KerColle: Unlocking Fine-Grained GPU Concurrency in Vision-Language-Action Models

Anna Li
University of Toronto

Christina Giannoula
Max Planck Institute for Software Systems (MPI-SWS)

Nandita Vijaykumar
University of Toronto

Abstract

Vision-Language-Action (VLA) models have emerged as foundational models for next-generation robotics. High VLA inference throughput is critical for meeting the control-rate requirements of robots. VLA models comprise two phases, a vision-language model (VLM) and an action head, that can be decoupled and executed asynchronously and concurrently across independent robot requests. Through a detailed characterization of four state-of-the-art VLAs, we observe that GPUs are severely underutilized in VLA inference as the Cooperative Thread Array (CTA) scheduler of GPU is unable to fully overlap the two independent phases of VLA execution. We identify that this inefficiency is caused by head-of-line blocking in the hardware thread-block dispatcher which causes the action head phase kernels to be stalled behind large VLM kernels despite idle resources being available.

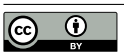
We demonstrate that prior scheduling frameworks do not address the challenges posed by VLA concurrency. First, the independent phases across different robot requests each comprise numerous kernels, and at any given time, there are different combinations of kernels that are executed in parallel. This makes static or ahead-of-time scheduling policies largely ineffective. Second, many of the action-head operators are short-running kernels and there are numerous such kernels. This leaves no headroom for online profiling or preemption-based mechanisms. To address these challenges, we present KerColle, a lightweight GPU scheduling framework that leverages online Streaming Multiprocessor (SM) utilization and individual kernel resource requirements to intelligently and dynamically co-schedule kernels to efficiently overlap the two phases of execution by (1) mitigating head-of-line blocking, and (2) co-scheduling kernels with complementary resource requirements. We demonstrate in simulation, across two GPU architectures, for 4 state-of-the-art VLA models, that KerColle delivers throughput gains of up to 28%, with an average gain of 20.5%.

CCS Concepts

• **Computer systems organization** → **Multicore architectures**.

Keywords

Vision-language-action models, GPU scheduling, Concurrent kernel execution



This work is licensed under a Creative Commons Attribution 4.0 International License.
PACT '26, Chicago, IL, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2915-7/2026/10
https://doi.org/10.1145/3838684.3846872

ACM Reference Format:
Anna Li, Christina Giannoula, and Nandita Vijaykumar. 2026. KerColle: Unlocking Fine-Grained GPU Concurrency in Vision-Language-Action Models. In *International Conference on Parallel Architectures and Compilation Techniques (PACT '26), October 19-22, 2026, Chicago, IL, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3838684.3846872

1 Introduction

Embodied systems such as robotic arms, humanoids, and autonomous vehicles are increasingly used in real-world environments; however, they are only useful when they can operate beyond controlled lab environments and handle the variability of real-world settings. The recent breakthroughs in foundational Vision-Language Models (VLMs) [8, 12, 35] tackle this constraint: VLMs contain broad visual and semantic priors from internet-scale image-text data, encoding world knowledge beyond what robot datasets cover. Vision-Language-Action (VLA) models extend the VLM with an action head fine-tuned on robot trajectories. VLAs take as input a video stream and language instructions, and translate latent representations into end-to-end motor commands [32, 41, 47, 61]. By inheriting the VLM's knowledge, VLAs generalize to objects, scenes, and instructions beyond their robot training distribution, and have therefore become the foundation for next-generation robotics.

VLA models translate natural-language instructions and a stream of frames into low-level motor actions that robots execute. This is achieved through a two-phase inference process: a VLM phase encodes the inputs into an intermediate representation, which an action expert then decodes into executable motor commands as output. While robotic platforms typically require control frequencies near or above 50 Hz [42, 79, 96], VLA inference latencies may not meet these requirements on edge devices and single GPUs [33, 41, 104]. High VLA inference throughput is critical since the action generation rate affects how quickly the robot can process new observations and react to dynamic environments [101]. Given limited inference latency and throughput, VLA models have adopted action chunking, where the VLA emits a longer sequence of actions for each inference process. However, during this window, the robot cannot react to new observations, producing abrupt motions [15, 49].

Recent VLA architectures decouple VLM execution from the action head [32, 61] so that the two phases can be pipelined across frames. This is enabled by two techniques: predicting a short sequence of future actions from a single VLM output rather than only the next action [15, 57], and running the two phases on separate execution streams [19, 112]. Together, these allow the action head for frame N to execute concurrently with the VLM for frame $N+1$, pipelining the two-phase execution [79, 112].

We conduct a comprehensive characterization of four production VLA models on modern GPU architectures and make three key observations. First, VLAs significantly underutilize the GPU

during inference: median Streaming Multiprocessor (SM) throughput across all four models ranges from only 18% to 30% of peak, and even at its busiest, the action phase leaves Tensor Cores below 3% utilized. Second, the two phases occupy different regions of the roofline: vision encoding and VLM prefill have high arithmetic intensity and average 37.4% Tensor Core utilization due to large GEMMs. In contrast, the action expert operates with a batch size of 1 and contains many low-intensity kernels. This creates an opportunity to overlap Tensor-Core-dominant compute-bound kernels with CUDA-Core execution of bandwidth-bound kernels. Third, concurrently executing the VLM and action phases causes near-zero slowdown for the VLM phase. However, pipelining does not translate into throughput gains. Naively dispatching the two phases on separate CUDA streams yields a counterintuitive result: the action head experiences significant slowdown, causing action chunks to be emitted at roughly half the rate of the action head running alone, despite negligible interference from the VLM phase and underutilized memory bandwidth.

We observe that this inefficiency arises from the hardware thread-block (CTA) scheduler's head-of-line blocking behavior [26, 28, 60, 81]. Because the VLM launches massive grids that exceed the GPU's spatial capacity, the dispatcher saturates all SMs almost exclusively with VLM thread blocks. Despite ample headroom in CUDA Cores on every SM throughout the VLM's execution, the hardware scheduler serializes the two phases, leaving these execution units idle.

Resolving this inefficiency and underutilization presents two challenges specific to VLA workloads:

Challenge 1: Non-Deterministic Kernel Co-Residency. The VLM kernel co-resident with each action-head kernel shifts at microsecond granularity. Because the two streams progress at independent rates, the VLM kernel executing when a given action-head kernel is dispatched varies across frames.

Challenge 2: Microsecond Kernel Granularity. The action head is dominated by hundreds of micro-kernels with a median duration of a few microseconds, even after optimizations and kernel fusion by backends such as TorchInductor (via `torch.compile`).

Together, these challenges render existing scheduling approaches ineffective for VLAs. Preemption-based hardware schedulers such as Maestro [72] and SMK [94] incur context-switching overheads that eclipse micro-kernel execution times. Online-profiling schedulers such as Warped-Slicer [102] require sampling windows that exceed the micro-kernel duration. Software-based approaches [26, 106, 107, 109] require source-code access, making them incompatible with the closed-source vendor libraries pervasive in VLAs. They also demand re-compilation whenever the action head or kernel implementations change. Runtime interposition layers [110] depend on specific launch sequences and cannot react in time.

Our key insight is that the information needed to co-schedule kernels from both phases is *static per kernel*: a kernel's utilization of Tensor Cores, ALU pipelines, and memory bandwidth can be measured once offline. The thread-block dispatcher can then dispatch kernels with *different* resource utilization to the same SM. This approach requires no kernel modification and is insensitive to kernel duration, making it robust to micro-kernel granularity.

We propose KerColle, a GPU scheduling framework that extends the thread-block dispatch path in three ways to support efficient concurrent execution of multiple streams. First, we extend the kernel launch descriptor with three 4-bit fields ($\tau_{TC}, \tau_{ALU}, \tau_{MEM}$), which encode the fraction of peak Tensor Core throughput, ALU throughput, and DRAM bandwidth that a kernel's CTAs consume. These tags are obtained via a one-time offline profiling pass using vendor counters [66] and injected at launch time by a lightweight runtime wrapper [67]. Second, we augment the work distributor's per-SM resource accounting with three 8-bit utilization counters (U_{TC}, U_{ALU}, U_{MEM}) that are updated on the GPU's CTA admission path and completion using the kernel's pre-profiled utilization values, leveraging the dispatcher's de-allocation signals. Third, we replace the dispatcher's greedy admission policy with a two-phase admission process that commits at most one CTA per admission decision. Phase 1 is responsible for admitting at least one CTA from every active kernel with pending work to prevent micro-kernel starvation behind monolithic GEMMs, and Phase 2 fills the remaining SM capacity by selecting the kernel whose intensity vector best aligns with the SM's headroom. If there is only one pending kernel, then that kernel will simply be selected.

We implement KerColle on Accel-Sim [38] configured for the RTX 4090 and RTX 5090, using NVBit traces of four representative VLAs: CogACT [47], $\pi_{0.5}$ [32], Octo [84], and GraspVLA [27]. We demonstrate that KerColle delivers throughput gains of up to 28%, with an average gain of 20.5% on both RTX 4090 and RTX 5090.

This paper makes the following contributions:

- We perform a detailed analysis of GPU utilization on state-of-the-art Vision-Language-Action (VLA) models and find that head-of-line blocking in the CTA dispatcher significantly limits execution concurrency of the two phases of VLA inference.
- We propose a lightweight scheduling framework that modifies the GPU hardware scheduler with a new admission algorithm and per-SM state-registers to address this gap and demonstrate its effectiveness on four state-of-the-art VLA models.
- We demonstrate that across 4 state-of-the-art VLA models, KerColle delivers throughput gains of up to 28%, with an average gain of 20.5%.

2 Background and Motivation

2.1 Vision-Language-Action Models

Traditional robots excel at repetitive tasks in structured settings but struggle in unstructured, real-world environments [32, 36]. The Vision-Language-Action (VLA) model is an emerging alternative to traditional robotics. A VLA model couples a pretrained Vision-Language Model (VLM) [8, 12, 35, 85] with a dedicated action head fine-tuned on domain-specific robotic datasets [25, 39, 90]. The VLM acts as a visual-semantic encoder, leveraging internet-scale information to process observations and natural language instructions as inputs. The action head is trained on robot-trajectory datasets and maps the high-level representations generated by the VLM into low-level, continuous motor commands, such as 7-degree-of-freedom (7-DoF) end-effector controls, for the robot to execute. VLA models are an emerging and promising technique for embodied AI [36, 105]. However, because action generation relies on computationally expensive VLMs, VLA inference latency currently fails to meet the high-frequency closed-loop requirements of modern robotics. Hence, improving the inference efficiency of VLA models has become a critical challenge in modern robotics.

Figure 1 illustrates a representative VLA architecture. In Phase 1, the model takes as input a frame and a textual instruction, propagating them through a VLM to generate an intermediate representation. In Phase 2, this representation is passed to the action head to predict a series of future actions, known as an *action chunk*, to be executed consecutively by the robotic arm. The action head produces these robotic actions through a variety of mechanisms, including autoregressive token generation [16, 41, 42, 73, 74, 93], diffusion action heads [10, 47, 53, 55, 84], flow-matching action heads [18, 32, 61, 79], and most recently, discrete diffusion action decoders [50, 56, 95].

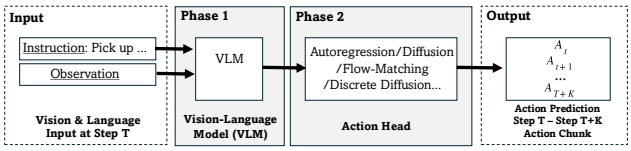


Figure 1: Representative VLA architecture decomposed into two phases.

Early VLA models relied on autoregressive token prediction [16, 17], with OpenVLA [41] remaining the dominant open-source baseline. Subsequent works replaced discrete token prediction with continuous-action generation. Diffusion Policy [10] laid the foundation for generating robotic actions via a denoising process, and was adopted in Octo [84], RDT-1B [55], and CogACT [47]. Recently, flow matching has emerged as the dominant action-head architecture. Pioneered by $\pi 0$ [14], flow-matching now serves as the action head driving both industrial models ($\pi 0.5$ [32], GR00T N1/N1.5 [61], GR-3 [18], Gemini Robotics 1.5 [83]) and open-source models (SmolVLA [79], AsyncVLA [34]). Emerging in late 2025, discrete diffusion as an action-head paradigm takes two distinct forms. One branch eliminates the traditional action head in favor of a monolithic, end-to-end transformer [50, 56], while the other retains the action-head abstraction but uses discrete denoising instead of continuous decoding [20, 95, 97].

Despite the rapid emergence of various VLA architectures, flow matching remains the dominant paradigm. We therefore focus on flow-matching-based VLAs in this paper, along with diffusion-head VLAs, which share the same two-phase structure and resource-utilization profile. We will focus on and evaluate four representative open-source models: **CogACT** [47], a diffusion-head model with a 300M DiT action head; **Octo** [84], a lightweight diffusion-head model with a compact 3M action head; π**0.5** [32], an industrial flow-matching model representative of the frontier deployment regime; and **GraspVLA** [27], a flow-matching model with a Chain-of-Thought perception-then-action pipeline. Together, these models span diffusion action heads at two parameter scales [47, 84] and flow-matching action heads from both industry [32] and academia [27]. All four follow the common two-phase structure of Figure 1: phase 1 (VLM phase) encodes images and language and produces an intermediate state consumed by phase 2 (action-head phase), which generates motor commands. There are slight differences in how phase 1's output is encoded: as readout-token embeddings [84], cognition features [47], KV cache [32] or generated token sequences [27].

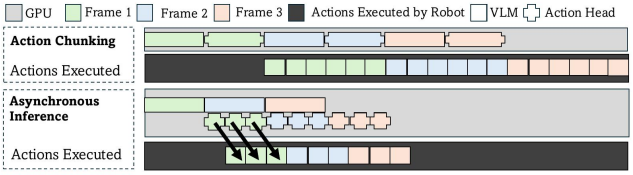


Figure 2: Execution timelines for action chunking and asynchronous inference.

To accelerate VLA inference for real-time robotic control [103], two complementary techniques are commonly adopted. First, action chunking [27, 32, 47, 84] emits multiple future actions, known as an *action chunk*, per inference call, amortizing inference cost across many control steps [15, 57, 77, 79, 82]. As illustrated in Figure 2, action chunking allows action execution to overlap with VLA inference. Second, asynchronous inference decouples the update frequencies of the VLM and the action head, allowing the action head to generate multiple action chunks from a single VLM forward pass [19, 112], reusing the latest available VLM output. As illustrated in Figure 2, multiple action head inference passes run after one VLM inference. While asynchronous VLAs explicitly parallelize the VLM and action head [19, 112], in VLAs that employ action chunking, it is also possible to decouple the VLM and action phase [15]. The VLM forward pass for chunk N+1 can be launched while the action head is still computing chunk N, pipelining the two phases across consecutive chunks on the same GPU. This is the execution model we target: the VLM phase of chunk N+1 runs concurrently with the action phase of chunk N via two CUDA streams on the same GPU. Each action head call still consumes the correct VLM output for its frame, preserving sequential semantics. This is timely because generating fresh action chunks at higher rates is critical for reactive control on dynamic tasks [15, 57, 82, 91]. However, we find that the naïve approach of assigning the two phases to separate CUDA streams [58, 67] falls short of optimal throughput. We identify the sources of inefficiency in Section 2.3.

2.2 GPU Architecture

Modern GPUs are parallel processors built on a Single Instruction Multiple Threads (SIMT) architecture (See Figure 3). Computation is distributed across an array of Streaming Multiprocessors (SMs).

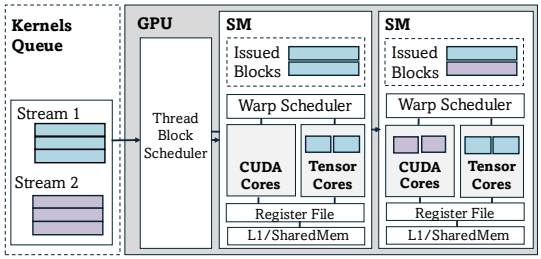


Figure 3: High-level overview of Streaming Multiprocessor (SM) and processing cores on modern GPUs.

Each SM contains Tensor Cores for dense matrix operations, general-purpose CUDA Cores for scalar arithmetic, a large register file, and a configurable shared memory/L1 cache [62]. When a kernel is launched, its threads are grouped into thread blocks, known as Cooperative Thread Arrays (CTAs). The CTA is the fundamental

scheduling granularity assigned to an SM, and each resident CTA on an SM is subdivided into warps of 32 threads that execute in lockstep. CTAs are dispatched to SMs via a hardware thread block scheduler. However, when scheduling concurrent workloads via CUDA streams [67], this scheduler has two idiosyncrasies [60, 81]: **(1) Greedy CTA Admission.** The CTA scheduler launches the maximum number of thread blocks per SM permitted by execution-context limits (registers, shared memory, thread slots), so kernels with large grid sizes saturate the per-SM context storage, preventing thread blocks of concurrent kernels from co-residing on the same SM [9]. Intra-SM co-execution of multiple concurrent kernels is thus rare when the running kernel has a large grid size [81, 94, 102]. **(2) Head-of-Line Blocking.** Even when SMs have spare resources along some dimensions (i.e., shared memory, register file, unused thread/block slots), the CTA scheduler does not dispatch thread blocks from another kernel until the first running kernel has fully launched all thread blocks defined by its grid size [28, 60, 81]. When a running kernel saturates one per-SM resource, such as registers, the remaining underutilized resources cannot be claimed by a waiting kernel. As a result, kernels with many thread blocks force smaller concurrent kernels to wait, serializing their execution.

2.3 Performance Characterization of VLAs on Modern GPUs

We perform a detailed performance study by profiling [66] two representative VLAs on three modern GPU architectures — Ada Lovelace (RTX 4090), Hopper (H100), and Blackwell (RTX 5090). We make three key observations.

First, VLAs significantly underutilize the GPU during inference (Figure 4). The median time-weighted SM throughput across all four models ranges from 18% to 30% of peak sustained throughput. Even the 20–30% of inference time during which SM utilization is highest never exceeds roughly 50% of peak. Figure 5 illustrates the utilization of Tensor Cores, ALU/FP/INT units, warp occupancy, and memory bandwidth of the VLM and action phases on RTX 4090, RTX 5090 and H100. The action phase significantly underutilizes the Tensor Cores, averaging below 3% utilization. This is because the action phase consists primarily of small matrix operations and element-wise computations that do not effectively leverage Tensor Cores, unlike the VLM phase which has a higher utilization of Tensor Cores because of its large reduced-precision GEMMs. L1 cache, L2 cache, and DRAM bandwidth utilization also differ substantially between the two phases.

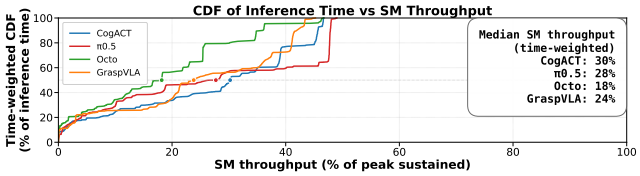


Figure 4: CDF of inference time vs. SM throughput for a single-image inference on the RTX 4090.

Second, the VLM phase exhibits higher arithmetic intensity than the action phase (Figure 6). The GEMM kernels of the VLM phase consistently achieve high tensor core utilization in all GPU architectures (37.4% on average).

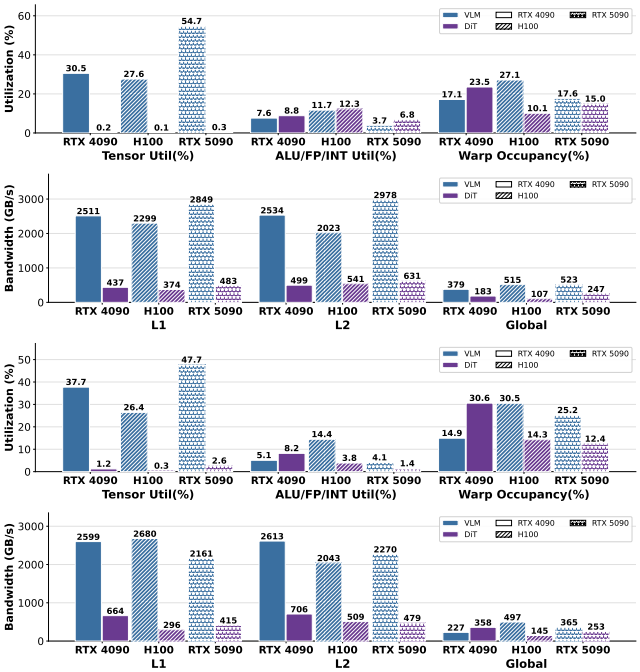


Figure 5: Resource utilization profiles of the VLM and Action Head phases for CogACT (top) and $\pi 0.5$ (bottom) across RTX 4090, H100, and RTX 5090.

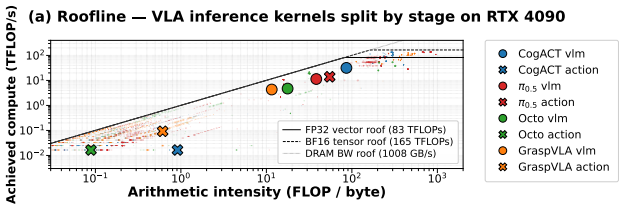


Figure 6: Roofline of VLA kernels on RTX 4090. Large markers are time-weighted centroids.

Third, when the two phases execute concurrently, the VLM phase runs at approximately the same rate as in isolation; the head-of-line blocking penalty falls entirely on the action phase, as observed in Nsight Systems profile for the four models (Figure 7b). Given these observations, overlapping the two phases will become beneficial when the penalty on the action head is mitigated.

2.4 Challenges

While both the VLM and action phases underutilize the GPU and stress different functional units, running them concurrently yields negligible throughput gain, as illustrated by the Naive Overlap in Figure 7b. This is caused by the head-of-line blocking behavior in GPUs, discussed in Section 2.2. Small action-phase kernels are serialized behind the large VLM-phase kernels by the GPU scheduler, even though the functional units needed by the action phase have substantial headroom on every SM throughout the execution [58].

This inefficiency arises from the CTA dispatcher's greedy admission policy, illustrated in Figure 8a. The VLM's GEMM kernels (blue) launch with large grid sizes, i.e., 768 CTAs. Because each SM

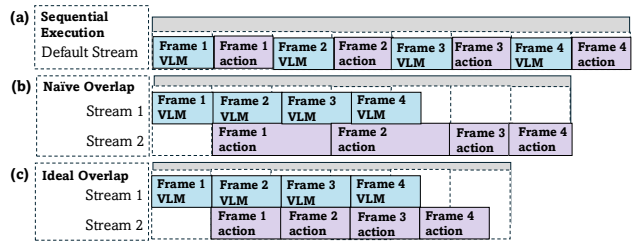


Figure 7: Execution patterns for asynchronous VLA inference. (a) Sequential execution serializes the two phases. (b) Naive two-stream overlap leaves the action stream lagging behind. (c) Ideal overlap fully hides action-head latency behind the next frame's VLM.

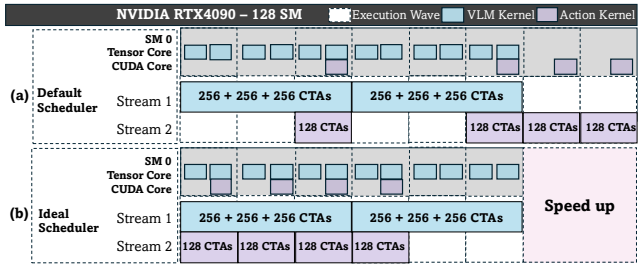


Figure 8: (a) Default GPU scheduling. Head-of-line blocking behavior. (b) Ideal co-scheduling. Fully overlapped execution within an SM.

can host only two such CTAs, given their register footprint, the kernel takes multiple waves (three, in this case) to drain. Meanwhile, the action phase launches a kernel with 128 thread blocks. The thread block dispatcher saturates every SM with two VLM CTAs in the first two waves of execution and admits action phase thread blocks (purple) only in the third wave, due to the head-of-line blocking behavior. Therefore, even when the VLM and action phases are launched in parallel, SMs spend most of the time executing only VLM kernels, saturating the Tensor Cores while leaving the CUDA Cores idle. The action phase does not execute despite ample headroom on the functional units it needs.

As established in Section 2.1, modern asynchronous VLA inference [15, 32] decouples the two phases across frames: the VLM encodes frame N+1 while the action head generates actions for frame N. Figure 7 illustrates the resulting execution patterns. Under sequential execution (a), the two phases serialize on a single stream. Under naive two-stream overlap (b), the VLM stream runs continuously, but head-of-line blocking at the GPU scheduler keeps the action phase kernels queued behind the VLM kernels, so action chunks are emitted at half the rate they would be otherwise, as observed in the Nsight System profiles. Under ideal overlap (c), each action-head invocation fully overlaps with the next frame's VLM phase, and chunks are emitted at a higher rate. The gap between (b) and (c) is the inefficiency we target, and closing it is challenging for two reasons specific to VLA workloads: (1) The VLM kernel with which a given action-head kernel overlaps is non-deterministic; there is no guarantee that a specific kernel from the VLM phase will overlap with a kernel from the action head phase since both kernels shift at microsecond granularity. This is because the VLM and action streams are launched independently and progress at different rates, so given a dispatched action-head kernel, the VLM kernel that's resident on the GPU at the same time as that action-head kernel is different across frames (Figure 7b). The dispatcher therefore sees a kernel mix that is a constantly shifting combination of one of many VLM kernels paired with one of many action-head kernels. (2) The action head is composed of micro-kernels. As shown in Table 1, the action phase consists of hundreds of small kernels with median duration on the order of a few microseconds. Combined with (1), this means the co-running kernel mix turns over at microsecond timescales, faster than any epoch-based profiling tool can track and far shorter than the millisecond-scale cost of preemption-based partition resizing, which must flush resident thread-block state or halt SMs to redistribute resources [29]. Any scheduler whose decision latency exceeds the duration of the kernels it is scheduling will spend more time deciding than computing.

Table 1: Execution profile of micro-kernels across different VLA workloads after applying a CUDA Graph capture.

Workload	Median (cyc)	<5K (%)	<10K (%)	<25K (%)	<50K (%)
CogACT [47]	7,156	18.9	67.2	88.9	95.2
Octo [84]	6,832	44.3	69.8	85.4	100.0
$\pi_{0.5}$ [32]	9,348	17.9	51.7	73.7	90.2
GRASP-VLA [27]	12,407	23.9	46.3	77.6	88.4

A VLA-aware co-scheduler must therefore be *micro-kernel responsive*: capable of reacting at microsecond granularity without paying preemption overhead or underutilizing resources due to stale decisions. Our goal in this work is to achieve the overlap depicted in Figure 8b, approaching the ideal pattern of Figure 7c.

2.5 Limitations of Existing Work

CUDA stream priorities affect only dispatch selection [9, 28], not admission, so head-of-line blocking persists. MPS [64] and MIG [65] target multi-tenant isolation across contexts and processes, not overlapping phases within a single pipeline.

Prior hardware-modification approaches to GPU scheduling, as discussed below, are insufficient for VLA inference. Table 2 evaluates the five closest prior frameworks against three essential properties that VLA inference must satisfy: *(i) Execution-unit awareness.* The scheduler must distinguish Tensor Core demand from CUDA-core and memory-bandwidth demand at dispatch time and avoid head-of-line blocking. In the VLA setting, the VLM phase is dominated by Tensor Core matrix multiplications while the action phase relies on CUDA-core arithmetic (Section 2.3). Co-locating thread blocks from both phases on the same SM would saturate disjoint execution units, raising aggregate utilization. (ii) *micro-kernel responsiveness.* The scheduler must react to VLA's rapidly shifting kernel mix at microsecond granularity. Preemption-based techniques cause significant overhead for micro-kernels from flushing resident thread-block state. (iii) *black-box kernel compatibility.* The scheduler should work with unmodified closed-source vendor binaries such as cuBLAS and cuDNN [24, 63]. The Primary Mechanism column captures the design philosophy each system adopts to approach these goals. As Table 2 shows, every prior system fails on at least one axis.

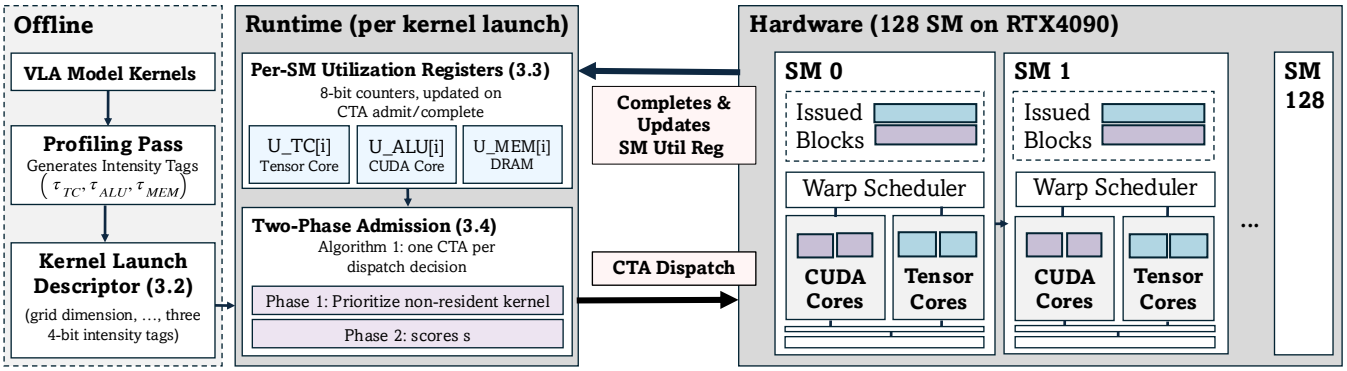


Figure 9: High-level overview of KerColle design.

Table 2: Qualitative comparison of prior GPU co-scheduling frameworks

Scheduler	Mechanism	Unit Aware	Micro-K. Resp.	Black-Box
SMK [94]	HW Ctx Switch	×	×	✓
Warped-Slicer [102]	Epoch + Static	×	×	✓
Maestro [72]	Epoch + 2-Stack	×	×	✓
GPUPool [81]	ML + TB/SM Cap	✓	×	✓
Kitsune [26]	Dataflow + L2 Q	✓	✓	×
KerColle (Ours)	**Dynamic Score**	✓	✓	✓

Unit Aware = TC vs. ALU execution-unit aware; *Micro-K. Resp.* = micro-kernel responsive; *Black-Box* = supports unmodified kernels.

Simultaneous Multikernel (SMK) [94] co-locates kernels on a single SM by partitioning registers, shared memory, and thread slots, but relies on preemption via Partial Context Switching, which drains in-flight instructions from an SM's pipelines and spills state to device memory. This overhead is prohibitive when a substantial fraction of VLA kernels complete in under 5K cycles (Table 1). **Maestro** [72] extends SMK by measuring co-located performance: it tests candidate partitions on designated Trial SMs during one epoch and applies the best-observed ratio to Follower SMs in the next. However, similar to SMK, Maestro resizes partitions by preempting SMs, introducing the same overhead that is incompatible with VLA's microsecond-scale kernel changes. **GPUPool** [81] modifies the thread block scheduler to bypass head-of-line blocking, but enforces a single static TBs/SM cap per application. This creates a static cap that cannot capture the diverse resource demands of rapidly changing VLA kernels, and it blocks the scheduler from filling idle units during transient VLM stalls. **Kitsune** [26] is execution-unit aware and preemption-free, but requires rewriting kernels with atomic L2 synchronization primitives—incompatible with closed-source operators such as cuBLAS and cuDNN that dominate the VLM phase. **Warped-Slicer** [102] uses online profiling to determine the number of thread blocks each co-located kernel may occupy on a shared SM. However, its design is mismatched to VLA workloads. First, non-deterministic launch times produce shifting VLM/action kernel pairings, forcing repeated invocation of its online profiling path; avoiding this would require either exhaustive offline cross-profiling of all kernel pairs or an online profile cache, neither of which the original design details. Second, VLA workloads are dominated by sub-5K-cycle micro-kernels (Table 1), which are shorter than Warped-Slicer's 5K-cycle profiling window and therefore fall back to its default partitioning. Since Warped-Slicer is non-preemptive, we include it as a baseline in Section 5.

Software-based approaches fail similarly. Orion's [80] runtime approach consumes offline profiles at host-side whole-kernel submission, so it cannot react to the dynamic resource utilization change of each SM or guarantee that CTAs from different kernels co-reside on the same SM. In addition to not being applicable to closed-source kernels, Tacker [107] and Adaptive Kernel Fusion [109] statically divide each fused thread block between the two kernels. If their execution times differ, resources assigned to the shorter kernel may become idle, and its dependent successor cannot begin until the entire fused kernel completes.

To conclude, VLAs demonstrate three characteristics that motivate a new design: *(i)* execution-unit awareness, *(ii)* microsecond-scale kernel responsiveness, and *(iii)* black-box vendor reliance. This motivates a lightweight CTA dispatcher that accepts per-kernel intensity tags, maintains per-SM utilization counters, and scores pending CTAs for complementarity without kernel modification or warp-scheduler changes.

3 Our Approach: KerColle

3.1 Overview

We propose KerColle, a GPU scheduling framework that enables high-throughput VLA inference. An overview of KerColle is shown in Figure 9. KerColle has two phases. The *offline profiling phase* profiles each kernel's per-CTA demand on the tensor, ALU, and memory pipelines and encodes it as an *intensity descriptor* (Section 3.2). The intensity descriptor is computed once per kernel, stored alongside its binary, and read by the hardware scheduler at launch with the kernel's launch descriptor. The *online scheduling phase* includes two components: (1) per-SM utilization registers (Table 4) that track aggregate execution-unit load, are updated on CTA dispatch and completion (Section 3.3); and (2) a dispatching algorithm that selects which pending kernel's thread block to issue by jointly considering kernel intensity descriptors and per-SM utilization registers, co-locating thread blocks with complementary resource demands on the same SM (Section 3.4).

KerColle addresses both challenges identified in Section 2.4: **Addressing Challenge 1: Head-of-line blocking.** KerColle resolves head-of-line blocking with two mechanisms in its dispatch policy (Algorithm 1). Phase 1 removes head-of-line blocking by

giving admission priority to kernels that have no CTA resident on the target SM, so a waiting kernel is admitted as soon as the SM has feasible capacity rather than after the running kernel has drained its grid. This guarantees concurrent execution on an SM among kernels. Phase 2 reads the intensity descriptor of each kernel (Section 3.2) and per-SM utilization registers (Section 3.3) to fill remaining SM capacity with thread blocks that best complement the pipelines currently underused on that SM. Execution-unit awareness thus emerges as a property of removing head-of-line blocking for VLAs. **Addressing Challenge 2: Responsiveness to microsecond-scale kernels.** KerColle's dispatch policy, Algorithm 1, reads the kernel intensity descriptors and per-SM utilization registers that are already available to the dispatcher at runtime. One invocation of Algorithm 1 evaluates K candidate kernels once and commits one CTA; it contains no iteration over CTA slots and no additional arbitration round. The cost of this decision is therefore independent of kernel duration and SM occupancy, making it applicable to kernels of any length.

3.2 Kernel Resource Profile Descriptor

3.2.1 Kernel Launch Descriptor Extension. Prior empirical studies and our profiling of GPUs identify that thread-block placement policies are based only on static occupancy constraints (e.g., register and shared-memory limits), ignoring which functional units (e.g., Tensor Cores, FP32 ALUs) each kernel stresses. This gap spans both sides of the hardware-software interface: the scheduler cannot observe per-kernel functional-unit usage, and existing APIs offer no channel to communicate it.

To bridge this gap, we extend the kernel launch descriptor, which is the metadata block passed from the host to the GPU at kernel launch time. The kernel launch descriptor already includes grid dimensions, per-thread register count, and shared memory allocation. We add three 4-bit fields to it that encode per-kernel execution-unit intensity values $(\tau_{TC}, \tau_{ALU}, \tau_{MEM})$ as defined in Table 3. Each τ_i is the fraction of the corresponding peak throughput consumed per CTA: τ_{TC} for Tensor Cores, τ_{ALU} for CUDA Cores, and τ_{MEM} for DRAM bandwidth. Together, the three fields form a 12-bit signature that the thread-block scheduler reads at dispatch time to determine whether thread blocks from different kernels will share an SM efficiently or contend for the same units. This 12-bit annotation can be communicated via an extension to existing per-kernel attribute APIs [67]. The change is confined to the driver.

Table 3: A kernel's Intensity Descriptor. Each field (4 bits) encodes a normalized intensity level in the range 0–15.

Field	Description
τ_{TC}	Tensor Core (TC) pipeline intensity. Fraction of peak TC throughput that CTAs consume.
τ_{ALU}	ALU pipeline intensity. Fraction of peak ALU (CUDA Core) throughput consumed per CTA.
τ_{MEM}	Memory bandwidth intensity. Fraction of peak DRAM bandwidth consumed per CTA.

3.2.2 Profile Metadata. The kernel intensity descriptors are obtained through a one-time offline profiling pass, performed once per model architecture at deployment time. Using hardware performance counters [66], each unique kernel is executed in isolation and profiled. For each distinct kernel, we record the achieved (1) tensor-core FLOP/s (FP16/BF16 matrix-multiply) as a fraction of the GPU's peak tensor-core FLOP/s; (2) FP32 FLOP/s on the CUDA Cores as a fraction of peak CUDA-core FP32 FLOP/s; and (3) DRAM throughput as a fraction of peak sustained bandwidth. The utilization numbers are then quantized to the 0-15 integer range to produce the corresponding τ fields in Table 3. The triplet $(\tau_{TC}, \tau_{ALU}, \tau_{MEM})$ is stored in a file and injected at kernel launch by a wrapper around the runtime kernel launch API.

For the VLA models evaluated in this paper, profiling all unique kernels typically takes two to three hours, but can take up to 6 hours depending on model size. This profiling cost happens *only once* per model, which is negligible in our target scenario: a VLA model is configured at deployment time and then serves millions of inference requests over the robot's operational lifetime, amortizing the one-time cost across the entire deployment.

Moreover, VLAs' input shapes are mainly fixed as their batch size, camera count, image resolution, action dimensions, data-types are determined at deployment and do not vary across inference calls. Input text-length is the only dimension that varies. We therefore profile a small set of text-length buckets to account for closed-source libraries selecting different kernels across input sequence lengths. If an unseen configuration is nevertheless dispatched at runtime and KerColle finds no matching descriptor, it will simply fall back to the baseline admission policy for that kernel.

3.3 Per-SM Utilization Registers

3.3.1 State Registers. We introduce three per-SM utilization registers, $U_{TC}[j]$, $U_{ALU}[j]$, and $U_{MEM}[j]$ (Table 4), one per execution-unit class (TC, ALU, MEM). Each register holds the sum of the corresponding τ field over all thread blocks currently resident on SM j. That is, $U_{TC}[j] = \sum_b \tau_{TC}(b)$ across resident blocks b on SM j, thereby giving an estimate of how heavily the Tensor Core execution unit is loaded. The three per-SM utilization registers are updated incrementally: a block's τ values are added on dispatch and subtracted on completion. The scheduler reads the triplet $(U_{TC}[j], U_{ALU}[j], U_{MEM}[j])$ of SM j at dispatch time to decide which kernel's thread block will be able to run on SM j without contending for units that are already saturated.

Table 4: Per-SM 8-bit utilization registers (range $[0, 255]$) maintained by the scheduler.

Register	Description
$U_{TC}[j]$	Estimated Tensor Core utilization on SM j.
$U_{ALU}[j]$	Estimated ALU pipeline utilization on SM j.
$U_{MEM}[j]$	Estimated memory bandwidth utilization on SM j.

3.3.2 Register Update Mechanism. The thread block scheduler already maintains per-SM resource accounting to determine admission feasibility: each SM's use of registers, shared memory, and thread block slots is tracked so that, at dispatch time, the scheduler checks whether a thread block of kernel K fits on SM_j, and upon thread block completion, the counters are updated to account for the released resources [28]. We extend this resource tracking mechanism present in the hardware scheduler. A thread block's τ values are added to $U_{TC}[j]$, $U_{ALU}[j]$, and $U_{MEM}[j]$ when the thread block is dispatched to SM j, and subtracted when the thread block completes. In doing so, we reuse the same thread block completion

signals that already trigger the release of registers, shared memory, and thread block slots back to the SM's free pool. Counters saturate at 255: an addition that would overflow is capped at 255, signaling the scheduler that the SM is fully loaded for that hardware resource. Since 0-255 is a large range, even if resources are capped, it would minimally affect the effectiveness of the scheduler. In addition, saturation is rare in practice, since it requires 17 co-resident thread blocks each at peak tensor-core intensity. This situation does not occur in the workloads we evaluate, where kernels at $\tau_{TC} = 15$ have register footprints that admit far fewer than 17 CTAs per SM.

Algorithm 1 Single-CTA Admission Decision on SM j

State (persistent, per SM j):
CTA_count(k, j): CTAs of kernel k resident on SM j
$U_{TC}[j], U_{ALU}[j], U_{MEM}[j]$: estimated utilization

Invoked once per CTA-admission decision; admits at most one CTA.
Phase 1: Prioritize one unrepresented feasible kernel
1: **for** each kernel k with pending CTAs, in dispatcher order **do**
2: **if** CTA_count$(k, j) = 0$ **then** ▷ k is not resident on SM j
3: **if** Feasible(k, j) **then** ▷ occupancy constraints permit admission
4: Admit(k, j)
5: Update resident-CTA count and per-SM utilization registers
6: **return** ▷ commit exactly one admission
7: **end if**
8: **end if**
9: **end for**

Phase 2: Select one feasible kernel by complementarity
Executed only when Phase 1 admits no CTA.
10: best_score $\leftarrow -1$
11: best_kernel $\leftarrow$ Null
12: **for** each kernel k with pending CTAs, in dispatcher order **do**
13: **if** $\neg$ Feasible(k, j) **then**
14: **continue** ▷ insufficient occupancy resources
15: **end if**
16: $s \leftarrow (U_{\max} - U_{TC}[j]) \cdot \tau_{TC}(k)$
$+ (U_{\max} - U_{ALU}[j]) \cdot \tau_{ALU}(k)$
$+ (U_{\max} - U_{MEM}[j]) \cdot \tau_{MEM}(k)$
17: **if** $s >$ best_score **then** ▷ strict comparison favors first-seen kernel
18: best_score $\leftarrow s$
19: best_kernel $\leftarrow k$
20: **end if**
21: **end for**
22: **if** best_kernel = Null **then** ▷ no pending kernel is feasible
23: **return**
24: **end if**
25: Admit(best_kernel, j)
26: Update resident-CTA count and per-SM utilization registers
27: **return** ▷ commit exactly one admission

On CTA completion (k, j):
CTA_count$(k, j) \leftarrow$ CTA_count$(k, j) - 1$
Subtract $\tau_{TC}(k), \tau_{ALU}(k), \tau_{MEM}(k)$ from $U_{TC}[j], U_{ALU}[j], U_{MEM}[j]$

3.4 Execution-Unit-Aware CTA Dispatch

3.4.1 Scoring Function. Co-locating kernels on an SM is beneficial only if their thread blocks need to use different hardware resources. Otherwise, contention may degrade performance. The thread block dispatcher must therefore (i) detect when a pending kernel's utilization of hardware resources complements an SM's current hardware resource utilization, and (ii) keep kernels co-resident on the same SM rather than draining one before admitting the next, in order to eliminate head-of-line blocking. For example, when a tensor-core-intensive kernel and a memory-intensive kernel are pending at the same SM, the dispatcher should admit thread blocks from each kernel in proportion to the SM's headroom along the corresponding pipelines, producing a co-resident mix that balances utilization across execution units. KerColle enables such kernel dispatching via Algorithm 1, applied independently to each SM. Algorithm 1 is invoked once for each ordinary CTA-admission decision on SM j and commits at most one CTA. It does not fill all available CTA slots within one invocation. If the SM has sufficient capacity after an admission, the work distributor invokes Algorithm 1 again during a subsequent admission cycle, using the updated state registers.

Phase 1: co-residency priority (Lines 1–8). Phase 1 ensures a kernel whose intensity profile scoring marginally but consistently below a co-running kernel's is not starved. Phase 2 could repeatedly select a same kernel and prevent kernel co-residency. Phase 1 avoids this by unconditionally admitting one thread block from each active kernel, guaranteeing forward progress on every SM. It collects the kernels that are pending, feasible, and have no CTA on SM j, then picks one by round-robin. The count $C[k, j]$ persists across invocations, so the next call sees that kernel as resident and picks a different one. Thus, every kernel that fits on SM j makes progress.

Phase 2: complementarity-driven selection (Lines 10–27). Phase 2 runs only when Phase 1 has not admitted any kernel: either every feasible kernel already has a CTA on SM j, or the kernel does not fit on SM j due to resource constraints. The dispatcher scans the pending kernels and skips any that do not fit in SM j's registers or shared memory (Line 13). For each remaining kernel, Line 16 computes a *compatibility score* s as the dot product of the kernel's descriptor with the SM j's headroom, stored in the per-SM state registers. Note $U_{\max}$ is the per-pipeline saturation cap, so $U_{\max} - U_*[j]$ measures the headroom remaining on pipeline $*$. s is large when the kernel demands the pipelines SM j is currently underusing. Lines 17–20 track the highest-scoring kernel, with the strict inequality breaking ties. The winning thread block is admitted (Line 25), SM j's resident-CTA count and utilization registers are updated (Line 26), and the algorithm returns. Re-scoring happens at the next admission decision on SM j, against the updated headroom.

3.4.2 Implementation Overhead. KerColle requires modest, localized changes to the GPU stack. **Compiler and runtime.** The kernel intensity descriptor is a 12-bit field appended to the kernel binary's metadata, read alongside the existing launch descriptor. **Hardware scheduler.** KerColle extends the thread-block dispatcher in two ways: *(i) Per-SM scheduling state:* three 8-bit utilization registers per SM (24 bits) join the dispatcher's per-SM status registers, updated by 8-bit saturating adders on CTA dispatch/completion using existing signals. Algorithm 1 additionally requires the resident-CTA count CTA_count(k, j); with $K = 2$ and at most 32 resident CTAs per SM (Table 5), this is two 6-bit counters, for 36 bits of state per SM. On an RTX 5090 (170 SMs), this is 765 bytes, which is negligible. The same counters are read directly by the fairness-augmented variants of Section 4. *(ii) Two-phase admission logic:* one invocation of Algorithm 1 iterates once over the K active kernels, where K is the number of concurrent streams, and commits exactly one Admit. The per-kernel score s is a three-term dot product of 4-bit intensity fields against 8-bit headroom values, computed in a small combinational block (six 8×4-bit multipliers for $K = 2$, two-level adder trees, ~1500 gates, <0.01% of the arbitration logic area) running in parallel with the baseline register file and shared-memory feasibility check. The logic scales with K, not with the 24–32 CTA slots per SM; filling multiple slots requires multiple arbitration cycles. No extra memory access, pipeline stage, or arbitration round is added, so dispatch completes within single-digit-cycle budget.

4 Evaluation Methodology

Simulation Methodology. We implement KerColle using Accel-Sim [38], a GPU simulator. We use two configurations that simulate NVIDIA RTX 4090 and RTX 5090 architectures. Table 5 lists the microarchitectural parameters of each configuration [38, 68].

Table 5: GPU Microarchitectural Parameters in Accel-Sim

Parameter	NVIDIA RTX 4090	NVIDIA RTX 5090
Generation	Ada Lovelace	Blackwell
SM Clock Frequency	2520 MHz	2407 MHz
SMs/GPU	128	170
Warp Schedulers/SM	4	4
Warp Scheduling Policy	GTO	GTO
Max #Thread Blocks/SM	24	32
Max Shared Memory/SM	100 KB	128 KB
Max Registers/SM	65536	65536
Register File Size	256 KB	256 KB
L1 Cache Size	128 KB	128 KB

Workloads. We evaluate KerColle using four VLA models that span a range of architectures and action-head designs: **CogACT** [47], **Octo** [84], $\pi_{0.5}$ [32], and **GRASP-VLA** [27]. We trace the GPU kernels of each model on physical RTX 4090 and RTX 5090 GPUs using NVBit [89], then replay the traces in Accel-Sim [38]. We perform inference on images from the Open X-Embodiment Dataset [25].

KerColle Implementations. We evaluate the baseline KerColle design (Section 3) — KerColle, and two variants that modify the Phase-2 scoring function in Algorithm 1: (1) KerColle-GridFair scales s by a grid-progress factor, $s_{GRID} = s \cdot G_k/(c_k(j) + 1)$, where $c_k(j)$ is kernel k's resident CTA count on SM j and G_k is its full grid size. This boosts kernels with large remaining grids and decays the score as a kernel accumulates CTAs, preventing any one kernel from monopolizing the SM. (2) KerColle-RegFair scales s by an inverse register-occupancy factor, $s_{REG} = s/(c_k(j) \cdot R_k + 1)$, where R_k is the per-CTA register count; the denominator is the kernel's current register-file footprint on SM j, so register-heavy kernels decay fastest and leave capacity for co-residents.

Comparison Points. We compare KerColle against two baselines: (i) *Baseline*, where the VLM of one frame and the action phases of another frame are overlapped by the default GPU scheduler; and (ii) Warped-Slicer [102], the closest prior work, where the VLM and action phases of different frames are overlapped using intra-SM concurrent-kernel scheduling. For each kernel, Warped-Slicer [102] sweeps the number of resident thread blocks from one to full SM capacity, recording IPC at each CTA count over a 5K-cycle profiling window. The resulting per-kernel IPC-vs-CTA curves are used to assign static per-SM CTA quotas that maximize aggregate IPC.

5 Evaluation

We evaluate KerColle by measuring the throughput (actions per second) of one VLM phase overlapped with one action phase using four models [27, 32, 47, 84]. We note that the throughput varies across models due to differences in the default action chunk size.

5.1 Performance Analysis of KerColle

Figures 10 and 11 present the performance benefits of KerColle, measuring throughput on RTX 4090 and RTX 5090 configurations using Accel-Sim. We make five key observations.

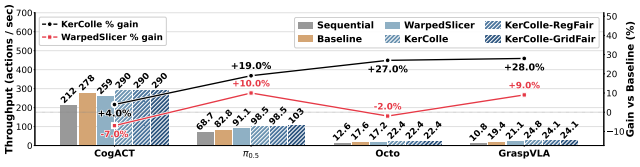


Figure 10: Left: VLA inference throughput on RTX 4090. Right: Percentage increase in throughput gain of KerColle and Warped-Slicer relative to Baseline.

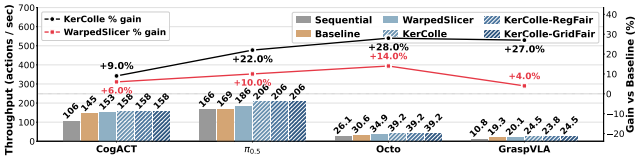


Figure 11: Left: VLA inference throughput on RTX 5090. Right: Throughput gain of KerColle and Warped-Slicer relative to Baseline.

First, we observe that all three variants of KerColle consistently outperform Warped-Slicer in throughput. The red and black lines indicate the percentage of Warped-Slicer's and KerColle's throughput gain relative to the Baseline. The red line is consistently below the black line. This is due to two key reasons: (1) Warped-Slicer's 5K-cycle profiling window is too long for VLA workloads: 18–44% of kernels, depending on the model, finish in fewer than 5K cycles, forcing Warped-Slicer to fall back to the baseline scheduler for these kernels (Table 1). (2) Warped-Slicer's profiling assigns a fixed target CTA count for each SM. This works well for large kernels that can fill the SM, but VLA action phases frequently launch small kernels that cannot reach the profiled target, leaving SMs idle for the entire 5K-cycle window. This problem is worse on the RTX 5090, where the increased SM count results in more idle units, which can cause a slowdown relative to the baseline scheduler.

Second, KerColle's throughput gains grow with GPU capability. We observe that KerColle's action throughput rises from 19% over baseline on the RTX 4090 to 22% on the RTX 5090 for $\pi_{0.5}$, and from 4% to 9% for CogACT. As the SM count grows from 128 to 170 from the RTX 4090 to the RTX 5090 alongside higher FLOPS and memory bandwidth, individual VLA kernels underutilize the GPU more severely, leaving more SMs idle. This widens the gap that KerColle can recover through overlap-aware scheduling. The baseline scheduler cannot exploit these idle SMs due to head-of-line blocking, whereas KerColle fills them with work from the second stream, thus its gains improve further in more powerful GPUs.

Third, CogACT achieves the smallest throughput gain under KerColle relative to the baseline for two reasons: (1) CogACT achieves the highest concurrent execution under the baseline scheduler, leaving little headroom for KerColle to improve (Figure 12), and (2) the majority of CogACT kernels contain fewer than five thread blocks, limiting the opportunity to co-schedule kernels from both streams onto the same SM (Figure 13). Together, these factors leave KerColle little room to improve over the baseline.

Fourth, the fairness-augmented variants yield negligible gains over KerColle in nearly all cases. KerColle-GridFair matches KerColle on six of the eight configurations and improves throughput in only one, raising $\pi_{0.5}$ on the RTX 4090 from 98.5 to 103 actions/sec. This is due to $\pi_{0.5}$'s kernel size distribution (Figure 13):

65% of its concurrent kernels contain 11+ CTAs with a median of 30, the highest across all evaluated workloads. These kernels are large enough that the scheduler requires multiple admission decisions to fully schedule them, yet KerColle may repeatedly prioritize the larger co-running stream, delaying the smaller one. KerColle-GridFair's grid-progress factor corrects this by increasing the priority of the delayed kernel as the larger stream accumulates more resident CTAs. This effect does not appear on the RTX 5090, where the additional SM capacity is sufficient to accommodate both streams simultaneously under KerColle, eliminating any admission contention for KerColle-GridFair to resolve.

Finally, KerColle-RegFair occasionally degrades throughput: GraspVLA drops from 24.5 to 23.8 actions/sec on the RTX 5090 and from 24.8 to 24.1 on the RTX 4090. This is because KerColle-RegFair penalizes register-heavy kernels and prioritizes register-light ones instead. However, for GraspVLA, the register-light kernels do not utilize the SM well, so admitting them over the register-heavy ones ultimately wastes SM capacity and reduces throughput.

We conclude that KerColle consistently improves throughput across diverse VLA models and GPU architectures, with gains growing as GPUs become more powerful. KerColle alone delivers significant performance improvements, and the fairness-augmented variants achieve comparable gains.

5.2 Concurrent SM Utilization Analysis

Figure 12 reports, for each model on both the RTX 4090 and RTX 5090, the fraction of the overlap window during which both streams have at least one thread block resident on every SM, measured under both the baseline scheduler and KerColle (left y-axis), alongside the corresponding KerColle throughput gain (right y-axis).

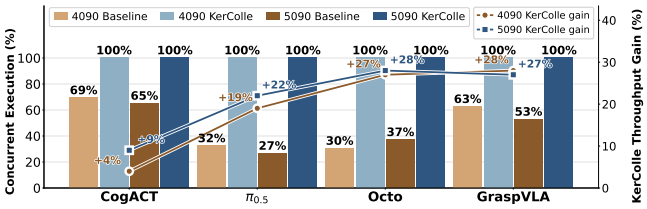


Figure 12: Concurrent SM utilization under the baseline scheduler and KerColle across all models and GPUs (left y-axis), and KerColle throughput gain relative to baseline (right y-axis).

We observe that Octo and GraspVLA achieve the highest throughput gains under KerColle at 27%–28% across both GPUs, while CogACT achieves the lowest at 4%–9%. This is because of the differences in concurrency headroom left by the baseline scheduler. Octo achieves only 30–37% concurrent execution under the baseline scheduler, meaning execution is serial during 63–70% of the overlap window despite both streams being ready to run. KerColle converts this serialized execution into true overlap, delivering significant throughput gains. Conversely, CogACT already achieves 65–69% concurrent execution under the baseline, leaving little room for KerColle to improve. We conclude that the degree of concurrency left unexploited by the baseline scheduler is the primary determinant of KerColle's throughput gains.

5.3 Effect of Grid-Size Disparity

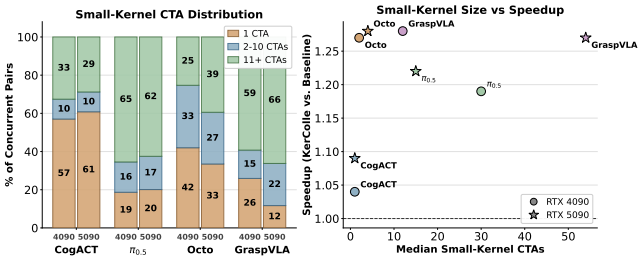


Figure 13: Left: Distribution of concurrent kernel pairs grouped by the smaller kernel's thread block count. Right: Throughput gains of KerColle over baseline for four models correlated to the median thread block count of each model's smaller kernels.

Figure 13 analyzes the thread block count of the smaller kernel in each concurrent kernel pair. For each model, the left part of the figure shows the distribution of concurrent kernel pairs grouped by the smaller kernel's thread block count as exactly 1, 2–10, or 11+ thread blocks. The right part of the figure plots each model's throughput gain over baseline, correlated to the median thread block count of its smaller kernels. Intuitively, the right part of the figure summarizes how small the smaller kernels of concurrent kernel pairs are for each model, and presents KerColle throughput gains over baseline scheduler. Grid-size disparity refers to the difference in the number of thread blocks between two kernels that are co-scheduled on the GPU at the same time.

We observe that models whose smaller concurrent kernel contains more thread blocks, such as GraspVLA and $\pi_{0.5}$, experience greater throughput gains under KerColle. When the smaller kernel in a concurrent pair has very few thread blocks, the grid-size disparity between the two co-scheduled kernels is large, meaning the smaller kernel occupies only a few SMs and overlap is restricted to those SMs alone. In the extreme case of a single-thread-block kernel, the block is pinned to a single SM regardless of the dispatch policy, leaving KerColle no flexibility to redistribute work across SMs. Conversely, when the smaller kernel has more thread blocks, the grid-size disparity is smaller and both kernels can be interleaved across a larger fraction of SMs, giving the dispatcher more opportunity to overlap them. GraspVLA illustrates the favorable case: only 26% of its concurrent pairs contain a single-thread-block kernel and its smaller kernels have a median of 11+ thread blocks, allowing the dispatcher to interleave both streams across many SMs and yielding a 28% throughput gain. Although 42% of Octo's smaller kernels contain only a single thread block, the baseline scheduler left substantial concurrency headroom (Section 5.2), and KerColle is therefore able to deliver high throughput gains for Octo as well. $\pi_{0.5}$ gains less than GraspVLA despite having ample concurrency headroom and a favorable thread block distribution in its smaller kernels. This is because of the arithmetic intensity of $\pi_{0.5}$'s kernels, as illustrated in Figure 6: both phases land in the compute-bound region of the roofline model, meaning co-execution finds little complementary headroom across execution pipelines to exploit. Overall, we conclude that grid-size disparity between

concurrent kernel pairs is a key factor in the overlap capability and throughput gains that KerColle can achieve.

5.4 Warp Occupancy Analysis

Figures 14 and 15 report average warp occupancy on RTX 4090 and RTX 5090 to show how effectively each scheduler utilizes the per-SM issue capacity. We observe that KerColle increases warp occupancy across nearly all models, while Warped-Slicer frequently reduces it below the baseline scheduler. Specifically, KerColle improves occupancy over the baseline in seven of eight configurations, with the largest gains occurring where throughput gains are also highest: KerColle improves warp occupancy in $\pi_{0.5}$ from 17.9% to 25.0% on RTX 4090 and from 15.2% to 19.5% on RTX 5090, and in GraspVLA from 23.2% to 28.6% on RTX 4090 and 23.0% to 24.0% on RTX 5090. These gains confirm that KerColle improves throughput by filling otherwise-idle SM issue slots with useful work from both streams, rather than simply changing the order in which kernels are dispatched. In contrast, Warped-Slicer frequently *reduces* warp occupancy below the baseline scheduler: Warped-Slicer decreases warp occupancy in Octo from 16.3% to 9.8% on RTX 4090, CogACT from 13.6% to 12.4% on RTX 5090, and GraspVLA from 23.0% to 22.0% on RTX 5090. This is because, under Warped-Slicer, SMs are pinned to a profiling target that they cannot reach, so they hold idle warp slots open for the remaining 5K-cycle window, reducing overall occupancy. Overall, we conclude that KerColle effectively improves warp occupancy across various VLA models and GPUs.

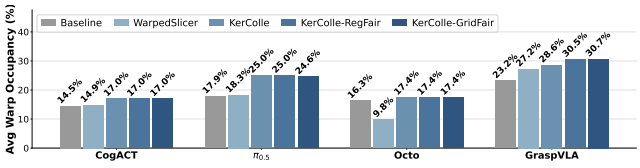


Figure 14: Warp occupancy of different KerColle configurations and Warped-Slicer on RTX 4090.

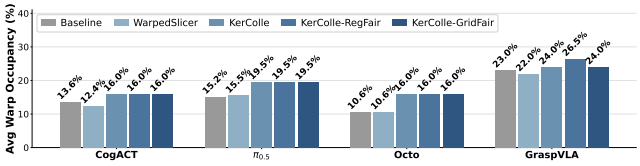


Figure 15: Warp occupancy of different KerColle configurations and Warped-Slicer on RTX 5090.

5.5 Sensitivity to Intra-Kernel CTA Heterogeneity

When a CTA within the kernel has a resource profile that differs from the majority of CTAs in the same kernel, it will receive the kernel-level intensity tag representing the majority of CTAs and can therefore be misrepresented. Consequently, the scheduler may place the CTA on an SM based on an inaccurate estimate of its resource utilization, resulting in either increased contention or underutilization. To characterize this impact, we inspected 586 unique launch configurations across four VLA models. CTAs differed from their configuration's typical behavior by at least 10% in 24.7% of CogACT configurations, 18.5% of π0.5, 11.6% of Octo, and 22.0% of GraspVLA. Most cases come from boundary GEMM tiles and copy CTAs performing different roles. Using CogACT on RTX 4090 as a case study, we augmented the simulator to track exact per-CTA profiles during scheduling. Throughput increases only from 290 to 294 actions/s, indicating that CTA-level deviations are too small to change the score s and, consequently, the scheduling decisions.

6 Related Work

To our knowledge, KerColle is the first work to leave closed-source VLM kernels unmodified and augment the thread block dispatcher to co-locate thread blocks from kernels exhibiting different functional-unit demands while mitigating head-of-line blocking. Prior approaches to GPU kernel co-location each have limitations for VLA inference.

Preemption and Runtime Interposition. Hardware preemption [69, 71, 72, 94, 99] is too coarse for sub-microsecond kernels. Runtime interposition [22, 23, 29, 60, 80, 110] adds per-launch overhead that short action-head kernels cannot amortize. Persistent-block transformations [6, 70, 100, 106, 108] and kernel fusion [46, 107, 109] require source code access. Thread block scheduler modifications [7, 40, 43, 44, 51, 78, 98] and intra-kernel auxiliary work [13, 30, 37, 88] either fail to eliminate head-of-line blocking or operate within a single kernel.

Spatial Partitioning. Prior approaches in spatial partitioning [3–5, 48, 64, 65, 111] allocate disjoint sets of SMs to concurrently execute kernels. While this approach mitigates head-of-line blocking, it is ineffective for VLAs since a VLA's two phases both underutilize their assigned SMs, so partitioning the SMs into kernel-exclusive sets does not mitigate the underutilization. For VLAs, the goal is to co-locate the kernels from distinct phases onto the same SM to exploit the different resource demands.

Data- and Locality-Aware Scheduling. Prior approaches explore data- and cache-aware intra-SM warp schedulers [45, 52, 54, 59, 75, 76], locality-aware CTA scheduling that captures inter-kernel and inter-CTA reuse [11, 21, 31, 86, 87, 92], and CTA dispatchers that honor inter-block DAG dependencies [1, 2]. However, these lines of work target cache contention within a single application or thread-block data dependencies—neither of which is the primary bottleneck in VLA workloads—and do not address head-of-line blocking or co-execute thread blocks from kernels with complementary execution-unit demands.

7 Conclusion

We characterized an important emerging workload for GPUs in Physical AI and we identified a gap in existing GPU scheduling frameworks: applications whose set of concurrently running kernels shifts continuously render static or ahead-of-time scheduling policies largely ineffective; applications dominated by short-running, numerous kernels leave no headroom for online profiling or preemption-based mechanisms. To address these challenges, we propose a lightweight scheduling framework that leverages online SM utilization and individual kernel resource requirements to intelligently and dynamically co-schedule kernels to efficiently overlap execution. We demonstrate that KerColle improves four representative VLA models' throughputs by enabling more efficient concurrent execution of their independent heterogeneous phases.

References

[1] AmirAli Abdolrashidi, Hodjat Asghari Esfeden, Ali Jahanshahi, Kaustubh Singh, Nael Abu-Ghazaleh, and Daniel Wong. 2021. BlockMaestro: enabling programmer-transparent task-based execution in GPU systems. In *Proceedings of the 48th Annual International Symposium on Computer Architecture* (Virtual Event, Spain) *(ISCA '21)*. IEEE Press, 333–346. doi:10.1109/ISCA52012.2021.00034

[2] AmirAli Abdolrashidi, Devashree Tripathy, Mehmet Esat Belviranli, Laxmi Narayan Bhuyan, and Daniel Wong. 2017. Wireframe: supporting data-dependent parallelism through dependency graph execution in GPUs. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (Cambridge, Massachusetts) *(MICRO-50 '17)*. Association for Computing Machinery, New York, NY, USA, 600–611. doi:10.1145/3123939.3123976

[3] Jacob T. Adriaens, Katherine Compton, Nam Sung Kim, and Michael J. Schulte. 2012. The case for GPGPU spatial multitasking. In *IEEE International Symposium on High-Performance Comp Architecture*. 1–12. doi:10.1109/HPCA.2012.6168946

[4] Paula Aguilera, Katherine Morrow, and Nam Sung Kim. 2014. Fair share: Allocation of GPU resources for both performance and fairness. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*. 440–447. doi:10.1109/ICCD.2014.6974717

[5] Paula Aguilera, Katherine Morrow, and Nam Sung Kim. 2014. QoS-aware dynamic resource allocation for spatial-multitasking GPUs. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 726–731. doi:10.1109/ASPDAC.2014.6742976

[6] Tyler Allen, Xizhou Feng, and Rong Ge. 2019. Slate: Enabling Workload-Aware Efficient Multiprocessing for Modern GPGPUs. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 252–261. doi:10.1109/IPDPS.2019.00035

[7] Mihir Awatramani, Joseph Zambreno, and Diane Rover. 2013. Increasing GPU throughput using kernel interleaved thread block scheduling. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*. 503–506. doi:10.1109/ICCD.2013.6657093

[8] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-VL: A Versatile Vision-Language Model for Understanding, Localization, Text Reading, and Beyond. arXiv:2308.12966 [cs.CV] https://arxiv.org/abs/2308.12966

[9] Joshua Bakita and James H. Anderson. 2023. Hardware Compute Partitioning on NVIDIA GPUs. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 54–66. doi:10.1109/RTAS58335.2023.00012

[10] Kostas Bekris, Kris Hauser, Sylvia Herbert, Jingjin Yu, Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. 2025. Diffusion policy: Visuomotor policy learning via action diffusion. *Int. J. Rob. Res.* 44, 10–11 (Sept. 2025), 1684–1704. doi:10.1177/02783649241273668

[11] Mehmet E. Belviranli, Farzad Khorasani, Laxmi N. Bhuyan, and Rajiv Gupta. 2016. CuMAS: Data Transfer Aware Multi-Application Scheduling for Shared GPUs. In *Proceedings of the 2016 International Conference on Supercomputing* (Istanbul, Turkey) *(ICS '16)*. Association for Computing Machinery, New York, NY, USA, Article 31, 12 pages. doi:10.1145/2925426.2926271

[12] Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, Thomas Unterthiner, Daniel Keysers, Skanda Koppula, Fangyu Liu, Adam Grycner, Alexey Gritsenko, Neil Houlsby, Manoj Kumar, Keran Rong, Julian Eisenschlos, Rishabh Kabra, Matthias Bauer, Matko Bošnjak, Xi Chen, Matthias Minderer, Paul Voigtlaender, Ioana Bica, Ivana Balazevic, Joan Puigcerver, Pinelopi Papalampidi, Olivier Henaff, Xi Xiong, Radu Soricut, Jeremiah Harmsen, and Xiaohua Zhai. 2024. PaliGemma: A versatile 3B VLM for transfer. arXiv:2407.07726 [cs.CV] https://arxiv.org/abs/2407.07726

[13] Hossein Bitalebi, Vahid Geraeinejad, Farshad Safaei, and Masoumeh Ebrahimi. 2023. LATOA: Load-Aware Task Offloading and Adoption in GPU. In *Proceedings of the 15th Workshop on General Purpose Processing Using GPU* (Montreal, Canada) *(GPGPU '23)*. Association for Computing Machinery, New York, NY, USA, 7–13. doi:10.1145/3589236.3589243

[14] 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. 2026. π_0: A Vision-Language-Action Flow Model for General Robot Control. arXiv:2410.24164 [cs.LG] https://arxiv.org/abs/2410.24164

[15] Kevin Black, Manuel Y. Galliker, and Sergey Levine. 2025. Real-Time Execution of Action Chunking Flow Policies. arXiv:2506.07339 [cs.RO] https://arxiv.org/abs/2506.07339

[16] 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. 2023. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. arXiv:2307.15818 [cs.RO] https://arxiv.org/abs/2307.15818

[17] 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. 2023. RT-1: Robotics Transformer for Real-World Control at Scale. arXiv:2212.06817 [cs.RO] https://arxiv.org/abs/2212.06817

[18] Chilam Cheang, Sijin Chen, Zhongren Cui, Yingdong Hu, Liqun Huang, Tao Kong, Hang Li, Yifeng Li, Yuxiao Liu, Xiao Ma, Hao Niu, Wenxuan Ou, Wanli Peng, Zeyu Ren, Haixin Shi, Jiawen Tian, Hongtao Wu, Xin Xiao, Yuyang Xiao, Jiafeng Xu, and Yichu Yang. 2025. GR-3 Technical Report. arXiv:2507.15493 [cs.RO] https://arxiv.org/abs/2507.15493

[19] Hao Chen, Jiaming Liu, Chenyang Gu, Zhuoyang Liu, Renrui Zhang, Xiaoqi Li, Xiao He, Yandong Guo, Chi-Wing Fu, Shanghang Zhang, and Pheng-Ann Heng. 2025. Fast-in-Slow: A Dual-System Foundation Model Unifying Fast Manipulation within Slow Reasoning. arXiv:2506.01953 [cs.RO] https://arxiv.org/abs/2506.01953

[20] Jiayi Chen, Wenxuan Song, Pengxiang Ding, Ziyang Zhou, Han Zhao, Feilong Tang, Donglin Wang, and Haoang Li. 2026. Unified Diffusion VLA: Vision-Language-Action Model via Joint Discrete Denoising Diffusion Process. arXiv:2511.01718 [cs.RO] https://arxiv.org/abs/2511.01718

[21] Li-Jhan Chen, Hsiang-Yun Cheng, Po-Han Wang, and Chia-Lin Yang. 2017. Improving GPGPU Performance via Cache Locality Aware Thread Block Scheduling. *IEEE Computer Architecture Letters* 16, 2 (2017), 127–131. doi:10.1109/LCA.2017.2693371

[22] Quan Chen, Hailong Yang, Minyi Guo, Ram Srivatsa Kannan, Jason Mars, and Lingjia Tang. 2017. Prophet: Precise QoS Prediction on Non-Preemptive Accelerators to Improve Utilization in Warehouse-Scale Computers. *SIGARCH Comput. Archit. News* 45, 1 (April 2017), 17–32. doi:10.1145/3093337.3037700

[23] Quan Chen, Hailong Yang, Jason Mars, and Lingjia Tang. 2016. Baymax: QoS Awareness and Increased Utilization for Non-Preemptive Accelerators in Warehouse Scale Computers. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. Association for Computing Machinery, New York, NY, USA, 681–696. doi:10.1145/2872362.2872368

[24] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient Primitives for Deep Learning. *arXiv preprint arXiv:1410.0759* (2014).

[25] Open X-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 Mart'in-Mart'in, 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. 2023. Open X-Embodiment: Robotic Learning Datasets and RT-X Models. https://arxiv.org/abs/2310.08864.

[26] Michael Davies, Neal Crago, Karthikeyan Sankaralingam, and Stephen W. Keckler. 2025. Kitsune: Enabling Dataflow Execution on GPUs. arXiv:2502.18403 [cs.AR] https://arxiv.org/abs/2502.18403

[27] Shengliang Deng, Mi Yan, Songlin Wei, Haixin Ma, Yuxin Yang, Jiayi Chen, Zhiqi Zhang, Taoyu Yang, Xuheng Zhang, Wenhao Zhang, Heming Cui, Zhizheng Zhang, and He Wang. 2025. GraspVLA: a Grasping Foundation Model Pre-trained on Billion-scale Synthetic Action Data. arXiv:2505.03233 [cs.RO] https://arxiv.org/abs/2505.03233

[28] Guin Gilman and Robert J. Walls. 2021. Characterizing Concurrency Mechanisms for NVIDIA GPUs under Deep Learning Workloads. arXiv:2110.00459 [cs.DC] https://arxiv.org/abs/2110.00459

[29] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. 2022. Microsecond-scale Preemption for Concurrent GPU-accelerated DNN Inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 539–558. https://www.usenix.org/conference/osdi22/presentation/han

[30] Khoa Ho, Hui Zhao, Adwait Jog, and Saraju Mohanty. 2022. Improving GPU Throughput through Parallel Execution Using Tensor Cores and CUDA Cores. In *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 223–228. doi:10.1109/ISVLSI54635.2022.00051

[31] Muhammad Huzaifa, Johnathan Alsop, Abdulrahman Mahmoud, Giordano Salvador, Matthew D. Sinclair, and Sarita V. Adve. 2020. Inter-kernel Reuse-aware Thread Block Scheduling. *ACM Trans. Archit. Code Optim.* 17, 3, Article 24 (Aug. 2020), 27 pages. doi:10.1145/3406538

[32] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. 2025. $\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization. arXiv:2504.16054 [cs.LG] https://arxiv.org/abs/2504.16054

[33] Wenqi Jiang, Jason Clemons, Karu Sankaralingam, and Christos Kozyrakis. 2026. How Fast Can I Run My VLA? Demystifying VLA Inference Performance with VLA-Perf. arXiv:2602.18397 [cs.RO] https://arxiv.org/abs/2602.18397

[34] Yuhua Jiang, Shuang Cheng, Yan Ding, Feifei Gao, and Biqing Qi. 2025. AsyncVLA: Asynchronous Flow Matching for Vision-Language-Action Models. arXiv:2511.14148 [cs.RO] https://arxiv.org/abs/2511.14148

[35] Siddharth Karamcheti, Suraj Nair, Ashwin Balakrishna, Percy Liang, Thomas Kollar, and Dorsa Sadigh. 2024. Prismatic VLMs: Investigating the Design Space of Visually-Conditioned Language Models. arXiv:2402.07865 [cs.CV] https://arxiv.org/abs/2402.07865

[36] Kento Kawaharazuka, Jihoon Oh, Jun Yamada, Ingmar Posner, and Yuke Zhu. 2025. Vision-Language-Action Models for Robotics: A Review Towards Real-World Applications. *IEEE Access* 13 (2025), 162467–162504. doi:10.1109/access.2025.3609980

[37] Onur Kayiran, Adwait Jog, Ashutosh Pattnaik, Rachata Ausavarungnirun, Xulong Tang, Mahmut T. Kandemir, Gabriel H. Loh, Onur Mutlu, and Chita R. Das. 2016. μC-States: Fine-grained GPU Datapath Power Management. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation* (Haifa, Israel) *(PACT '16)*. Association for Computing Machinery, New York, NY, USA, 17–30. doi:10.1145/2967938.2967941

[38] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 473–486. doi:10.1109/ISCA45697.2020.00047

[39] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, Vitor Guizilini, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Muhammad Zubair Irshad, Donovon Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. 2025. DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset. arXiv:2403.12945 [cs.RO] https://arxiv.org/abs/2403.12945

[40] Jiho Kim, Jehee Cha, Jason Jong Kyu Park, Dongsuk Jeon, and Yongjun Park. 2019. Improving GPU Multitasking Efficiency Using Dynamic Resource Sharing . *IEEE Computer Architecture Letters* 18, 01 (Jan. 2019), 1–5. doi:10.1109/LCA.2018.2889042

[41] 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. 2024. OpenVLA: An Open-Source Vision-Language-Action Model. *arXiv preprint arXiv:2406.09246* (2024).

[42] Moo Jin Kim, Chelsea Finn, and Percy Liang. 2025. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success. arXiv:2502.19645 [cs.RO] https://arxiv.org/abs/2502.19645

[43] Sejin Kim and Yoonhee Kim. 2022. K-Scheduler: dynamic intra-SM multitasking management with execution profiles on GPUs. *Cluster Computing* 25, 1 (Feb. 2022), 597–617. doi:10.1007/s10586-021-03429-7

[44] Minseok Lee, Seokwoo Song, Joosik Moon, John Kim, Woong Seo, Yeongon Cho, and Soojung Ryu. 2014. Improving GPGPU resource utilization through alternative thread block scheduling. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 260–271. doi:10.1109/HPCA.2014.6835937

[45] Shin-Ying Lee and Carole-Jean Wu. 2014. CAWS: criticality-aware warp scheduling for GPGPU workloads. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation* (Edmonton, AB, Canada) *(PACT '14)*. Association for Computing Machinery, New York, NY, USA, 175–186. doi:10.1145/2628071.2628107

[46] Ao Li, Bojian Zheng, Gennady Pekhimenko, and Fan Long. 2022. Automatic Horizontal Fusion for GPU Kernels . In *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE Computer Society, Los Alamitos, CA, USA, 14–27. doi:10.1109/CGO53902.2022.9741270

[47] Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, Xiaofan Wang, Bei Liu, Jianlong Fu, Jianmin Bao, Dong Chen, Yuanchun Shi, Jiaolong Yang, and Baining Guo. 2024. CogACT: A Foundational Vision-Language-Action Model for Synergizing Cognition and Action in Robotic Manipulation. arXiv:2411.19650 [cs.RO] https://arxiv.org/abs/2411.19650

[48] Yun Liang and Xiuhong Li. 2017. Efficient Kernel Management on GPUs. *ACM Trans. Embed. Comput. Syst.* 16, 4, Article 115 (May 2017), 24 pages. doi:10.1145/3070710

[49] Yuanchang Liang, Xiaobo Wang, Kai Wang, Shuo Wang, Xiaojiang Peng, Haoyu Chen, David Kim Huat Chua, and Prahlad Vadakkepat. 2026. Adaptive Action Chunking at Inference-time for Vision-Language-Action Models. arXiv:2604.04161 [cs.RO] https://arxiv.org/abs/2604.04161

[50] Zhixuan Liang, Yizhuo Li, Tianshuo Yang, Chengyue Wu, Sitong Mao, Tian Nian, Liuao Pei, Shunbo Zhou, Xiaokang Yang, Jiangmiao Pang, Yao Mu, and

Ping Luo. 2025. Discrete Diffusion VLA: Bringing Discrete Diffusion to Action Decoding in Vision-Language-Action Policies. arXiv:2508.20072 [cs.CV] https://arxiv.org/abs/2508.20072

[51] Zhen Lin, Hongwen Dai, Michael Mantor, and Huiyang Zhou. 2019. Coordinated CTA Combination and Bandwidth Partitioning for GPU Concurrent Kernel Execution. *ACM Trans. Archit. Code Optim.* 16, 3, Article 23 (June 2019), 27 pages. doi:10.1145/3326124

[52] Zhen Lin, Hongwen Dai, Michael Mantor, and Huiyang Zhou. 2019. Coordinated CTA Combination and Bandwidth Partitioning for GPU Concurrent Kernel Execution. *ACM Trans. Archit. Code Optim.* 16, 3, Article 23 (June 2019), 27 pages. doi:10.1145/3326124

[53] Jiaming Liu, Hao Chen, Pengju An, Zhuoyang Liu, Renrui Zhang, Chenyang Gu, Xiaoqi Li, Ziyu Guo, Sixiang Chen, Mengzhen Liu, Chengkai Hou, Mengdi Zhao, KC alex Zhou, Pheng-Ann Heng, and Shanghang Zhang. 2025. HybridVLA: Collaborative Diffusion and Autoregression in a Unified Vision-Language-Action Model. arXiv:2503.10631 [cs.CV] https://arxiv.org/abs/2503.10631

[54] Jiwei Liu, Jun Yang, and Rami Melhem. 2015. SAWS: synchronization aware GPGPU warp scheduling for multiple independent warp schedulers. In *Proceedings of the 48th International Symposium on Microarchitecture* (Waikiki, Hawaii) *(MICRO-48)*. Association for Computing Machinery, New York, NY, USA, 383–394. doi:10.1145/2830772.2830822

[55] Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. 2025. RDT-1B: a Diffusion Foundation Model for Bimanual Manipulation. arXiv:2410.07864 [cs.RO] https://arxiv.org/abs/2410.07864

[56] Yang Liu, Pengxiang Ding, Tengyue Jiang, Xudong Wang, Wenxuan Song, Minghui Lin, Han Zhao, Hongyin Zhang, Zifeng Zhuang, Wei Zhao, Siteng Huang, Jinkui Shi, and Donglin Wang. 2026. MMaDA-VLA: Large Diffusion Vision-Language-Action Model with Unified Multi-Modal Instruction and Generation. arXiv:2603.25406 [cs.RO] https://arxiv.org/abs/2603.25406

[57] Yuxiang Lu, Zhe Liu, Xianzhe Fan, Zhenya Yang, Jinghua Hou, Junyi Li, Kaixin Ding, and Hengshuang Zhao. 2026. FASTER: Rethinking Real-Time Flow VLAs. arXiv:2603.19199 [cs.RO] https://arxiv.org/abs/2603.19199

[58] Yunchao Ma, Yizhuang Zhou, Yunhuan Yang, Tiancai Wang, and Haoqiang Fan. 2025. Running VLAs at Real-time Speed. arXiv:2510.26742 [cs.RO] https://arxiv.org/abs/2510.26742

[59] Veynu Narasiman, Michael Shebanow, Chang Joo Lee, Rustam Miftakhutdinov, Onur Mutlu, and Yale N. Patt. 2011. Improving GPU performance via large warps and two-level warp scheduling. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 308–317.

[60] Kelvin K. W. Ng, Henri Maxime Demoulin, and Vincent Liu. 2023. Paella: Low-latency Model Serving with Software-defined GPU Scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 595–610. doi:10.1145/3600006.3613163

[61] NVIDIA, :, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. 2025. GR00T N1: An Open Foundation Model for Generalist Humanoid Robots. arXiv:2503.14734 [cs.RO] https://arxiv.org/abs/2503.14734

[62] NVIDIA. 2022. NVIDIA Ada GPU Architecture. https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf. Whitepaper v2.02.

[63] NVIDIA Corporation. [n. d.]. cuBLAS Library. https://docs.nvidia.com/cuda/cublas/. Accessed: 2026-04-22.

[64] NVIDIA Corporation. 2024. Multi-Process Service (MPS). https://docs.nvidia.com/deploy/mps/ Accessed: 2025.

[65] NVIDIA Corporation. 2024. NVIDIA Multi-Instance GPU User Guide. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/ Accessed: 2025.

[66] NVIDIA Corporation. 2024. *NVIDIA Nsight Compute – Kernel Profiling Guide*. https://docs.nvidia.com/nsight-compute/ProfilingGuide/

[67] NVIDIA Corporation. 2025. *CUDA C++ Programming Guide*. https://docs.nvidia.com/cuda/cuda-c-programming-guide/ Version 12.6.

[68] NVIDIA Corporation. 2025. *NVIDIA RTX Blackwell GPU Architecture: Built for Neural Rendering*. Whitepaper V1.1. NVIDIA Corporation. https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf Accessed: 2026-04-16.

[69] Sreepathi Pai, R. Govindarajan, and Matthew J. Thazhuthaveetil. 2014. Preemptive thread block scheduling with online structural runtime prediction for concurrent GPGPU kernels . In *2014 23rd International Conference on Parallel Architecture and Compilation (PACT)*. IEEE Computer Society, Los Alamitos, CA, USA, 483–484. doi:10.1145/2628071.2628117

[70] Sreepathi Pai, Matthew J. Thazhuthaveetil, and R. Govindarajan. 2013. Improving GPGPU concurrency with elastic kernels. *SIGARCH Comput. Archit. News* 41, 1 (March 2013), 407–418. doi:10.1145/2490301.2451160

[71] Jason Jong Kyu Park, Yongjun Park, and Scott Mahlke. 2015. Chimera: Collaborative Preemption for Multitasking on a Shared GPU. *SIGPLAN Not.* 50, 4 (March 2015), 593–606. doi:10.1145/2775054.2694346

[72] Jason Jong Kyu Park, Yongjun Park, and Scott Mahlke. 2017. Dynamic Resource Management for Efficient Utilization of Multitasking GPUs. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 527–540. doi:10.1145/3037697.3037707

[73] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. 2025. FAST: Efficient Action Tokenization for Vision-Language-Action Models. arXiv:2501.09747 [cs.RO] https://arxiv.org/abs/2501.09747

[74] Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, and Xuelong Li. 2025. SpatialVLA: Exploring Spatial Representations for Visual-Language-Action Model. arXiv:2501.15830 [cs.RO] https://arxiv.org/abs/2501.15830

[75] Timothy G. Rogers, Mike O'Connor, and Tor M. Aamodt. 2012. Cache-Conscious Wavefront Scheduling. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture* (Vancouver, B.C., CANADA) *(MICRO-45)*. IEEE Computer Society, USA, 72–83. doi:10.1109/MICRO.2012.16

[76] Timothy G. Rogers, Mike O'Connor, and Tor M. Aamodt. 2013. Divergence-aware warp scheduling. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture* (Davis, California) *(MICRO-46)*. Association for Computing Machinery, New York, NY, USA, 99–110. doi:10.1145/2540708.2540718

[77] Kohei Sendai, Maxime Alvarez, Tatsuya Matsushima, Yutaka Matsuo, and Yusuke Iwasawa. 2025. Leave No Observation Behind: Real-time Correction for VLA Action Chunks. arXiv:2509.23224 [cs.RO] https://arxiv.org/abs/2509.23224

[78] S.-Kazem Shekofteh, Hamid Noori, Mahmoud Naghibzadeh, Holger Froning, and Hadi Sadoghi Yazdi. 2020. cCUDA: Effective Co-Scheduling of Concurrent Kernels on GPUs . *IEEE Transactions on Parallel & Distributed Systems* 31, 04 (April 2020), 766–778. doi:10.1109/TPDS.2019.2944602

[79] 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. 2025. SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics. arXiv:2506.01844 [cs.LG] https://arxiv.org/abs/2506.01844

[80] Foteini Strati, Xianzhe Ma, and Ana Klimovic. 2024. Orion: Interference-aware, Fine-grained GPU Sharing for ML Applications. In *Proceedings of the Nineteenth European Conference on Computer Systems* (Athens, Greece) *(EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 1075–1092. doi:10.1145/3627703.3629578

[81] Xiaodan Serina Tan, Pavel Golikov, Nandita Vijaykumar, and Gennady Pekhimenko. 2023. GPUPool: A Holistic Approach to Fine-Grained GPU Sharing in the Cloud. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques* (Chicago, Illinois) *(PACT '22)*. Association for Computing Machinery, New York, NY, USA, 317–332. doi:10.1145/3559009.3569650

[82] Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. 2025. VLASH: Real-Time VLAs via Future-State-Aware Asynchronous Inference. arXiv:2512.01031 [cs.RO] https://arxiv.org/abs/2512.01031

[83] Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, Steven Bohez, Konstantinos Bousmalis, Anthony Brohan, Thomas Buschmann, Arunkumar Byravan, Serkan Cabi, Ken Caluwaerts, Federico Casarini, Oscar Chang, Jose Enrique Chen, Xi Chen, Hao-Tien Lewis Chiang, Krzysztof Choromanski, David D'Ambrosio, Sudeep Dasari, Todor Davchev, Coline Devin, Norman Di Palo, Tianli Ding, Adil Dostmohamed, Danny Driess, Yilun Du, Debidatta Dwibedi, Michael Elabd, Claudio Fantacci, Cody Fong, Erik Frey, Chuyuan Fu, Marissa Giustina, Keerthana Gopalakrishnan, Laura Graesser, Leonard Hasenclever, Nicolas Heess, Brandon Hernaez, Alexander Herzog, R. Alex Hofer, Jan Humplik, Atil Iscen, Mithun George Jacob, Deepali Jain, Ryan Julian, Dmitry Kalashnikov, M. Emre Karagozler, Stefani Karp, Chase Kew, Jerad Kirkland, Sean Kirmani, Yuheng Kuang, Thomas Lampe, Antoine Laurens, Isabel Leal, Alex X. Lee, Tsang-Wei Edward Lee, Jacky Liang, Yixin Lin, Sharath Maddineni, Anirudha Majumdar, Assaf Hurwitz Michaely, Robert Moreno, Michael Neunert, Francesco Nori, Carolina Parada, Emilio Parisotto, Peter Pastor, Acorn Pooley, Kanishka Rao, Krista Reymann, Dorsa Sadigh, Stefano Saliceti, Pannag Sanketi, Pierre Sermanet, Dhruv Shah, Mohit Sharma, Kathryn Shea, Charles Shu, Vikas Sindhwani, Sumeet Singh, Radu Soricut, Jost Tobias Springenberg, Rachel Sterneck, Razvan Surdulescu, Jie Tan, Jonathan Tompson, Vincent Vanhoucke, Jake Varley, Grace Vesom, Giulia Vezzani, Oriol Vinyals, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Fei Xia, Ted Xiao, Annie Xie, Jinyu Xie, Peng Xu, Sichun Xu, Ying Xu, Zhuo Xu, Yuxiang Yang, Rui Yao, Sergey Yaroshenko, Wenhao Yu, Wentao Yuan, Jingwei Zhang, Tingnan Zhang, Allan Zhou, and Yuxiang Zhou. 2025. Gemini Robotics: Bringing AI into the Physical World. arXiv:2503.20020 [cs.RO] https://arxiv.org/abs/2503.20020

[84] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. 2024. Octo: An Open-Source Generalist Robot Policy. arXiv:2405.12213 [cs.RO] https://arxiv.org/abs/2405.12213

[85] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL] https://arxiv.org/abs/2307.09288

[86] Devashree Tripathy, Amirali Abdolrashidi, Laxmi Narayan Bhuyan, Liang Zhou, and Daniel Wong. 2021. PAVER: Locality Graph-Based Thread Block Scheduling for GPUs. *ACM Trans. Archit. Code Optim.* 18, 3, Article 32 (June 2021), 26 pages. doi:10.1145/3451164

[87] Aditya Ukarande, Suryakant Patidar, and Ram Rangan. 2021. Locality-Aware CTA Scheduling for Gaming Applications. *ACM Trans. Archit. Code Optim.* 19, 1, Article 1 (Dec. 2021), 26 pages. doi:10.1145/3477497

[88] Nandita Vijaykumar, Gennady Pekhimenko, Adwait Jog, Abhishek Bhowmick, Rachata Ausavarungnirun, Chita Das, Mahmut Kandemir, Todd C. Mowry, and Onur Mutlu. 2015. A case for core-assisted bottleneck acceleration in GPUs: enabling flexible data compression with assist warps. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA '15)*. Association for Computing Machinery, New York, NY, USA, 41–53. doi:10.1145/2749469.2750399

[89] Oreste Villa, Mark Stephenson, David Nellans, and Stephen W. Keckler. 2019. NVBit: A Dynamic Binary Instrumentation Framework for NVIDIA GPUs. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO-52)*. Association for Computing Machinery, New York, NY, USA, 372–383. doi:10.1145/3352460.3358307

[90] Homer Walke, Kevin Black, Abraham Lee, Moo Jin Kim, Max Du, Chongyi Zheng, Tony Zhao, Philippe Hansen-Estruch, Quan Vuong, Andre He, Vivek Myers, Kuan Fang, Chelsea Finn, and Sergey Levine. 2024. BridgeData V2: A Dataset for Robot Learning at Scale. arXiv:2308.12952 [cs.RO] https://arxiv.org/abs/2308.12952

[91] Haoxuan Wang, Gengyu Zhang, Yan Yan, Ramana Rao Kompella, and Gaowen Liu. 2026. VLA Knows Its Limits. arXiv:2602.21445 [cs.RO] https://arxiv.org/abs/2602.21445

[92] Jin Wang, Norm Rubin, Albert Sidelnik, and Sudhakar Yalamanchili. 2016. LaPerm: locality aware scheduler for dynamic parallelism on GPUs. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea) *(ISCA '16)*. IEEE Press, 583–595. doi:10.1109/ISCA.2016.57

[93] Yuqi Wang, Xinghang Li, Wenxuan Wang, Junbo Zhang, Yingyan Li, Yuntao Chen, Xinlong Wang, and Zhaoxiang Zhang. 2025. Unified Vision-Language-Action Model. arXiv:2506.19850 [cs.CV] https://arxiv.org/abs/2506.19850

[94] Zhenning Wang, Jun Yang, Rami Melhem, Bruce Childers, Youtao Zhang, and Minyi Guo. 2016. Simultaneous Multikernel GPU: Multi-tasking throughput processors via fine-grained sharing. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 358–369. doi:10.1109/HPCA.2016.7446078

[95] Junjie Wen, Minjie Zhu, Jiaming Liu, Zhiyuan Liu, Yicun Yang, Linfeng Zhang, Shanghang Zhang, Yichen Zhu, and Yi Xu. 2025. dVLA: Diffusion Vision-Language-Action Model with Multimodal Chain-of-Thought. arXiv:2509.25681 [cs.RO] https://arxiv.org/abs/2509.25681

[96] 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. 2025. TinyVLA: Towards Fast, Data-Efficient Vision-Language-Action Models for Robotic Manipulation. arXiv:2409.12514 [cs.RO] https://arxiv.org/abs/2409.12514

[97] Yuqing Wen, Hebei Li, Kefan Gu, Yucheng Zhao, Tiancai Wang, and Xiaoyan Sun. 2025. LLaDA-VLA: Vision Language Diffusion Action Models. arXiv:2509.06932 [cs.RO] https://arxiv.org/abs/2509.06932

[98] Bo Wu, Guoyang Chen, Dong Li, Xipeng Shen, and Jeffrey Vetter. 2015. Enabling and Exploiting Flexible Task Assignment on GPU through SM-Centric Program Transformations. In *Proceedings of the 29th ACM on International Conference on Supercomputing* (Newport Beach, California, USA) *(ICS '15)*. Association for Computing Machinery, New York, NY, USA, 119–130. doi:10.1145/2751205.2751213

[99] Bo Wu, Xu Liu, Xiaobo Zhou, and Changjun Jiang. 2017. FLEP: Enabling Flexible and Efficient Preemption on GPUs. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 483–496. doi:10.1145/3037697.3037742

[100] Hao Wu, Weizhi Liu, Huanxin Lin, and Cho-Li Wang. 2020. A Model-Based Software Solution for Simultaneous Multiple Kernels on GPUs. *ACM Trans. Archit. Code Optim.* 17, 1, Article 7 (March 2020), 26 pages. doi:10.1145/3377138

[101] Haozhe Xie, Beichen Wen, Jiarui Zheng, Zhaoxi Chen, Fangzhou Hong, Haiwen Diao, and Ziwei Liu. 2026. DynamicVLA: A Vision-Language-Action Model for Dynamic Object Manipulation. arXiv:2601.22153 [cs.RO] https://arxiv.org/abs/2601.22153

[102] Qiumin Xu, Hyeran Jeon, Keunsoo Kim, Won Woo Ro, and Murali Annavaram. 2016. Warped-slicer: efficient intra-SM slicing through dynamic resource partitioning for GPU multiprogramming. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea) *(ISCA '16)*. IEEE Press, 230–242. doi:10.1109/ISCA.2016.29

[103] Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. 2025. VLA-Cache: Efficient Vision-Language-Action Manipulation via Adaptive Token Caching. arXiv:2502.02175 [cs.RO] https://arxiv.org/abs/2502.02175

[104] Zhaoshu Yu, Bo Wang, Pengpeng Zeng, Haonan Zhang, Ji Zhang, Zheng Wang, Lianli Gao, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. 2026. A Survey on Efficient Vision-Language-Action Models. arXiv:2510.24795 [cs.CV] https://arxiv.org/abs/2510.24795

[105] Dapeng Zhang, Jing Sun, Chenghui Hu, Xiaoyan Wu, Zhenlong Yuan, Rui Zhou, Fei Shen, and Qingguo Zhou. 2025. Pure Vision Language Action (VLA) Models: A Comprehensive Survey. arXiv:2509.19012 [cs.RO] https://arxiv.org/abs/2509.19012

[106] Han Zhao, Weihao Cui, Quan Chen, and Minyi Guo. 2023. ISPA: Exploiting Intra-SM Parallelism in GPUs via Fine-Grained Resource Management. *IEEE Trans. Comput.* 72, 5 (2023), 1473–1487. doi:10.1109/TC.2022.3214088

[107] Han Zhao, Weihao Cui, Quan Chen, Youtao Zhang, Yanchao Lu, Chao Li, Jingwen Leng, and Minyi Guo. 2022. Tacker: Tensor-CUDA Core Kernel Fusion for Improving the GPU Utilization while Ensuring QoS. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 800–813. doi:10.1109/HPCA53966.2022.00064

[108] Han Zhao, Weihao Cui, Quan Chen, Jieru Zhao, Jingwen Leng, and Minyi Guo. 2021. Exploiting Intra-SM Parallelism in GPUs via Persistent and Elastic Blocks. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*. 290–298. doi:10.1109/ICCD53106.2021.00054

[109] Han Zhao, Junxiao Deng, Weihao Cui, Quan Chen, Youtao Zhang, Deze Zeng, and Minyi Guo. 2025. Adaptive Kernel Fusion for Improving the GPU Utilization While Ensuring QoS. *IEEE Trans. Comput.* 74, 2 (Feb. 2025), 386–400. doi:10.1109/TC.2024.3477995

[110] Wei Zhao, Anand Jayarajan, and Gennady Pekhimenko. 2025. Tally: Non-Intrusive Performance Isolation for Concurrent Deep Learning Workloads. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '25)*. 1052–1068. doi:10.1145/3669940.3707282

[111] Xia Zhao, Zhiying Wang, and Lieven Eeckhout. 2018. Classification-Driven Search for Effective SM Partitioning in Multitasking GPUs. In *Proceedings of the 2018 International Conference on Supercomputing* (Beijing, China) *(ICS '18)*. Association for Computing Machinery, New York, NY, USA, 65–75. doi:10.1145/3205289.3205311

[112] Teqiang Zou, Hongliang Zeng, Yuxuan Nong, Yifan Li, Kehui Liu, Haotian Yang, Xinyang Ling, Xin Li, and Lianyang Ma. 2025. Asynchronous Fast-Slow Vision-Language-Action Policies for Whole-Body Robotic Manipulation. arXiv:2512.20188 [cs.RO] https://arxiv.org/abs/2512.20188