FLASHDRIVE:
FLASH VISION-LANGUAGE-ACTION INFERENCE FOR AUTONOMOUS DRIVING

Zekai Li[1,*] **Yihao Liang**[2,*] **Hongfei Zhang**[3] **Jian Chen**[1] **Zhijian Liu**[1]
[1]UC San Diego [2]Princeton [3]Independent Researcher
[*]Indicates equal contributions

https://z-lab.ai/projects/flashdrive

ABSTRACT

Vision-Language-Action (VLA) models promise to bring end-to-end reasoning to autonomous driving, but their computational cost is orders of magnitude too high for real-time control. The core challenge is that VLA inference is not a single bottleneck but a pipeline of four stages (visual encoding, language model prefilling, reasoning-token decoding, and flow-matching action prediction), each dominated by a different source of inefficiency: spatial redundancy in overlapping video frames, autoregressive serialization of reasoning chains, and over-computation in iterative denoising. Addressing any one stage in isolation leaves the others untouched. We propose FlashDrive, an algorithm-system co-design framework that targets all four stages simultaneously. A key insight is that each bottleneck admits a distinct, lightweight algorithmic shortcut: temporal overlap enables streaming KV-cache reuse across frames; the low entropy of domain-specific reasoning tokens makes them amenable to diffusion-based speculative decoding; and the non-uniform structure of the denoising velocity field, sharp at the endpoints but nearly constant in the middle, permits adaptive step caching that concentrates compute where it matters. Layered on top of system-level CUDA Graph compilation and kernel fusion, these techniques compound multiplicatively. Applied to Alpamayo 1.5 10B with W4A8 quantization, FlashDrive reduces end-to-end latency from 716 ms to 159 ms ($4.5\times$) with negligible accuracy loss ($\leq$0.08 m on minADE_6@6.4s and ADE@6.4s even improves), bringing VLA-driven autonomous driving to real-time viability on consumer-level GPUs.

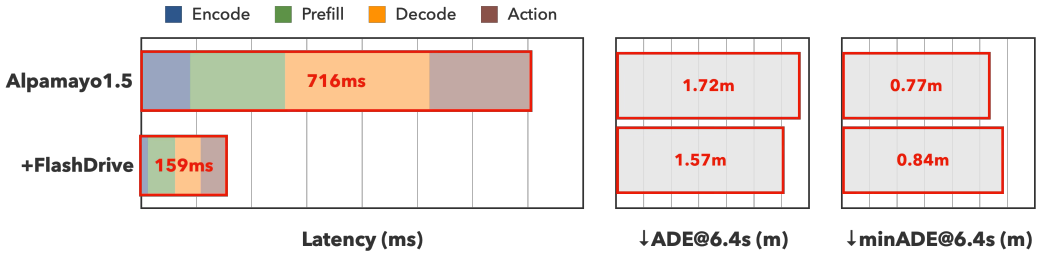


Figure 1: Reasoning VLA models for autonomous driving such as Alpamayo 1.5 exhibit a prohibitive end-to-end inference latency of 716ms on RTX PRO 6000, far exceeding real-time requirements. FlashDrive achieves a $4.5\times$ latency reduction (down to 159 ms) while incurring negligible degradation on minADE@6.4s and even improving ADE@6.4s.

1 INTRODUCTION

Autonomous driving is undergoing a paradigm shift. Modular pipelines, where perception, prediction, and planning are developed independently, are giving way to end-to-end Vision-Language-Action

(VLA) models that reason directly over raw sensor streams (Wang et al., 2025b; Zhou et al., 2025; Tian et al., 2024). By unifying visual understanding, chain-of-thought reasoning, and trajectory prediction within a single model, VLAs can handle complex, long-tail scenarios where hand-crafted interfaces between modules would fail. This unified reasoning is precisely what makes VLAs attractive for real-world autonomy.

Yet it is also what makes them prohibitively slow. The recent open-source Alpamayo 1.5-10B model (Wang et al., 2025b), for instance, requires 716 ms per frame on an NVIDIA RTX PRO 6000 GPU (Fig. 1), yielding a control frequency of just 1.4 Hz, far too slow for safe driving. Importantly, this bottleneck is not specific to one model; it is structural. Every VLA must encode high-resolution multi-view video, attend over long token sequences, autoregressively generate reasoning chains, and iteratively denoise trajectories. Each of these stages is expensive for a fundamentally different reason, and optimizing one in isolation leaves the others untouched.

To map out these bottlenecks, we decompose the VLA pipeline into four stages and profile each:

- *Encode*: the vision tower processes multi-view, multi-frame images. In a sliding-window setup, $\sim$75% of these frames have already been seen, yet the model re-encodes them from scratch.
- *Prefill*: the VLM ingests thousands of image and text tokens to populate the KV cache, repeating work that could be carried over from the previous timestep.
- *Decode*: the model autoregressively generates reasoning tokens one at a time, despite the fact that driving-domain reasoning chains are short and predictable.
- *Action*: a multi-step flow-matching module runs many denoising iterations, even though the velocity field is nearly constant through the middle of the trajectory and only changes meaningfully at the endpoints.

The key observation is that each stage harbors a distinct form of redundancy, and each admits a correspondingly distinct algorithmic shortcut. This motivates FlashDrive, an algorithm-system co-design framework. On the **system side**, we compile each pipeline stage into a CUDA Graph and fuse attention and MLP kernels, yielding a $1.56\times$ baseline speedup by eliminating launch overhead. On the **algorithm side**, we exploit the stage-specific redundancies:

- **Streaming inference** exploits the temporal overlap in driving video: we encode only the new frame and reuse the KV cache from preceding frames, cutting the encode and prefill cost by $\sim 3\times$. A lightweight streaming fine-tuning procedure adapts the action expert to the resulting distributional shift.
- **Speculative reasoning** exploits the low entropy of domain-specific reasoning tokens: we use DFlash (Chen et al., 2026), a diffusion-based parallel drafter, to generate entire candidate blocks at once. A two-layer draft model achieves an average accept length of $\sim$5.6 tokens, delivering a $4\times$ decoding speedup.
- **Adaptive-step flow matching** exploits the non-uniform structure of the denoising velocity field: velocities change sharply at the endpoints, where the trajectory departs from noise and snaps onto the data manifold, but are nearly constant through the middle. We cache the intermediate velocities and concentrate compute on the steps that matter, halving the number of function evaluations with negligible accuracy loss.

We instantiate FlashDrive on Alpamayo 1.5-10B and, together with W4A8 weight quantization, achieve a $4.5\times$ end-to-end speedup (716 ms $\rightarrow$ 159 ms, Fig. 1). Crucially, the acceleration is nearly lossless: minADE_6@6.4s degrades by only 0.07 m, while ADE@6.4s actually improves by 0.15 m, likely because streaming fine-tuning acts as a regularizer that reduces prediction variance. While demonstrated on Alpamayo 1.5, the techniques in FlashDrive are general and applicable to any reasoning VLA for autonomous driving with the same encode–prefill–decode–action structure.

2 Related Work

VLA Models for Autonomous Driving. End-to-end autonomous driving has evolved rapidly from perception-only models to Vision-Language-Action (VLA) systems that close the loop from sensors to control. Early work unifies perception and planning through textual waypoints or meta-actions (Cui

et al., 2025; Sima et al., 2024; Hwang et al., 2025), while more recent models attach specialized action heads that output continuous trajectories (Shao et al., 2024; Xu et al., 2024; Fu et al., 2025; Zhou et al., 2025). A recurring tension in this line of work is between the richness of deliberative reasoning and the strict latency constraints of real-time control; some systems resolve this by decoupling a slow VLM reasoner from a fast downstream planner (Tian et al., 2024; Pan et al., 2024). Alpamayo 1 (Wang et al., 2025b) avoids this decoupling by bridging chain-of-causation reasoning with flow-matching action prediction in a single model, achieving strong accuracy but at a high computational cost. Our work takes this cost as the starting point.

Efficient VLA Inference. A growing body of work aims to make VLA models deployable on edge hardware. Architectural approaches reduce the cost of individual components: linear-time attention and KV caching (Leal et al., 2024; Xu et al., 2025b) tame the quadratic prefill, state-space models replace transformers entirely (Liu et al., 2024), and parallel or diffusion-based decoders bypass autoregressive generation (Kim et al., 2025; Song et al., 2025; Black et al., 2025). Orthogonally, compression techniques shrink the model footprint, including lightweight backbones (Wen et al., 2025; Budzianowski et al., 2025), mixture-of-experts routing (Song et al., 2024), hierarchical reasoning-execution decoupling (Zhang et al., 2024), layer pruning (Zhang et al., 2026; Yue et al., 2024), extreme quantization (Kim et al., 2024; Wang et al., 2025a), and token-level optimizations such as faster tokenization (Pertsch et al., 2025), token pruning (Tan et al., 2025), and visual feature caching (Xu et al., 2025a). However, most of these methods target a single stage or a single axis of efficiency. FlashDrive is complementary: rather than redesigning the VLA architecture, it accelerates the *full* inference pipeline of an existing model through co-design across all four stages, yielding compounding gains that no single-stage optimization can achieve alone.

3 Methodology

Our central thesis is that VLA inference latency is not a single bottleneck but a cascade of four distinct ones (encode, prefill, decode, and action), each requiring a different solution. We first introduce stage-specific algorithmic techniques (§3.1–§3.4) that exploit the unique redundancy structure of each stage, then describe system-level optimizations (§3.5) that reduce execution overhead across the board. While we use Alpamayo 1 (Wang et al., 2025b) as the concrete instantiation, the techniques transfer to any VLA model with the same pipeline structure.

3.1 Encode & Prefill: Streaming Inference

Observation. In continuous driving, the VLA model processes a sliding window of temporal frames (typically 4 frames $\times$ 4 views). Because the window advances by one frame at each timestep, three out of four frames have already been encoded at the previous step. Re-encoding them from scratch wastes $\sim$75% of the visual computation and, since the prefill stage operates on these same tokens, a comparable fraction of the prefill as well. This redundancy is unique to the driving domain: unlike chat-based VLMs where each query is independent, autonomous driving VLAs process a continuous, highly overlapping sensory stream.

Streaming design. We propose streaming inference: encode only the newest frame and persist the KV cache from preceding frames. Two challenges arise. First, when the VLA model arranges image tokens in view-major order (as in Alpamayo 1.5), simply appending new tokens would break the expected layout. Instead, we insert the new frame's tokens at the last-frame position of each view and apply a streaming attention mask (Fig. 3a) that preserves cross-view causality. Second, because visual token positions shift with each new frame, the standard post-RoPE key cache becomes stale: RoPE encodes absolute position, so a token that was at position p in the previous window now needs to be treated as if it were at position $p - \Delta$. We resolve this by storing keys in a pre-RoPE state and computing rotary embeddings on the fly at the correct shifted positions. This reduces the effective sequence length by 75%, delivering a $\sim$3$\times$ encode-and-prefill speedup.

Streaming fine-tuning. The streaming KV cache is an approximation: the cached keys and values were computed under a different attention context than the current frame would produce in a full forward pass. In practice, this distributional shift degrades action accuracy by $\sim$0.3 m ADE and $\sim$0.2 m minADE (Fig. 3b). Interestingly, the shift primarily affects the action expert rather than the

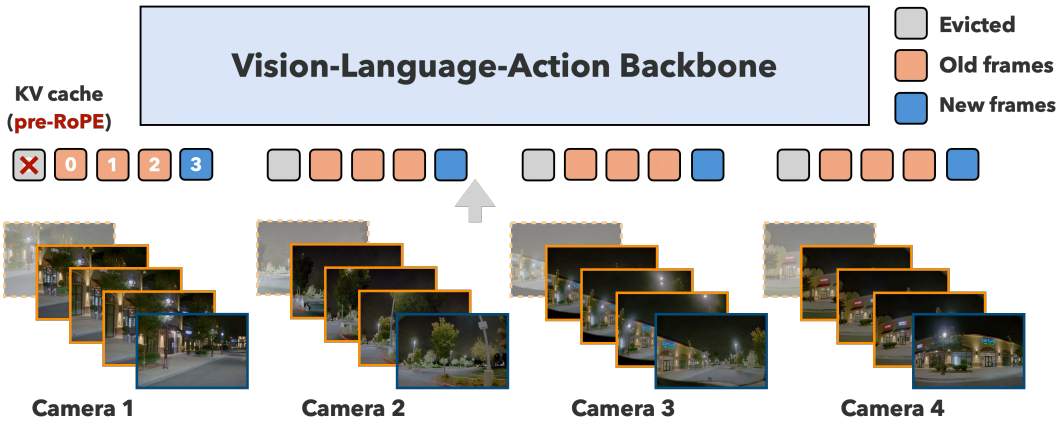
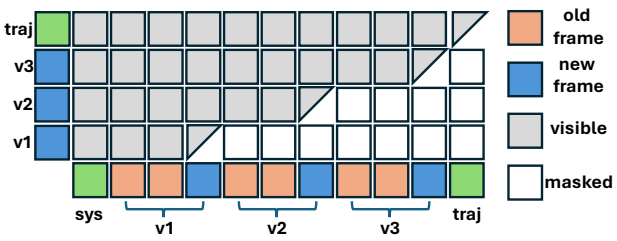
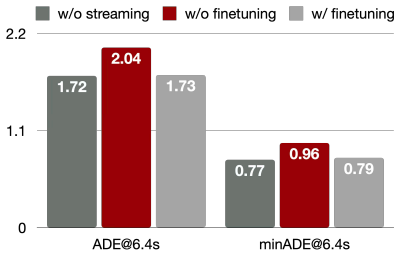


Figure 2: Streaming inference design. We respect the view-major token ordering by inserting incoming frames at the end of each camera view. To accommodate the dynamic position shift, we store pre-RoPE keys in the cache and apply RoPE on the fly.



(a) Streaming attention mask maintains causal attention between views and within each new frame.



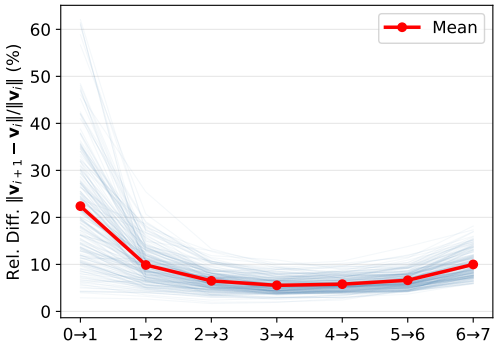
(b) Streaming finetuning effectively recovers trajectory accuracy.

language head. We attribute this to a difference in how the two components use context: reasoning tokens are generated autoregressively and attend mainly to recent tokens, making them robust to stale cache entries from older frames. The action expert, by contrast, integrates information across the *entire* KV cache through cross-attention to produce continuous trajectories, amplifying even small distributional mismatches.

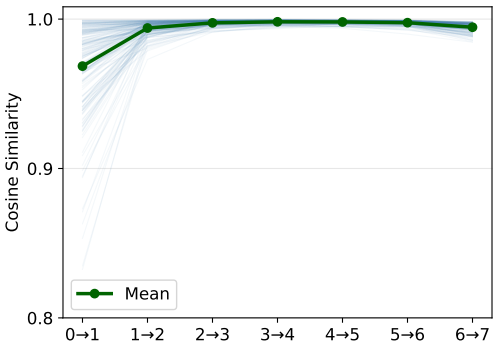
This asymmetry suggests a targeted fix: freeze the VLM backbone and fine-tune only the action expert. We adopt a rollout-based teacher-forcing scheme designed to expose the action expert to the compounding approximation errors it will encounter at deployment. For a randomly sampled sliding window of length L, the model first rolls out $L-1$ steps under the streaming mask to populate the KV cache (no gradients), then enables gradients at the final step to compute the action loss. By training on windows of varying length, the action expert learns to produce accurate trajectories even when the KV cache has accumulated multiple rounds of streaming approximation. This lightweight procedure fully recovers accuracy (1.73 m ADE and 0.79 m minADE_6 in Fig 3b).

3.2 Decode: Speculative Reasoning

Observation. Recent VLA models for AD usually incorporate reasoning to generate explicit reasoning tokens to guide trajectory prediction (Zhou et al., 2025; Wang et al., 2025b). In Alpamayo 1.5, these chain-of-thought (CoT) tokens describe the ego vehicle state, nearby obstacles, and intended maneuvers in a structured, template-like format. While this deliberative reasoning improves driving accuracy, it comes at a steep cost: in our profiling, autoregressive CoC decoding accounts for 263.8 ms, or 36.8% of the total latency, at just 62.1 tokens-per-second throughput. The question is whether this cost is intrinsic to reasoning or an artifact of token-by-token generation.



(a) The normalized relative differences between velocities from consecutive steps are high at the beginning and at the end, but low in the middle (U-shape).



(b) The cosine similarity of velocities from consecutive steps are low at the beginning and at the end, but high in the middle (inverted U-shape).

Figure 5: The ten-step flow-matching process exhibits high inter-step redundancy. We randomly sample 10 clips and 20 window inputs from each clip to plot the relative difference (a) and cosine similarity (b) between consecutive flow-matching steps.

Why driving-domain reasoning is easy to draft. We argue that it is largely the latter. Reasoning sequences are short (∼16 tokens), follow a highly structured template, and are conditioned on a rich visual context that already determines most of the content. This makes the per-token entropy substantially lower than in open-ended language generation, creating an opportunity for speculative decoding with high acceptance rates. Moreover, the structured nature of CoT means that tokens within a block are strongly correlated (e.g., a lane-change decision constrains the subsequent speed and trajectory descriptions), which favors drafters that generate entire blocks at once rather than one token at a time.

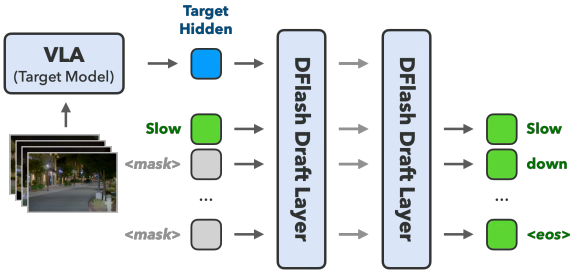


Figure 4: Speculative reasoning with DFlash draft layers. We fuse the hidden states of the last target token to the draft model. We use only two draft layers with an average accept length of 5 tokens.

Diffusion-based drafting. Motivated by this, we adopt DFlash (Chen et al., 2026), which uses a diffusion language model as a non-autoregressive parallel drafter. Unlike conventional sequential drafters, the diffusion model produces an entire block of candidates in a single forward pass, naturally capturing the intra-block correlations present in structured reasoning. We train a lightweight two-layer draft model (block size = 8) on ∼60k clips from the NVIDIA Autonomous Vehicle Dataset (NVIDIA, 2025) and fuse the hidden state of only the last eight target-model tokens into the draft layer's KV cache. The single-token fusion is sufficient because the last token already encodes the full driving context needed to predict the next reasoning block. This configuration achieves an average accept length of 5.6 tokens, reducing decoding latency to 61.2 ms, a 4.3× speedup.

3.3 Action: Adaptive-Step Flow Matching

VLA models must bridge the gap between language-level reasoning and continuous vehicle control. This is typically accomplished by a flow-matching head that converts the VLM's hidden representations into trajectory waypoints through iterative denoising. The standard configuration in Alpamayo 1 uses 10 steps, but this comes at a steep cost: each step requires a full forward pass through the action network, making the action stage one of the dominant latency contributors.

Why uniform step reduction fails. The naïve remedy is to simply reduce the number of uniformly spaced steps. While this does speed up inference, it treats all regions of the denoising trajectory as equally important, an assumption that turns out to be wrong. To see why, we profile the velocity field v_t across the 10-step trajectory. Fig. 5 reveals a striking U-shaped pattern in both the normalized relative differences between consecutive velocities (Fig. 5a) and their cosine similarities (Fig. 5b). The velocity changes sharply at the first and last steps, where the trajectory departs from the noise prior and converges onto the data manifold, but is nearly constant through the middle, where the ODE flow traverses a smooth, low-curvature region of the learned vector field.

Insight. This non-uniformity has a clear physical interpretation: the early steps establish the coarse trajectory structure (lane choice, turn direction), the final steps snap the prediction onto the manifold of physically plausible trajectories (satisfying kinematic constraints and road geometry), and the intermediate steps perform only minor refinements to an already well-determined path. The endpoints carry the signal; the middle carries the inertia.

Adaptive caching. We exploit this structure by caching the velocity at the middle steps and reusing it in lieu of recomputation. Concretely, after evaluating the first few and last few steps with fresh network calls, we replace the intermediate evaluations with the cached velocity from the preceding step. This effectively halves the number of function evaluations while concentrating compute on the steps that shape the trajectory the most. As shown in Table 1, this adaptive strategy achieves near-lossless accuracy: minADE_6 degrades by only $\sim$0.05 m, while ADE actually improves, suggesting that skipping redundant mid-trajectory refinements may even reduce accumulated numerical error from the ODE solver.

3.4 Quantization

Large VLA models often exceed the memory capacity of consumer-level GPUs; for example, Alpamayo 1-10B requires $\sim$31.6 GB in FP16 to generate 6 trajectory samples, more than many consumer-level devices can accommodate. Quantization compresses model weights and activations to lower precision, trading numerical headroom for speed. But it presents a choice. Standard methods like AWQ quantize only the weights to 4-bit (W4A16): this helps memory-bound decoding by shrinking the data the GPU must load per token, but leaves the compute-bound prefill stage untouched. For a chatbot LLM where decoding dominates, that trade-off is acceptable. For a VLA model with thousands of vision tokens in every prompt, prefill is too expensive to ignore.

W4A8 quantization targets both regimes: 4-bit weights cut memory bandwidth for decoding, while 8-bit activations unlock faster INT8 matrix multiplies for the compute-heavy prefill. We apply ParoQuant (Liang et al., 2026) to quantize model weights to 4-bit, then implement inference with 8-bit activation via Marlin kernels (Frantar et al., 2024). This effectively cuts the memory footprint down to $\sim$18.3 GB while further speedup the overall inference by 14%.

3.5 System Optimizations

The algorithmic techniques above reduce the amount of computation; system-level optimizations reduce the cost of executing what remains. Unlike standard LLM serving, where the workload is dominated by a single homogeneous decode loop, VLA inference chains together four heterogeneous stages (vision encoder, language prefill, autoregressive decode, flow-matching action), each with its own kernel mix. This heterogeneity amplifies the CPU-side dispatch cost: the pipeline involves hundreds of small kernel launches per forward pass, and at the low arithmetic intensities typical of single-batch decoding, launch overhead becomes a significant fraction of wall-clock time.

CUDA Graph. We compile each pipeline stage into a CUDA Graph, which records the full kernel sequence and replays it in a single GPU-side launch. This is particularly impactful for the decode stage, where the GPU would otherwise idle between every token while waiting for the CPU to schedule the next kernel.

Kernel Fusion. Many VLM backbones dispatch the Q, K, and V projections as three separate kernels, and likewise for the gate and up projections in the MLP, totaling six launches where two

	Latency (with 1 Trajectory) (ms)					Trajectory Error (m)	
	Encode	Prefill	Decode	Action	Total	ADE↓	minADE_6 ↓
Alpamayo 1.5	88.0	177.2	263.8 62.1	187.4	716.4	1.721 ± 0.061	0.767 ± 0.013
+ System Optimizations	43.2	192.4	167.1 91.6	112.6	515.3	1.719	0.777
+ Streaming Inference	12.5	62.1	171.3 89.3	115.6	361.5	1.733	0.792
+ Speculative Reasoning	44.0	197.5	61.2 241.8	114.4	417.1	1.650	0.754
+ Adaptive-Step Flow Matching	43.9	194.6	169.6 92.0	45.9	454.0	1.566	0.818
+ All above	12.4	61.5	60.7 242.1	46.6	181.2	1.561	0.855
+ Quantization	12.5	52.5	48.2 305.0	46.2	159.4	1.568	0.844

Table 1: FlashDrive achieves 4.5× overall speedup with one trajectory on NVIDIA RTX PRO 6000. The speedup breakdown demonstrates the effectiveness of each component. The throughput next to decoding time denotes the throughput (token/s) for that setting.

suffice. We fuse each group and additionally compile under the max-autotune mode, which merges consecutive element-wise and reduction operations and auto-selects the fastest implementation. Together, CUDA Graphs and kernel fusion provide a 1.39× speedup without changing the model's computation.

4 Evaluation

4.1 Setup

Model. We evaluate FlashDrive on Alpamayo 1.5-10B, the recent open-source state-of-the-art VLA model for autonomous driving. While we use Alpamayo 1.5-10B as the primary testbed, FlashDrive's techniques are applicable to any VLA model sharing the encode–prefill–decode–action pipeline.

Dataset. We use the open-source NVIDIA Autonomous Vehicle Dataset (NVIDIA, 2025) for both training and evaluation. For streaming fine-tuning, we sample 4k clips across different chunks and randomly select starting points and rollout lengths within each clip, yielding ∼600k training samples (at most 150 per clip). For evaluation, we sample 100 clips and extract sliding-window inputs at 10 FPS, producing 120 windows per clip and 12k evaluation samples in total. For diffusion draft model training, we randomly sample one window input from 60k clips from the dataset.

Metrics. Following the official protocol, we report minADE_6@6.4s, the minimum L2 distance between the ground-truth trajectory and each of six predicted trajectories over the next 6.4 s. We additionally report ADE@6.4s based on a single inference sample.

4.2 Efficiency Results

End-to-end speedup. As shown in Table 1, FlashDrive achieves a 4.5× end-to-end speedup, reducing latency to 159.4 ms (6.3 FPS). This crosses a critical threshold: at over 6 FPS, the control loop is fast enough for practical deployment in urban driving scenarios.

System-level gains. CUDA Graphs and kernel fusion alone reduce end-to-end latency by 28.1% (1.39×). The gains are not uniform across stages: encode, decode, and action benefit the most because their many small kernels make them disproportionately sensitive to launch overhead. Prefill, dominated by a single long matrix multiplication, sees no improvement. This asymmetry underscores why system and algorithm optimizations are complementary: system-level changes address overhead, while algorithmic changes address the computation itself.

Algorithm-level gains. Table 1 ablates each technique on top of the system-optimized baseline. Streaming inference accelerates encoding by 3.5× and prefilling by 3.1×. DFlash boosts decoding throughput by 2.6×, directly targeting the single largest latency contributor. Adaptive-step flow matching reduces function evaluations from 10 to 4, yielding a 2.5× action-stage speedup. Jointly, the algorithmic gains save 334.1 ms, roughly 1.7× the system-level savings, confirming that eliminating redundant computation matters more than eliminating overhead when both are present. W4A8

Figure 6: **Qualitative comparison of trajectories and CoC results.** ***(Left)*** shows the action visualization, ***(Right)*** illustrates the corresponding CoT comparisons (Baseline *v.s.* **FlashDrive (F.D.)**). Two common autonomous driving scenarios are selected for illustration: *Longitudinal Straight Driving (Row 1)*, and *Oncoming Vehicle Encounter (Row 2)*.

quantization contributes a further 21.8 ms reduction through lower memory bandwidth pressure. The speedup on both prefill and decode further validate our design: the 8-bit activation quantization is important for VLA models due to the much more restrict efficiency requirement.

Cross-device deployment. Beyond the RTX PRO 6000, we benchmark FlashDrive on the Jetson Thor, RTX 3090, RTX 4090, and RTX 5090 (Table 2). With a single trajectory sample, FlashDrive achieves a consistent speeup. ranging from $4.0\times$ to $5.7\times$. Notably, FlashDrive unlocks the deployment of VLA models on edge devices such as the Jetson Thor, delivering a $4.0\times$ speedup.

Model		Jetson Thor	RTX 3090	RTX 4090	RTX 5090	RTX PRO 6000
1 sample	Alpamayo1.5	3770.3	1787.5	1187.2	986.2	716.4
	FlashDrive	943.6 (4.0×)	362.9 (4.9×)	208.6 (5.7×)	196.0 (5.0×)	159.4 (4.5×)

Table 2: FlashDrive demonstrates robust acceleration on multiple consumer-level GPU devices, achieving maximally $5.7\times$ speedup with one trajectory sample on RTX 4090.

4.3 Accuracy Results

Table 1 compares the trajectory accuracy of FlashDrive against the baseline. For minADE_6@6.4s (six-sample), FlashDrive incurs a degradation of only $\sim$0.07 m, roughly 1.1 cm per second of prediction horizon, well within the safety margin of a standard lane. Surprisingly, single-sample ADE@6.4s actually *improves* by $\sim$0.15 m. We hypothesize that this is because streaming fine-tuning, by training the action expert to be robust to approximate KV caches, acts as a form of regularization that reduces prediction variance. Similarly, our speculative reasoning also drags the model towards the optimal trajectory direction by reducing the entropy of CoT sequences while maintaining accuracy. Fig. 6 corroborates this qualitatively: FlashDrive generates the same or highly similar CoC reasoning tokens and produces trajectories closely aligned with the ground truth, confirming that the $4.5\times$ acceleration comes at essentially no cost to driving quality.

5 Conclusion

We presented FlashDrive, an algorithm-system co-design framework that makes VLA-based autonomous driving run in real time on consumer-level GPUs. The key lesson from our work is that VLA inference is not a monolithic bottleneck but a cascade of four stages, each dominated by a different form of redundancy: temporal overlap in vision, serialization in reasoning, and over-iteration in

denoising. By matching each bottleneck to a lightweight algorithmic shortcut (streaming, speculative decoding, adaptive step caching) and layering them on top of system-level compilation and fusion, the speedups compound to $4.5\times$ with negligible accuracy loss. We believe this "profile-then-exploit" methodology extends beyond autonomous driving to any VLA deployment where latency is the binding constraint.

ACKNOWLEDGMENT

We gratefully acknowledge Yotta Labs for providing the compute resources supporting this work.

REFERENCES

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A Vision-Language-Action Model with Open-World Generalization. In *Conference on Robot Learning (CoRL)*, 2025. 3

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. *arXiv preprint arXiv:2507.14049*, 2025. 3

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding. *arXiv preprint arXiv:2602.06036*, 2026. 2, 5

Erfei Cui, Wenhai Wang, Zhiqi Li, Jiangwei Xie, Haoming Zou, Hanming Deng, Gen Luo, Lewei Lu, Xizhou Zhu, and Jifeng Dai. DriveMLM: Aligning Multi-Modal Large Language Models with Behavioral Planning States for Autonomous Driving. *Visual Intelligence*, 2025. 2

Elias Frantar, Roberto L Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. Marlin: Mixed-precision auto-regressive parallel inference on large language models. *arXiv preprint arXiv:2408.11743*, 2024. 6

Haoyu Fu, Diankun Zhang, Zongchuang Zhao, Jianfeng Cui, Dingkang Liang, Chong Zhang, Dingyuan Zhang, Hongwei Xie, Bing Wang, and Xiang Bai. ORION: A Holistic End-to-End Autonomous Driving Framework by Vision-Language Instructed Action Generation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025. 3

Jyh-Jing Hwang, Runsheng Xu, Hubert Lin, Wei-Chih Hung, Jingwei Ji, Kristy Choi, Di Huang, Tong He, Paul Covington, Benjamin Sapp, Yin Zhou, James Guo, Dragomir Anguelov, and Mingxing Tan. EMMA: End-to-End Multimodal Model for Autonomous Driving. *Transactions on Machine Learning Research (TMLR)*, 2025. 3

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. In *Conference on Robot Learning (CoRL)*, 2024. 3

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success. In *Robotics: Science and Systems (RSS)*, 2025. 3

Isabel Leal, Krzysztof Choromanski, Deepali Jain, Kumar Avinava Dubey, Jake Varley, Michael S. Ryoo, Yao Lu, Frederick Liu, Vikas Sindhwani, Quan Ho Vuong, Tamás Sarlós, Kenneth Oslund, Karol Hausman, and Kanishka Rao. SARA-RT: Scaling Up Robotics Transformers with Self-Adaptive Robust Attention. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024. 3

Yesheng Liang, Haisheng Chen, Zihan Zhang, Song Han, and Zhijian Liu. ParoQuant: Pairwise Rotation Quantization for Efficient Reasoning LLM Inference. In *International Conference on Learning Representations (ICLR)*, 2026. 6

Jiaming Liu, Mengzhen Liu, Zhenyu Wang, Pengju An, Xiaoqi Li, Kaichen Zhou, Senqiao Yang, Renrui Zhang, Yandong Guo, and Shanghang Zhang. RoboMamba: Efficient Vision-Language-Action Model for Robotic Reasoning and Manipulation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. 3

NVIDIA. PhysicalAI Autonomous Vehicles Dataset, 2025. URL https://huggingface.co/datasets/nvidia/PhysicalAI-Autonomous-Vehicles. 5, 7

Chenbin Pan, Burhaneddin Yaman, Tommaso Nesti, Abhirup Mallik, Alessandro G Allievi, Senem Velipasalar, and Liu Ren. VLP: Vision Language Planning for Autonomous Driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 3

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. FAST: Efficient Action Tokenization for Vision-Language-Action Models. In *Robotics: Science and Systems (RSS)*, 2025. 3

Hao Shao, Yuxuan Hu, Letian Wang, Steven L. Waslander, Yu Liu, and Hongsheng Li. LMDrive: Closed-Loop End-to-End Driving with Large Language Models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 3

Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Jens Beißwenger, Ping Luo, Andreas Geiger, and Hongyang Li. DriveLM: Driving with Graph Visual Question Answering. In *European Conference on Computer Vision (ECCV)*, 2024. 3

Wenxuan Song, Han Zhao, Pengxiang Ding, Can Cui, Shangke Lyu, Yaning Fan, and Donglin Wang. GeRM: A Generalist Robotic Model with Mixture-of-Experts for Quadruped Robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024. 3

Wenxuan Song, Jiayi Chen, Pengxiang Ding, Han Zhao, Wei Zhao, Zhide Zhong, Zongyuan Ge, Zhijun Li, Donglin Wang, Jun Ma, Lujia Wang, and Haoang Li. PD-VLA: Accelerating Vision-Language-Action Model Integrated with Action Chunking via Parallel Decoding. *arXiv preprint arXiv:2503.02310*, 2025. 3

Xudong Tan, Yaoxin Yang, Peng Ye, Jialin Zheng, Bizhe Bai, Xinyi Wang, Jia Hao, and Tao Chen. Think Twice, Act Once: Token-Aware Compression and Action Reuse for Efficient Inference in Vision-Language-Action Models. *arXiv preprint arXiv:2505.21200*, 2025. 3

Xiaoyu Tian, Junru Gu, Bailin Li, Yicheng Liu, Yang Wang, Zhiyong Zhao, Kun Zhan, Peng Jia, Xianpeng Lang, and Hang Zhao. DriveVLM: The Convergence of Autonomous Driving and Large Vision-Language Models. In *Conference on Robot Learning (CoRL)*, 2024. 2, 3

Hongyu Wang, Chuyan Xiong, Ruiping Wang, and Xilin Chen. BitVLA: 1-Bit Vision-Language-Action Models for Robotics Manipulation. *arXiv preprint arXiv:2506.07530*, 2025a. 3

Yan Wang, Wenjie Luo, Junjie Bai, Yulong Cao, Tong Che, Ke Chen, Yuxiao Chen, Jenna Diamond, Yifan Ding, Wenhao Ding, Liang Feng, Greg Heinrich, Jack Huang, Peter Karkus, Boyi Li, Pinyi Li, Tsung-Yi Lin, Dongran Liu, Ming-Yu Liu, Langechuan Liu, Zhijian Liu, Jason Lu, Yunxiang Mao, Pavlo Molchanov, Lindsey Pavao, Zhenghao Peng, Mike Ranzinger, Ed Schmerling, Shida Shen, Yunfei Shi, Sarah Tariq, Ran Tian, Tilman Wekel, Xinshuo Weng, Tianjun Xiao, Eric Yang, Xiaodong Yang, Yurong You, Xiaohui Zeng, Wenyuan Zhang, Boris Ivanovic, and Marco Pavone. Alpamayo-R1: Bridging Reasoning and Action Prediction for Generalizable Autonomous Driving in the Long Tail. *arXiv preprint arXiv:2511.00088*, 2025b. 2, 3, 4

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. *IEEE Robotics and Automation Letters*, 2025. 3

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. VLA-Cache: Efficient Vision-Language-Action Manipulation via Adaptive Token Caching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025a. 3

Wanshun Xu, Long Zhuang, and Lianlei Shan. KV-Efficient VLA: A Method to Speed Up Vision Language Models with RNN-Gated Chunked KV Cache. *arXiv preprint arXiv:2509.21354*, 2025b. 3

Zhenhua Xu, Yujia Zhang, Enze Xie, Zhen Zhao, Yong Guo, Kwan-Yee K. Wong, Zhenguo Li, and Hengshuang Zhao. DriveGPT4: Interpretable End-to-End Autonomous Driving via Large Language Model. *IEEE Robotics and Automation Letters*, 2024. 3

Yang Yue, Yulin Wang, Bingyi Kang, Yizeng Han, Shenzhi Wang, Shiji Song, Jiashi Feng, and Gao Huang. DeeR-VLA: Dynamic Inference of Multimodal Large Language Models for Efficient Robot Execution. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. 3

Jianke Zhang, Yanjiang Guo, Xiaoyu Chen, Yen-Jen Wang, Yucheng Hu, Chengming Shi, and Jianyu Chen. HiRT: Enhancing Robotic Control with Hierarchical Robot Transformers. In *Conference on Robot Learning (CoRL)*, 2024. 3

Rongyu Zhang, Menghang Dong, Yuan Zhang, Liang Heng, Xiaowei Chi, Gaole Dai, Li Du, Yuan Du, and Shanghang Zhang. MoLe-VLA: Dynamic Layer-Skipping Vision Language Action Model via Mixture-of-Layers for Efficient Robot Manipulation. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2026. 3

Zewei Zhou, Tianhui Cai, Seth Z. Zhao, Yun Zhang, Zhiyu Huang, Bolei Zhou, and Jiaqi Ma. AutoVLA: A Vision-Language-Action Model for End-to-End Autonomous Driving with Adaptive Reasoning and Reinforcement Fine-Tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. 2, 3, 4