO subsets. For models that were fine-tuned, the number of steps is indicated in parentheses.

Model	Spatial	Object	Goal	10	90	Avg
OpenVLA-OFT (paper)	**96.2**	**98.3**	**96.2**	**90.7**	–	**95.3**
OpenVLA-OFT (reproduced)	98.4	98.6	97.2	93.6	–	96.95
SmolVLA (paper, N=32)	**90**	**96**	**92**	**71**	–	**87.3**
SmolVLA (reproduced, N=32)	81.2	93.0	82.0	46.4	–	75.65
SmolVLA (paper, N=8)	**77**	**88**	**86**	**49**	–	**75.0**
SmolVLA (reproduced, N=8)	71.8	81.6	81.6	46.2	–	70.3
SmolVLA (reproduced, N=4)	45.8	61.0	57.4	33.2	–	49.4
MiniVLA (paper)	–	–	–	–	**61.4**	–
MiniVLA (reproduced)	–	–	–	–	48.8	–
OpenVLA LoRA FT (70k steps)	–	–	–	–	63.3	–
OpenVLA LoRA FT (125k steps)	–	–	–	–	71.4	–

Table 4: Baseline throughput measurements for speculative decoding evaluation.

Model	Mean Time (ms)	Throughput (Hz)	Relative
OpenVLA (target)	181.99	5.49	1.00×
MiniVLA + FastPath (draft)	58.07	17.22	3.13×
(Empirical) Cost ratio c	$58.07/181.99 = 0.32$		

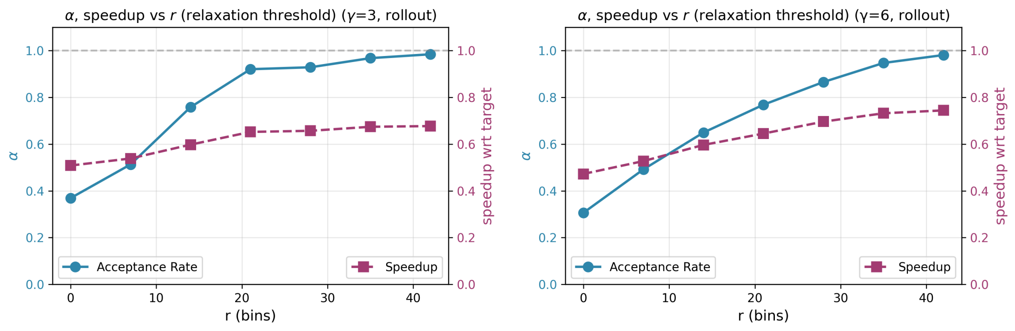

Figure 4: Acceptance rate α and speedup vs. relaxation threshold r for $\gamma = 3$ (left) and $\gamma = 6$ (right). Increasing r improves acceptance rate, but speedup remains below $1.0\times$ even with higher acceptance rates. The longer speculation depth ($\gamma = 6$) shows slightly better speedup but still cannot reach break-even.

The low acceptance rate (27.5%) combined with the overhead of managing two models results in a net slowdown rather than speedup.

4.4.2 Effect of r and γ

Figure 4 shows how acceptance rate and speedup vary with the relaxation threshold r for different speculation depths. We observe that acceptance rate increases monotonically with r. Speedup improves with higher r, but even with high acceptance rates, speculative decoding remains slower than the baseline due to the overhead of both draft and target. The effect of the relaxation threshold (which trades accuracy for acceptance) is still not enough in this case to achieve a speedup. Figure 5 shows the effect of varying γ for different relaxation thresholds r. All $\gamma > 0$ values result in slowdown regardless of r. With exact matching ($r = 0$), acceptance rate is roughly constant at 30–40% across γ values. With relaxed acceptance ($r = 7$), acceptance rate slightly increases to 50–65% and with maximal relaxation ($r = 255$) which leads to 100% acceptance, speedup remains below $1.0\times$.

4.4.3 Why Speculative Decoding Fails

Before the theoretical analysis in Section 4.5, we provide some early insights into why speculative decoding fails. First, the cost ratio is insufficient: with $c = 0.32$, the draft model is not fast enough compared to the target, and does not break even. Second, the acceptance rate is low at 27.5%, meaning that most draft tokens are rejected and require additional target forward passes. Finally, the sequences are short, with only seven tokens to generate for each observation, leaving little opportunity for speculation to amortize its overhead, whereas

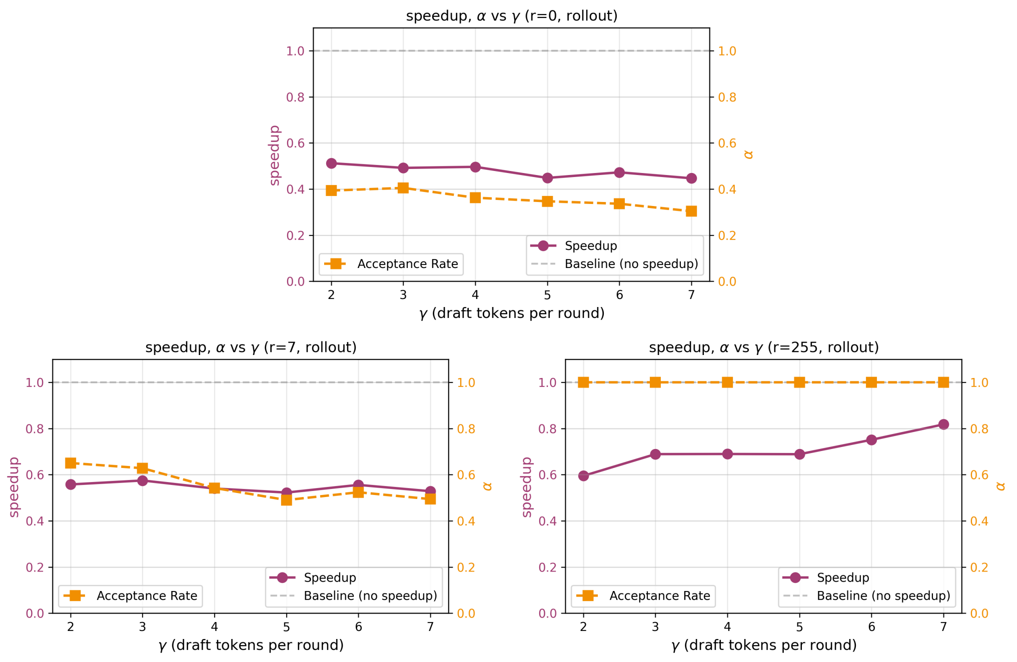

Figure 5: Speedup and acceptance rate α vs. speculation depth γ for different relaxation thresholds: $r = 0$ (top, exact matching), $r = 7$ (bottom right, small relaxation), and $r = 255$ (bottom left, maximum relaxation). All values of $\gamma > 0$ result in slowdown (speedup < 1.0) regardless of relaxation threshold.

speculative decoding is most effective for long sequences where occasional multi-token acceptances yield meaningful savings.

4.5 Theoretical Analysis with GenZ

To complement our empirical measurements and understand what causes speculative decoding to fail in our scenario, we conduct a theoretical analysis using GenZ [21]. This section presents the theoretical predictions and compares them with our empirical observations. We extend GenZ to model the OpenVLA and MiniVLA architectures by defining custom configurations for the vision backbones (DINOv2, SigLIP) and the MiniVLA LLM backbone (Qwen2.5-0.5B), which were not included in the default model registry. Using GenZ with H100 NVL specifications (3958 GB/s memory bandwidth, 989 TFLOPS BF16), we compute the latency estimates in Table 5. The theoretical decode latency ratio of 13.5× reflects the ∼14× parameter ratio between Llama-2-7B and Qwen2.5-0.5B.

Table 5: GenZ theoretical latency breakdown for VLA components (H100 NVL). The number of input tokens to the LLM backbone is approximately 281 tokens (256 vision + ∼25 language instruction).

Component	**Latency (ms)**	**Bound**
Vision (DINOv2 + SigLIP)	0.36	Compute
OpenVLA LLM prefill (281 tokens)	3.92	Compute
OpenVLA LLM decode (per token)	3.15	Memory
MiniVLA LLM prefill (281 tokens)	0.41	Compute
MiniVLA LLM decode (per token)	0.23	Memory
Decode latency ratio	3.15/0.23 = 13.5×	

4.5.1 Theoretical Speedup Analysis

Following Leviathan et al. [2], the speedup from speculative decoding depends on the cost ratio $c = T_{\text{draft}}/T_{\text{target}}$ (time for a forward pass of the draft model / time for a forward pass of the target model) and acceptance rate α:

$$\text{Speedup} = \frac{E[\text{tokens}]}{1 + c \cdot \gamma} = \frac{1 - \alpha^{\gamma+1}}{(1-\alpha)(1 + c \cdot \gamma)} \tag{2}$$

where $E[\text{tokens}] = (1 - \alpha^{\gamma+1})/(1 - \alpha)$ is the expected number of tokens accepted per speculation round, and speedup is the expected improvement in total walltime. Figure 6 shows the theoretical speedup landscape, and also predicts that the current OpenVLA/MiniVLA setup yields a slowdown. Even with theoretical $c = 0.074$ (13.5× faster draft), we are in the slowdown region. At the same cost ratios, achieving 1.5× speedup requires $\alpha > 50\%$ with the

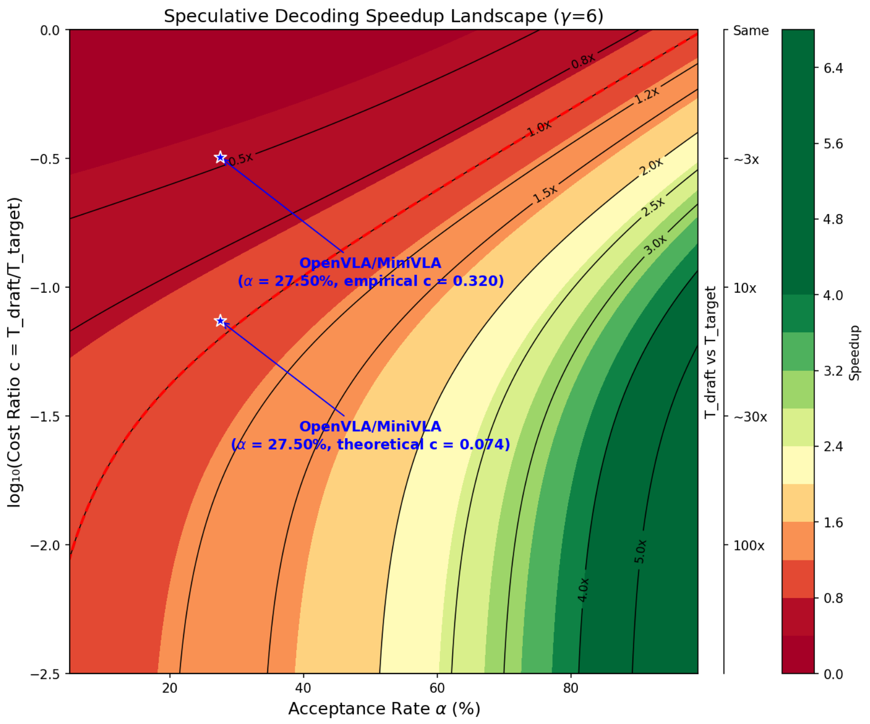

Figure 6: Speculative decoding speedup landscape as a function of acceptance rate α and cost ratio c. Two stars mark the OpenVLA/MiniVLA operating points with both empirical ($c = 0.32$) and theoretical ($c = 0.074$) cost ratios, and empirically observed $\alpha = 27.5\%$. The red dashed contour indicates break-even (speedup = 1.0×).

theoretical cost ratio, and $\alpha > 85\%$ with the empirical cost ratio. The figure also hints at the importance of the acceptance rate over the cost ratio for speculative decoding: even if T_{draft} is 100× faster than T_{target}, the acceptance rate needs to be at ≈40% to get a 1.5× speedup.

For a better understanding of the cost ratio-acceptance rate relationship, we predict the break-even acceptance rates for different speedup targets in Figure 7. For our setup with $\gamma = 6$, break-even requires $\alpha \geq 31\%$ with theoretical cost ratio, while with empirical cost ratio break-even requires $\alpha \geq 68\%$. For 1.5× speedup, we would need $\alpha \geq 55\%$ with theoretical cost ratio and $\alpha > 85\%$ with empirical cost ratio. Our measured acceptance rate of 27.5% falls short of even the optimistic break-even threshold.

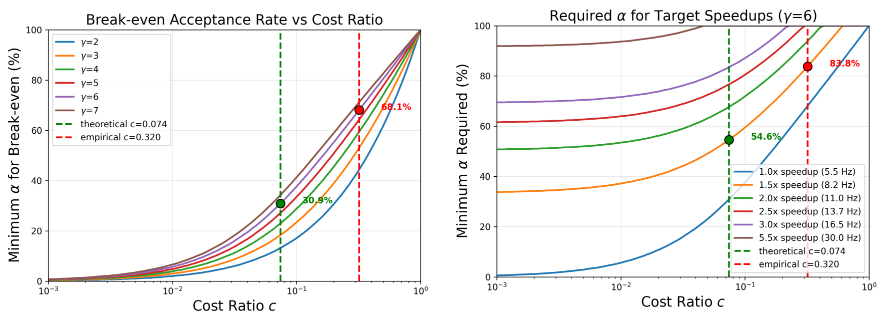

Figure 7: Left: Minimum acceptance rate required for break-even speedup vs. cost ratio for different γ. Right: Required acceptance rates for different speedup targets. The red vertical line indicates our empirical cost ratio ($c = 0.32$) and the green vertical line indicates the theoretical cost ratio ($c = 0.074$). Intersection points are marked for $\gamma = 6$ (left) and $1.5\times$ speedup (right).

4.5.2 Theory vs. Empirical

We compare theoretical predictions with empirical measurements in Table 6. As GenZ provides an optimistic upper bound by modeling compute and memory costs under roofline assumptions, it does not capture real-world system slowdowns. This gap is particularly large for MiniVLA because, as the backbone becomes smaller, overheads such as CUDA kernel launches and Python control flow become more significant relative to the total runtime. Moreover, OpenVLA and MiniVLA inference includes image preprocessing, vision encoding, and a long multimodal prefill (256 vision + $\sim$25 language tokens) followed by only 7 action tokens to predict. With such a short decode length, these fixed costs are not amortized over many output tokens like in LLMs, limiting the attainable speedup from token-level verification batching. Despite differences in absolute throughput, the theoretical and empirical analyses agree on the qualitative outcome: with the measured acceptance rate and empirical cost ratio, speculative decoding is expected to slow down. Even under the optimistic GenZ cost ratio, break-even at $\gamma = 6$ would require $\alpha \geq 31\%$, which remains above our observed operating point. Thus, the fundamental obstacle to speculative decoding for the studied autoregressive VLA models is a combination of low acceptance rates and insufficient cost ratio reduction.

Table 6: Comparison of theoretical (GenZ) vs. empirical measurements.

Metric	**GenZ Theory**	**Empirical**	**Discrepancy**
OpenVLA throughput	38.0 Hz	5.49 Hz	6.9× slower
MiniVLA throughput	415.7 Hz	17.22 Hz	24.1× slower
Decode latency ratio	13.5×	3.13×	4.3× smaller
Cost ratio c	0.074	0.32	4.3× higher
Break-even α ($\gamma = 6$)	31%	68%	+37%
Measured α	—	27.5%	$<$ break-even
Speedup ($\alpha = 27.5\%$, $\gamma = 6$)	0.87×	0.46×	1.9× slower

5 Discussion

Both the empirical ablations in Section 4 and the theoretical analysis in Section 4.5 indicate that vanilla speculative decoding does not provide speedup for our OpenVLA/MiniVLA setup, even under optimistic assumptions (perfect GPU utilization). Indeed, with exact matching, throughput drops by more than a half ($0.46\times$). Increasing the relaxation threshold r improves acceptance (Figure 4), but speedup remains below $1.0\times$ across all tested settings (Figure 5), including the artificial setting of perfect acceptance (maximal relaxation threshold $r = 255$). The theoretical speedup landscape (Figure 6) places our operating point well inside the slowdown region. The two main causes are (i) the insufficient cost ratio and (ii) the low acceptance rate.

For effective speculative decoding, a very fast draft ($c = T_{\text{draft}}/T_{\text{target}} \ll 1$) is needed. In our pairing (Table 4), OpenVLA runs at 5.49 Hz (182 ms/action) and MiniVLA at 17.22 Hz (58 ms/action), yielding $c \approx 0.32$, well above the typical cost ratios where speculative decoding typically produces substantial gains. Moreover, due to the OpenVLA/MiniVLA model architecture, each action prediction requires processing a new image and running a multimodal prefill before decoding the action tokens. These fixed costs per inference step can hardly be amortized due to the short length of the predicted sequences (7 tokens, one per action dimension). EAGLE-2, for example, reports average acceptance lengths of 4-5.5 tokens per cycle for LLMs. This brings attention to action chunking (predicting multiple actions per step), which has been explored in numerous recent works [16, 24], as a direct way to improve amortization and thus speculative decoding throughput.

To reach high acceptance rates, the draft and target models need to produce similar probability distributions. Our findings are consistent with Spec-VLA: due to the complexity of action prediction, naive speculation underperforms for VLAs without relaxed criteria [15]. In our setting, this is aggravated by the following factors:

- MiniVLA is based on Qwen2.5-0.5B while OpenVLA uses Llama-2-7B. These architectures have different tokenizers, attention mechanisms and learned representations. Draft/target alignment is a crucial next step, as proven by EAGLE's success through a draft head operating on hidden states from the same target model. This ensures a strong alignment at the feature level.
- Due to computational and time constraints, pretraining OpenVLA and MiniVLA from scratch on a shared dataset was not an option. Although both models are fine-tuned on LIBERO-90, the differences in pretraining corpora and procedures create distinct prior distributions that the fine-tuning does not fully align.
- Due to the multimodal nature of action prediction and the fact that a robot action can be successfully accomplished by following many different trajectories (e.g. reaching for an object from different angles), the draft

and target models may learn different valid policies. This contrasts to the unimodal text generation setting, where next-token prediction is more "deterministic", as there are less possibilities thanks to language being (relatively) strongly structured. We hypothesize that this higher inherent uncertainty is one of the main contributing factors to the low acceptance rates.

A substantial part of this project was spent on understanding if continuous variants of speculative decoding [19] could be applied to Flow Matching models such as SmolVLA [10]. This interest is motivated by the recent shift in the literature towards Diffusion and Flow Matching Transformer-based [11, 12] VLA models, as they show advantages in performance and efficiency [25]. Unfortunately, speculative decoding in its current form cannot be applied to this type of models, as they generate actions through iterative denoising rather than autoregressive token prediction. The seven action chunk components are generated in parallel over few denoising steps, meaning that the "speculation" would need to occur in the denoising trajectory space. As addressing this limitation would require the design of fundamentally new diffusion optimization techniques, we did not explore this direction further as it falls beyond the scope of the current project.

5.1 Insights for Future Work

Our results suggest that achieving practical speedups for autoregressive VLA models such as OpenVLA likely requires approaches that (i) substantially increase effective acceptance and (ii) reduce end-to-end fixed costs. Based on our experiments and recent literature, we highlight several promising directions:

- Training lightweight draft heads (e.g., adapting EAGLE-3 [26]) specifically for robot action speculation, rather than pairing independently trained models. Sharing the target model's features could improve acceptance while avoiding a heavy second multimodal forward pass for the draft.
- Approaches that reduce or remove sequential token-level autoregression such as parallel action decoding (predicting a full action in parallel) and action chunking (predicting multiple actions per pass) [16], or diffusion decoder heads [10, 11, 25], can consistently amortize overhead and may be better suited for short-horizon policies.
- The "Running VLAs at Real-Time Speed" paper [17] shows that CUDA graphs and kernel-level optimizations can enable real-time inference by eliminating CPU-side overhead and kernel launch costs. This suggests that aggressive system-level optimizations may be a key factor in designing very fast drafters that could make speculative decoding worthwhile for autoregressive VLAs. However, this requires that future VLA architectures adopt inference paths with more predictable tensor shapes and control flow for such optimizations to be most effective.

- Analytical tools such as GenZ, when combined with profiling, prove valuable for rapidly determining whether a speculative decoding configuration is likely to yield a speedup under ideal conditions, before investing in complex implementations. Further develpment of similar tools with a focus on multimodal LLMs would be of significant interest.

5.2 Challenges

The open-source VLA ecosystem is considerably less mature than that of LLMs. Fewer open-source autoregressive and performant VLA models are publicly available: primarily OpenVLA and variants [4, 5, 16, 24, 27], as well as other early VLA models. And most lack the model size diversity common in the LLM space. Unlike LLMs, which often come in multiple sizes (e.g., 1B, 7B, 14B, 70B), VLA models typically exist in a single configuration, making it difficult to find suitable target/draft model pairs for speculative decoding. Moreover, the field is rapidly shifting towards non-autoregressive and diffusion/flow-based policies (e.g., [10, 18, 25]), reducing the number of models for which speculative decoding is directly applicable. The only viable pairing available at the time of this project was OpenVLA and MiniVLA. Even this pairing required significant integration effort due to incompatibilities in tokenizers, inference code paths and software environments. MiniVLA's codebase also lacked comprehensive documentation, complicating efforts to fine-tune on LIBERO subsets to align it with OpenVLA. In our attempts, training loss decreased to extremely low values while evaluation success remained at 0%, suggesting issues with the default training configuration. We also discovered a critical bug that led to incorrect evaluation statistics when using the openvla-mini codebase with OpenVLA, due to incorrect preprocessing. All of these issues slowed down experiments. An attempt to contact the authors was made approximately two months before submission, but we received no response.

6 Conclusion

This project evaluated whether lossless speculative decoding can accelerate autoregressive Vision-Language-Action (VLA) inference using OpenVLA and MiniVLA as target and draft models. Empirically, speculative decoding slowed inference: at $\gamma = 6$ with exact acceptance, we measured $\alpha = 27.5\%$ and achieved $0.46\times$ throughput relative to baseline, and relaxing acceptance did not reach break-even. Theoretical modeling explains this outcome: even with an optimistic cost ratio $c = 0.074$, break-even at $\gamma = 6$ requires $\alpha \geq 31\%$, whereas our end-to-end measurements imply $c \approx 0.32$ and thus $\alpha \geq 68\%$. These constraints are amplified by the specific OpenVLA architecture, including the short 7-token decode and large multimodal prefill which limit amortization and compounding speedups. Throughout the present work, we worked around MiniVLA codebase bugs to implement a speculative decoding wrapper for OpenVLA/MiniVLA, optimized MiniVLA inference with a lightened generation function and selective `torch.compile`. Overall, our findings suggest that meaningful VLA speedups are more likely to come from tailored and more aligned draft models, a richer draft/target model selection or non-speculative architectural and/or system optimizations than from vanilla speculative decoding between independently trained VLAs.

Acknowledgments

I would like to thank Dr. Wenqi Jiang for his guidance and supervision throughout this project. I also thank the ETH Systems Group for their administrative help and for providing access to computing resources.

Generative artificial intelligence has been used to improve the grammar and readability of certain parts of this report.

References

[1] Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Andrey Kolobov, 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 Foster, Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Felipe Vieira Frujeri, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guangwen Yang, Guanzhi Wang, Hao Su, Hao-Shu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homanga Bharadhwaj, 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, Jay Vakil, Jeannette Bohg, Jeffrey 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, Muhammad Zubair Irshad, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J Joshi, Niko Suenderhauf, 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, Sean Kirmani, Sergey Levine, Shan Lin, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham Sonawani, Shubham Tulsiani, 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, Vikash Kumar, Vincent Vanhoucke, Vitor Guizilini, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiangyu Chen, Xiaolong Wang, Xinghao Zhu, Xinyang Geng, Xiyuan Liu, Xu Liangwei, Xuanlin Li, Yansong Pang, Yao Lu, Yecheng Jason Ma, Yejin Kim, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Yilin Wu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yongqiang Dou, 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, Zipeng Fu, and Zipeng Lin. Open x-embodiment: Robotic learning datasets and rt-x models, 2025. URL `https://arxiv.org/abs/2310.08864`.

[2] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL `https://arxiv.org/abs/2211.17192`.

[3] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL `https://arxiv.org/abs/2302.01318`.

[4] Suneel Belkhale and Dorsa Sadigh. Minivla: A better vla with a smaller footprint, 2024. URL `https://ai.stanford.edu/blog/minivla/`.

[5] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag 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, 2024. URL `https://arxiv.org/abs/2406.09246`.

[6] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex 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 Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar

Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong 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, 2023. URL https://arxiv.org/abs/2212.06817.

[7] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023. URL https://arxiv.org/abs/2307.15818.

[8] Siddharth Karamcheti, Suraj Nair, Ashwin Balakrishna, Percy Liang, Thomas Kollar, and Dorsa Sadigh. Prismatic vlms: Investigating the design space of visually-conditioned language models, 2024. URL https://arxiv.org/abs/2402.07865.

[9] Seungjae Lee, Yibin Wang, Haritheja Etukuru, H. Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions, 2024. URL https://arxiv.org/abs/2403.03181.

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

[11] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2023. URL https://arxiv.org/abs/2210.02747.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

[13] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty, 2025. URL https://arxiv.org/abs/2401.15077.

[14] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees, 2024. URL https://arxiv.org/abs/2406.16858.

[15] Songsheng Wang, Rucheng Yu, Zhihang Yuan, Chao Yu, Feng Gao, Yu Wang, and Derek F. Wong. Spec-vla: Speculative decoding for vision-language-action models with relaxed acceptance, 2025. URL `https://arxiv.org/abs/2507.22424`.

[16] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025. URL `https://arxiv.org/abs/2502.19645`.

[17] Yunchao Ma, Yizhuang Zhou, Yunhuan Yang, Tiancai Wang, and Haoqiang Fan. Running vlas at real-time speed, 2025. URL `https://arxiv.org/abs/2510.26742`.

[18] 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, 2026. URL `https://arxiv.org/abs/2410.24164`.

[19] Zili Wang, Robert Zhang, Kun Ding, Qi Yang, Fei Li, and Shiming Xiang. Continuous speculative decoding for autoregressive image generation, 2025. URL `https://arxiv.org/abs/2411.11925`.

[20] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning, 2023. URL `https://arxiv.org/abs/2306.03310`.

[21] Abhimanyu Bambhaniya, Ritik Raj, Geonhwa Jeong, Souvik Kundu, Sudarshan Srinivasan, Suvinay Subramanian, Midhilesh Elavazhagan, Madhu Kumar, and Tushar Krishna. Demystifying ai platform design for distributed inference of next-generation llm models, 2025. URL `https://arxiv.org/abs/2406.01698`.

[22] Karl Pertsch. rlds_dataset_builder: An example rlds dataset builder for x-embodiment dataset conversion. `https://github.com/kpertsch/rlds_dataset_builder`, 2024. GitHub repository.

[23] Perfetto contributors. Perfetto - system profiling, app tracing and trace analysis. URL `https://github.com/google/perfetto`.

[24] Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. A survey on vision-language-action models for embodied ai, 2025. URL `https://arxiv.org/abs/2405.14093`.

[25] 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, 2025. URL https://arxiv.org/abs/2409.12514.

[26] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test, 2025. URL https://arxiv.org/abs/2503.01840.

[27] Paweł Budzianowski, Wesley Maa, Matthew Freed, Jingxiang Mo, Winston Hsiao, Aaron Xie, Tomasz Młoduchowski, Viraj Tipnis, and Benjamin Bolte. Edgevla: Efficient vision-language-action models, 2025. URL https://arxiv.org/abs/2507.14049.