Towards Deterministic End-to-end Latency for Medical AI Systems in NVIDIA Holoscan

Soham Sinha
NVIDIA
Santa Clara, CA, USA
sohams@nvidia.com

Shekhar Dwivedi
NVIDIA
Santa Clara, CA, USA
shekhard@nvidia.com

Mahdi Azizian
NVIDIA
Santa Clara, CA, USA
mazizian@nvidia.com

***Abstract*—The introduction of AI and ML technologies into medical devices has revolutionized healthcare diagnostics and treatments. Medical device manufacturers are keen to maximize the advantages afforded by AI and ML by consolidating multiple applications onto a single platform. However, concurrent execution of several AI applications, each with its own visualization components, leads to unpredictable end-to-end latency, primarily due to GPU resource contentions. To mitigate this, manufacturers typically deploy separate workstations for distinct AI applications, thereby increasing financial, energy, and maintenance costs. This paper addresses these challenges within the context of NVIDIA's Holoscan platform, a real-time AI system for streaming sensor data and images. We propose a system design optimized for heterogeneous GPU workloads, encompassing both compute and graphics tasks. Our design leverages CUDA MPS for spatial partitioning of compute workloads and isolates compute and graphics processing onto separate GPUs. We demonstrate significant performance improvements across various end-to-end latency determinism metrics through empirical evaluation with real-world Holoscan medical device applications. For instance, the proposed design reduces maximum latency by 21–30% and improves latency distribution flatness by 17–25% for up to five concurrent endoscopy tool tracking AI applications, compared to a single-GPU baseline. Against a default multi-GPU setup, our optimizations decrease maximum latency by 35% for up to six concurrent applications by improving GPU utilization by 42%. This paper provides clear design insights for AI applications in the edge-computing domain including medical systems, where performance predictability of concurrent and heterogeneous GPU workloads is a critical requirement.**

***Index Terms*—Concurrent and Heterogeneous GPU Workloads, Predictable End-to-end Latency, Medical AI Devices**

I. Introduction

The integration of Artificial Intelligence (AI) and Machine Learning (ML) technologies into the medical devices represents a paradigm shift in the healthcare and diagnostics industry. AI and ML applications have gained great traction in recent years by enhancing medical procedures, ranging from disease diagnostics to surgical interventions. They empower automated real-time monitoring and provide valuable insights to medical professionals and physicians, leading to early diagnosis of diseases and reduced patient recovery times, among other benefits [3], [9], [11]. These medical AI applications need both accelerated AI workload processing and optimized graphical rendering, enabled by modern GPUs, which has given rise to innovative platforms like the NVIDIA Holoscan.

Holoscan is NVIDIA's scalable edge-computing platform for AI-enabled sensor processing. It empowers medical device manufacturers with the ability to create real-time pipelines for AI-based analysis and visualization of streaming data and medical images [34]. Holoscan consists of optimized libraries for I/O, data processing, inference and graphical rendering on the NVIDIA GPUs. It provides a modular and intuitive data-flow programming model and is compatible with both ARM- and x86-based hardware equipped with GPUs, offering medical device vendors the freedom to select their architecture.

As the field of AI continues its rapid evolution, developers aspire to harness the potential of NVIDIA Holoscan SDK for integrating *multiple* AI applications with visualization capabilities into their systems. This is especially true for edge-computing domains like medical devices, as compute workloads (enabled by CUDA) in this domain cannot leverage the full potential of massive parallelism offered by today's GPUs [16], [55]. Therefore, a manufacturer may seek to concurrently run multiple Holoscan AI applications [23], [40], aiming to enhance the efficiency of medical procedures. Real-world constraints such as space limitations, power consumption, financial cost considerations, maintenance, and regulatory requirements also necessitate utilizing as minimized set of hardware as possible, like a single workstation with GPU(s).

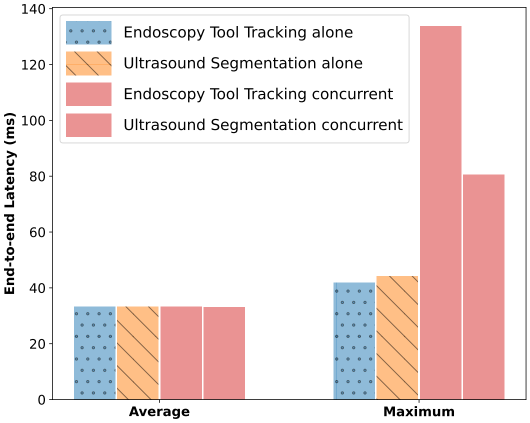


Fig. 1: Average and Maximum End-to-end Latency on an x86 Workstation with an RTX A4000 GPU

These constraints, however, often pose challenges to performance predictability [50], since heterogeneous nature of the simultaneous AI-based compute and graphical rendering workloads create resource contention on a GPU [32]. For example, Figure 1 shows the average and maximum end-to-end (E2E) latency when running endoscopy tool tracking and ultrasound segmentation Holoscan applications alone and concurrently on an x86 workstation with an RTX A4000 GPU. Simultaneous execution of multiple AI and visualization workloads leads to increased maximum latency. To avoid this issue, device manufacturers employ distinct workstations for separate AI applications, increasing the economic burden on themselves, hospitals and ultimately patients.

This paper addresses the critical challenge of deterministic end-to-end latency in medical AI systems with concurrent and heterogeneous GPU workloads. The primary contributions of the paper are outlined as follows:

- We propose a novel design approach that combines CUDA MPS (**M**ulti-**p**rocess **S**ervice) [37] for spatial partitioning between compute workloads and a load-balancing technique to isolate compute (CUDA kernel) and graphics tasks onto distinct GPUs. Additionally, we use an admission control policy to prevent SM-oversubscription by concurrent compute tasks. Our pragmatic design is straightforward to implement and minimizes heavy context-switch overheads [27], mitigating the resource contention within a GPU for medical AI workloads.
- Empirical evaluation using a set of E2E latency determinism metrics reveals substantial performance improvement with our design: maximum latency, standard deviation, latency distribution tail, and flatness are reduced by 21–30%, 17–24%, 21–47%, and 17–25%, respectively, for up to five concurrent endoscopy tool tracking AI applications, against a traditionally used single-GPU workstation. Compared to a baseline multi-GPU system, our design decreases the maximum latency by 35% for up to six concurrent applications by improving GPU utilization by 42%.

Our design and results hold the promise of facilitating more AI deployments in medical systems with more safety and predictability, while also providing design suggestions for other embedded and edge-computing application pipelines seeking to leverage concurrent and heterogeneous GPU workloads.

Next section provides an overview of the NVIDIA Holoscan SDK, followed by its usage in medical devices in Section III. Our system design approach for predictable performance is detailed in Section IV. Section V describes our experimental setup for performance evaluation, followed by a comprehensive analysis of the experimental results using real-world, industry-grade Holoscan medical applications in Section VI. Finally, we discuss the related work in Section VII and conclude the paper in Section VIII.

II. NVIDIA Holoscan SDK

NVIDIA Holoscan SDK is an open-source AI sensor processing platform designed for low-latency real-time steaming data and images [34]. It includes optimized libraries for data processing and AI, as well as core software services for various applications ranging from embedded systems to edge computing platforms. Initially designed for medical devices, the SDK has evolved to be domain-agnostic, capable of serving multiple sectors like High-performance Computing at the Edge, Industrial Automation and Space Computing. It offers both C++ and Python APIs streamlining application prototyping and production deployment. It also includes built-in capabilities for functionalities such as I/O, ML inference, processing, and visualization, optimized for NVIDIA GPUs.

A typical Holoscan application acquires and processes streaming data and images, and either controls an actuator or renders the processed output to a screen. An application consists of a number of *fragments* where each fragment is a connected graph of *operators*. Each fragment is assigned to a physical or virtual machine. For this paper, we only focus on single-fragment applications in this work.

An operator is a core execution unit for a specific task. Operators interact with each other through *Ports* which are data entry and exit points for an operator. An operator ingests streaming data at an input port and transmits data via an output port. The NVIDIA Holoscan SDK provides a number of built-in operators, such as *FormatConverterOp* for converting data format, *InferenceOp* for AI inferences [35]. Application developers can also write their own operators by implementing initialization (`start()`), processing (`compute()`) and completion (`stop()`) functions of an operator. A single-fragment Holoscan application is created by connecting multiple operators in a graph to handle streaming AI workloads.

A Holoscan application is executed by topologically sorting the operators in a connected graph-fragment on a single thread.[1] It ensures that the data processing of previous operators is completed before the next ones are executed. Each Holoscan operator has a set of *Conditions* for scheduling decisions. Detailed discussions on these topics are not relevant to this work. Next, we describe two key components of the Holoscan SDK, both of which utilize the GPU - inference and visualization modules - before explaining the end-to-end latency measurements with the data flow tracking component.

A. Inference

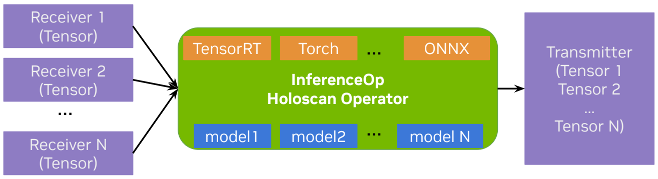


Fig. 2: InferenceOp Operator in Holoscan SDK

The inference module in the SDK provides APIs for designing inference-based AI applications. The *InferenceOp* utilizes this module to load an AI model at runtime. It can also be configured with a range of parameters, including specifying inference backend, particular GPU device. It supports a variety of ML library backends, including TensorRT, Torch and ONNX runtime. The operator receives tensors into its input ports, performs inferences on the input tensors based on a given model for an input port, and finally emits output data

[1] Multithreaded execution in Holoscan is very recently enabled.

as a processed tensor (see Figure 2). Multiple inference workloads are concurrently launched in parallel CUDA streams on separate CPU threads.

B. Visualization

Holoviz is the visualization component in the Holoscan SDK that allows developers to seamlessly create and display advanced visual effects with minimal effort. It can combine real-time streams of frames in multiple layers including image stream, segmentation mask, geometry, text and GUI layers. It also supports depth map and volumetric 3d rendering, pertinent for robotic surgery and other advanced medical procedures. Holoviz leverages the Vulkan graphics API [46] to optimize rendering performance and adopts the immediate mode design pattern [22] to accelerate the rapid visualization updates in medical devices. *HolovizOp* operator can be used to connect and transfer tensors and video buffers from other operators for visualization. Since it utilizes the default CUDA stream for compute operations, resource contention between inference and other CUDA compute operations may occur in absence of any isolation between graphics and compute contexts.

C. Data Flow Tracking

Inspired by the ideas of Information Flow Tracking [15], Holoscan implements data flow tracking within its fragment-graph. With this feature, each incoming and outgoing message in an operator is timestamped. The end-to-end latency of individual messages is calculated using these fine-grained timestamps. End-to-end latency [49] is the time interval between a message's arrival at a root operator of the fragment-graph via camera or other sensory input and its departure from a leaf operator for either rendering or actuation. We evaluate Holoscan applications through a set of performance determinism metrics based on the observed end-to-end latencies using the data flow tracking component.

III. Medical Device Application Use-cases

NVIDIA Holoscan SDK is useful for different edge-computing domains, including industrial automation and space computing, but medical AI is one of its primary target areas. There are several available Holoscan medical AI applications [33]. This section describes two example applications to provide a high-level overview of their functionalities.

A. Endoscopy Tool Tracking Application

The Endoscopy Tool Tracking application is designed to detect and annotate endoscopic instruments within a live stream of video frames. It ingests a video stream input either via a *Replayer* or a custom video card [2] operator. The video stream is converted into a tensor and handed over to a *FormatConverter* operator which changes the datatype of the tensor elements as part of pre-processing. The tensor is then fed into an *LSTMTensortRTInference* operator for inference via TensorRT [38]. *LSTMTensortRTInference* loads a pre-trained inference model on a GPU at runtime. The output tensor containing the inference result is then processed by a *ToolTrackingPostProcessorOp* operator for endoscopic tool segmentation. Finally, a *Holoviz* operator overlays the segmented tools and their text labels onto the original input video frame for visualization. Figure 3 summarizes the application data flow and its interaction with GPU(s).

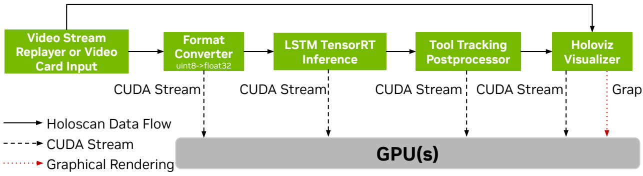


Fig. 3: Endoscopy Tool Tracking Application

The application is implemented in both C++ and Python. A snippet of the C++ code is provided below. It creates an `App` class derived from `Application` class. The `compose()` member function instantiates the required Holoscan operators. The operator are configured with the `from_config` function that reads a YAML configuration file. CUDA stream for an operator is allocated with `CudaStreamPool` class which manages data-transfer and compute operations on the GPU(s). A CUDA stream is initialized once for `FormatConverterOp` and reused by subsequent operators in the pipeline.

```
class App : public holoscan::Application {
public:
...
void compose() override {
...
auto source =
  make_operator<ops::VideoStreamReplayerOp>(
  "replayer", from_config("replayer"));

auto format_converter =
  make_operator<ops::FormatConverterOp>(
  "format_converter",...,
  Arg("cuda_stream_pool") = cuda_stream_pool);

auto lstm_inferer =
  make_operator<ops::LSTMTensorRTInferenceOp>(
  "lstm_inferer",...);

auto tool_tracking_postprocessor =
  make_operator<ops::ToolTrackingPostprocessorOp>(
  "tool_tracking_postprocessor");

auto visualizer =
  make_operator<ops::HolovizOp>("holoviz",...);
...
}
}
```

After initialization, operators are connected in `compose` to define the data flow using the `add_flow` function. The first two parameters specify source and destination operators, while the third parameter defines the port map connecting them.

```
// Flow definition
add_flow(source, format_converter,
  {{"output", "source_video"}});
add_flow(format_converter, lstm_inferer);
add_flow(lstm_inferer,
  tool_tracking_postprocessor,{{"tensor", "in"}});
add_flow(tool_tracking_postprocessor, visualizer,
  {{"out", "receivers"}});
```

B. Multi AI Ultrasound Application

The Multi-AI Ultrasound application utilizes three machine learning models from iCardio [21] concurrently for real-time cardiovascular ultrasound diagnostics. It uses four instances of *FormatConverterOp*, one for each model and an additional one for visualization. The *InferenceOp* facilitates parallel inferences across multiple CUDA streams, each spawned from individual CPU threads. The inference models perform anatomical heart component measurements and classification for various cardiac views and aortic stenosis anomalies. *InferenceProcessorOp* executes post-processing operations, while *HolovizOp* renders the aggregated overlaid layers along with the original video frames. Unlike the endoscopy example, this application consists of multiple paths in its graph of operators.

IV. Deterministic Design Approach

In this section, we describe our design to enhance performance determinism within the NVIDIA Holoscan platform when applied to medical AI applications. These techniques primarily focus on mitigating resource contention arising from concurrent and heterogeneous GPU workloads. Our design spatially isolates GPU workloads including graphics on dedicated SMs. Our design and optimization techniques, although studied in the context of medical AI, are generally applicable to heterogeneous GPU workloads.

A. CUDA MPS Partition

CUDA MPS [37] is NVIDIA's technology designed for concurrent execution of multiple GPU-accelerated applications on GPUs. It leverages the Hyper-Q capability [7] present in NVIDIA GPUs, enabling the concurrent processing of multiple CUDA kernels on the same GPU. This functionality helps in embedded and edge computing applications like medical devices software, where GPU workloads predominantly involve inference and lightweight data processing tasks and do not saturate the computational capabilities of today's GPUs like the RTX A4000 and A6000.

Traditionally, a CUDA program that wants to run a GPU workload begins by creating a CUDA context or using the default primary CUDA context, encapsulating the essential hardware resources. Multiple GPU workloads can be organized into CUDA streams in a single CUDA context. Furthermore, multiple processes can create separate CUDA contexts but depend on the GPU hardware scheduler for time-sliced GPU workload execution. The time-multiplexed execution of CUDA streams and contexts on the GPU and dependence on the GPU hardware scheduler for SM allocation may increase throughput at the cost of lower performance determinism.

CUDA MPS aims to better utilize the GPU for concurrently running GPU workloads with a client-server model. An MPS server is a single process that manages the GPU resources and coordinates the execution of multiple MPS clients' CUDA tasks. The CUDA kernel launches are intercepted by the MPS server to be launched on behalf of the client. This allows the MPS server to optimize GPU resource allocation and run concurrent CUDA kernels in separate CUDA contexts.

Before the NVIDIA Volta GPU architecture, the MPS server was responsible for managing hardware resources for a single CUDA context. All the client CUDA contexts from multiple processes would route their CUDA workloads through the MPS server. Since the Volta architecture, MPS client CUDA contexts directly manage most hardware resources. The MPS server continues to play a role in mediating the shared resources. The communication between MPS clients and the server remains transparent to applications, facilitated through the usual CUDA API. Existing CUDA programs do not require any modification to utilize CUDA MPS features.

We utilize CUDA MPS functionalities to establish isolated partitions resembling GPU *sandboxes* consisting of exclusive SMs and device memory. To ensure deterministic application performance, a CUDA program requires, at least, exclusive access to its required number of SMs and memory, so that concurrent CUDA tasks do not interfere with or hamper each other's performance. CUDA MPS partitioning enables it.

1) MPS Partition Creation: By default, all SMs in a GPU are available to any MPS client. This may lead to potential latency variability as concurrent CUDA kernels may contend for the same set of SMs. CUDA MPS provides a mechanism for constraining the allocation of GPU resources to each MPS client process, with the `CUDA_MPS_ACTIVE_THREAD_PERCENTAGE` and `CUDA_MPS_PINNED_DEVICE_MEM_LIMIT` environment variables to, respectively, limit the number of SMs and device memory. We employ this mechanism to create isolated GPU partitions for each MPS client. The required number of SMs and device memory are determined through offline profiling [53] of a Holoscan application. Future work will consider cache, memory bus, and other microarchitecture partitioning.

2) Admission Control: To avoid GPU overload and contention possibility caused by an excessive number of MPS clients, we implement an admission control that denies launching CUDA workload for an MPS client if its SM requirements surpass the currently available SMs. As CUDA MPS does not support strong SM affinity [6], our approach prevents resource sharing, reducing further interference between applications.

TABLE I: Feature Comparison between MIG, vGPU and CUDA MPS for Deterministic Performance against Heterogeneous (Compute and Graphics) GPU Workloads

Feature	MIG	vGPU	CUDA MPS
Dedicated SM Allocation (No Time-slicing)	Yes	No	Yes
Custom SM Allocation	No	-	Yes
Graphics Support	No	Yes	No
Supported in RTX A4000 or A6000 (or other ProViz GPUs)	No	Yes	Yes

B. Hardware Isolation Between Compute and Graphics

GPU workloads in medical AI systems are often heterogeneous, comprising both graphics rendering and compute tasks. However, multiplexing these heterogeneous workloads on a single GPU can lead to nondeterministic performance. The

NVIDIA GPUs incorporate an optimized and coherently structured graphics pipeline that cannot be externally preempted or controlled [27]. As medical equipment applications consider both graphics and compute equally important, it is critical to ensure timing-predictable rendering and compute task output. Technologies like Multi-instance GPU (MIG), virtual GPU (vGPU) [39], and CUDA MPS are limited in supporting predictable performance against heterogeneous GPU workloads, and they are summarized in Table I. Although MIG provides strong hardware isolation, it does not support graphics. vGPU in time-slicing mode cannot reserve exclusive SMs. We are using CUDA MPS partitioning with a maximum cap on the number of dedicated SMs, but it does not handle graphics.

It is clear that existing solutions do not isolate graphics and compute on dedicated SMs. To address this, we divide the graphics and compute workloads to separate physical GPUs. Although this introduces the overhead of transferring data from the compute GPU memory to the graphics GPU memory, NVIDIA's Unified Virtual Addressing [45] is leveraged to initiate zero-copy transfer with minimal overhead. Additionally, the data-transfer cost amortizes with more workload because of overlapped data-transfer and compute [36], [45]. The experiments demonstrate that isolating two types of GPU workloads to distinct GPUs is an effective mitigation of resource contention in this case. Generally, using a more compute-capacity GPU for compute and a lesser one for graphics is advised since compute workloads are more SM-intensive. A compute GPU is used for executing the CUDA kernels including data pre- and post-processing, AI inferences. The graphics GPU is mainly used for rendering, which leverages Vulkan APIs via *Holoviz*.

C. CPU Affinity

Medical device systems typically pre-determine the number and types of applications for deployment. Such foresight allows for pinning each application to specific CPU cores to reduce Linux scheduling overhead, as advised for safety-critical systems [10]. We utilize Linux CPU affinity APIs (`sched_setaffinity`), and utilities (`taskset`, `cset`, `isolcpus`) for this purpose.

D. Design Summary

Figure 4 summarizes our system design proposal for concurrent Holoscan applications. Every Holoscan application is launched as a separate process, that could be pinned to a set of CPU cores. Their GPU workloads are divided into distinct GPUs based on the workload type. Their compute workloads are launched to a number of dedicated SMs on a compute GPU, enabled by CUDA MPS partitioning, while rendering is directed to a graphics GPU. The resultant workload partition is elaborated as an addendum in Appendix (Figure 12).

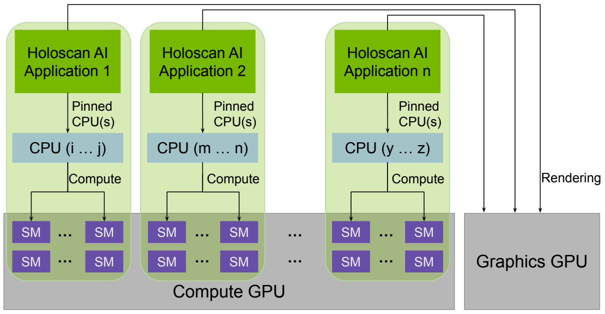


Fig. 4: System Design for Concurrent AI Applications

E. Scope and Limitations

We discuss our solutions in the context of NVIDIA GPUs and the NVIDIA Holoscan SDK for medical AI devices because of their widespread adoption, accessibility, and long-term software support. However, these techniques are also applicable to other GPUs [41] and heterogeneous workloads. Furthermore, this work mainly focuses on compute resource (SM) isolation. Future work will explore mitigation for cache, memory bus, and other shared resource contention [5].

V. Experimental Setup

Holoscan officially supports ARM-based NVIDIA IGX Orin devkit with RTX A6000 GPU, and x86 workstations with selected discrete GPUs. Typically, medical device manufacturers use a high-performance x86 workstation with a discrete GPU. For this paper, we used an Intel i7-7920X 2.90 GHz CPU, equipped with GPUs: RTX A4000 (48 SMs, 6144 CUDA cores, 19.2 single-precision (SP) TFLOPS) and A6000 (84 SMs, 10752 CUDA cores, 38.7 SP TFLOPS). Although the emerging IGX Orin platform has gained significant traction for medical AI [1], [29], it does not support CUDA MPS for now [37]. Future work will consider it with inter-operable iGPU and dGPU for multi-GPU and MPS configurations.

A. End-to-end Latency Determinism Metrics

E2E latency, a well-known performance indicator in real-time cyber-physical systems, is the time-interval between a message's arrival to and departure from an application. In Holoscan, a message may traverse through multiple paths in the graph of operators. We consider every path's E2E latency.

To gauge the overall performance quality of an application, we examine the average E2E latency, which also indicates the frame-rate for visualization. In contexts where safety, reliability, and certification concerns are paramount, the maximum E2E latency is a primary metric, used also for latency determinism. Although the industry, especially the medical devices sector, rarely uses any other metrics to quantify latency determinism [28], [30], we use three more key metrics to assess it.

1) **Standard Deviation:** It measures the spread of latencies around the average value. A lower standard deviation indicates concentrated latencies around the mean latency. However, it does not provide information about the overall latency distribution and outliers like the maximum latency.
2) **Latency Distribution Tail:** It is the difference between the 95th and 100th percentile values of the observed E2E latencies. It indicates how widely the outlier latencies are distributed at the higher end of the distribution. A smaller tail means that the outlier latencies are less stretched out near the maximum latency, indicating greater predictability.

3) **Latency Distribution Flatness:** It is defined as the difference between the 10th and 90th percentile values of the observed end-to-end latencies. A smaller flatness indicates more concentrated and deterministic most-observed latencies.

B. Applications

We carried out most of our experiments with the Endoscopy Tool Tracking Holoscan application [33]. It is representative of a typical medical device AI application. For experiments, we use a Holoscan operator that replays a saved video on the disk. However, NVIDIA Holoscan also supports live video feed from high-fidelity cameras directly to the GPU memory through DMA [2], [14] which minimizes I/O overhead and facilitates faster data-processing in a GPU. Accelerated I/O solutions such as RDMA improves the performance further, but these topics are out of scope of this paper.

We experimented with two additional applications. The Ultrasound Segmentation pipeline [33] is similar to the Endoscopy application and performs automatic segmentation of the spine from a trained AI model for scoliosis visualization and measurement. The Multi-AI Ultrasound application [33], described in Section III-B, is more complicated than the others as it launches three AI inferences in parallel CUDA streams.

For most experiments, we run the same applications from different processes to emulate a scenario of multiple Holoscan applications running on the same system. We experiment with up to five concurrent applications, sufficient for the needs of medical AI devices on a single platform.

VI. Result Analysis

This section presents an experimental analysis of our system design. For every experiment, an application is executed for a period of 1000 messages or approximately 30 seconds, where the first and last 10 messages are discarded in every experiment as warm-up and cool-down periods. Each experiment is repeated 10 times. We analyze our results using end-to-end (E2E) latency and GPU utilization metrics. Frame-rate for graphics rendering is inherently captured by the average E2E latency. Our analysis focuses on the latency distribution and dispersion, based on the metrics discussed in Section V-A. The experimental artifacts will be made available later [33].

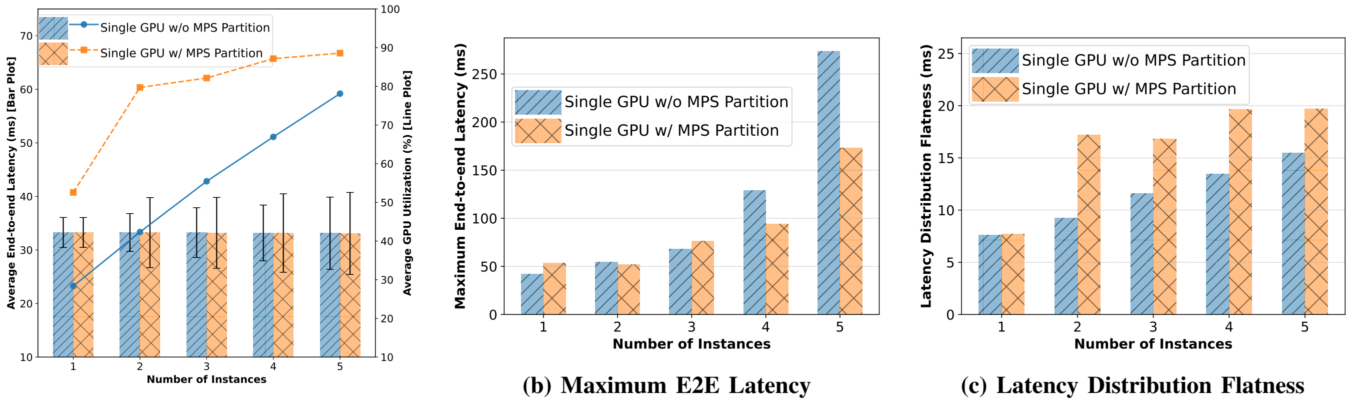
A. Performance Result with CUDA MPS Partitions

We first analyze the performance results with and without CUDA MPS partitioning on a single RTX A4000 GPU. For the experiments, we configured CUDA MPS partitions to limit 20% of the GPU threads and 2GB of device pinned memory per client. These configurations were determined by profiling the application. Our experiments included up to five MPS clients as Holoscan applications, ensuring that allocated threads and memory did not exceed the GPU's limits.

Figure 5a reveals that average latency (left Y-axis, bar plot) remains consistent for up to 5 concurrent endoscopy tool tracking instances, indicating a non-prohibitive amortized frame-rate. However, the standard deviation, represented as error caps, rises with instance count. CUDA MPS yields similar or slightly elevated standard deviations compared to a non-MPS setup, as it does not handle contention between compute and graphics contexts. Figure 5a also captures GPU utilization (right Y-axis, line plot) which is better with CUDA MPS than with the non-MPS setup, due to MPS' reservation on the number of GPU threads per client. More GPU utilization is adequately leveraged to improve the maximum E2E latency by up to a 37% (Figures 5b), and latency distribution tail by up to 45% for five instances (Figure 15 in the Appendix). The benefits of MPS partitioning are amplified with more instances, as more resource contention deteriorates the performance predictability of a non-MPS single GPU setup. Figure 5c shows no improvement in distribution flatness with CUDA MPS, due to software overhead and context-switching costs between graphics and compute contexts.

B. Performance Result with Compute and Graphics Isolation

This section compares performance between two setups: one where compute and graphics workloads are segregated onto two distinct GPUs (2x A4000, and A4000+A6000), and another where all the tasks are handled by a single GPU (A4000 or A6000), both on a single x86 workstation. Figure 6a shows that the average E2E latency stays consistent across single and multi-GPU configurations, enabled by average frame-rate of the source video. However, the standard deviation increases with more instances in single-GPU setups but remains stable in

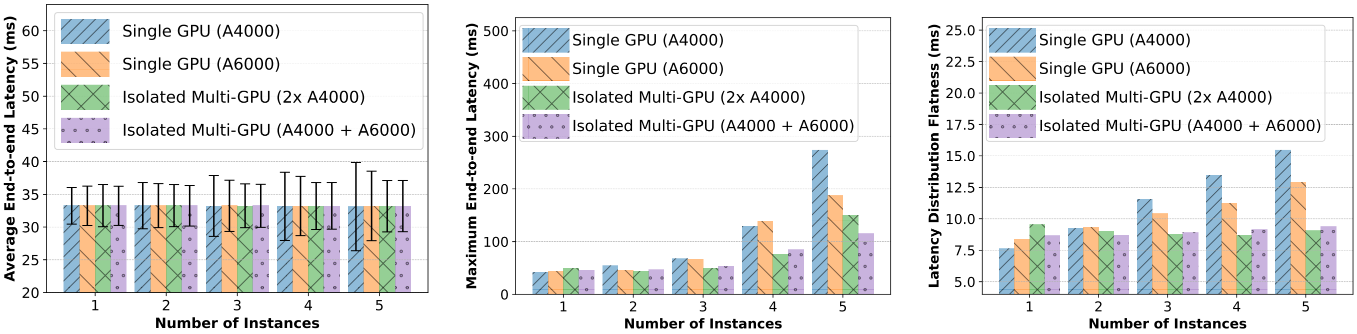


(a) Average Latency and GPU Utilization

(b) Maximum E2E Latency

(c) Latency Distribution Flatness

Fig. 5: Performance on Single GPU without vs. with CUDA MPS Partitions for Endoscopy Tool Tracking



(a) Average E2E Latency **(b) Maximum E2E Latency** **(c) Latency Distribution Flatness**

Fig. 6: Performance on Single GPU vs. Compute and Graphics Isolated on Multi-GPUs for Endoscopy Tool Tracking

workload-isolated multi-GPU arrangements, implying reduced context-switch costs and greater predictability.

Figure 6b demonstrates 16–24% reduction in maximum E2E latency with our design of isolated multi-GPU configurations. Specifically, *the dual A4000 configuration, using our workload partitioning design, achieves 16% lower maximum latency than a single A6000 setup, offering 7% potential energy saving and over 50% cost reduction*, based on specifications and prices listed on CDW and Amazon.com, and making medical AI applications more affordable and sustainable.

Although a 10% latency penalty is incurred for 1 instance in the multi-GPU setups due to data-transfer overheads between the GPUs, this cost is offset with more instances [36], [45]. Moreover, latency distribution tail (Figure 16 in Appendix) and flatness in Figure 6c are improved by 30% and 17%, respectively, in multi-GPU setups. The result demonstrates that workload isolation in multi-GPUs overcomes the limitations of MPS partitioning by segregating the graphics and compute contexts. Section VI-D validates that these performance gains indeed stem from workload isolation, not merely from increased compute capacity with multiple GPUs.

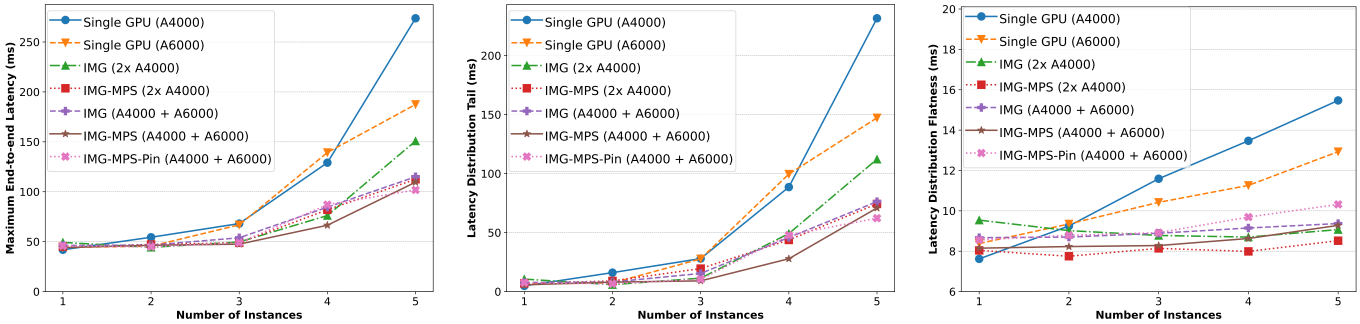
C. Performance Result with All Optimizations

In the subsequent experiments, we evaluate all configurations from our proposed design approaches:

1) **Single GPU**: Both compute and graphics workloads are executed on a single GPU. This is the baseline, as manufacturers typically use such configurations.
2) **Isolated Multi-GPU (IMG)**: Compute and graphics tasks are isolated onto distinct physical GPUs.
3) **Isolated Multi-GPU + MPS Partition (IMG-MPS)**: Similar to IMG but MPS partitioning applied on top of it.
4) **Isolated Multi-GPU + MPS Partition + CPU Pinning (IMG-MPS-Pin)**: Extends IMG-MPS by pinning each application instance to a specific CPU core.

Average latency is similar across different configurations, like before, but the standard deviation reduces 12–24% with IMG and IMG-MPS configurations (Figure 17 in Appendix) compared to baselines. IMG and IMG-MPS configurations overall perform better across all the determinism metrics, as illustrated in Figure 7. Importantly, IMG, IMG-MPS, and IMG-MPS-Pin achieve maximum E2E latencies below 50 ms for up to three concurrent endoscopy applications. This implies that *a single x86 workstation with two GPUs like A4000 and A6000 can run up to three such applications without exceeding 50 ms E2E latency*, obviating the need for dedicated workstation per AI application and reducing cost, power consumption, and physical footprint in healthcare facilities and hospitals without compromising performance predictability and safety.

IMG-MPS improves the maximum E2E latency by 21-30%, standard deviation by 17-24%, latency distribution tail by 21–47% and flatness by 17-25%, on average, compared to single GPU setups. IMG and IMG-MPS with dual A4000s offer better performance than a single A6000, while improving energy usage and reducing cost by over 50%. IMG and IMG-MPS yield comparable performance across different metrics,



(a) Maximum End-to-end Latency **(b) Latency Distribution Tail** **(c) Latency Distribution Flatness**

Fig. 7: Performance Comparison between All Optimizations for Endoscopy Tool Tracking Application

underscoring the benefits of isolating compute and graphics tasks on distinct GPUs. Nevertheless, IMG-MPS surpasses IMG in reducing the maximum latency, latency distribution tail, and flatness by an average of 7%, 7%, and 8%, respectively. This performance gain indicates the importance of MPS partitioning, even with workload isolation on separate GPUs.

The latency distribution tail in Figure 7b is below 75ms for IMG-MPS and IMG-MPS-Pin configurations, demonstrating the mitigation of outlier risks. Furthermore, latency distribution flatness is constrained to less than 10 ms for IMG-MPS (see Figure 7c), indicating consistent most-observed latencies. These metrics collectively exhibit a holistically greater predictable E2E latency for its full distribution of observed values.

More instances raise the maximum latency for IMG-MPS, although at a lower rate, due to shared resource contention [54], whose mitigation is subject to future work [5]. Single GPU CUDA MPS partitioning results are not shown here (see Section VI-A), as it is less effective than IMG, IMG-MPS and IMG-MPS-Pin, due to overhead from multiplexing graphics context with multiple compute contexts on a GPU outweighing the benefits of isolating compute workloads on a single GPU. Pairing MPS partitioning with multi-GPU workload isolation results in more performance benefits.

Impact of CPU Affinity: Figure 7 reveals marginal advantages of CPU pinning. While some metrics, like maximum E2E latency and latency distribution tail, improve in certain cases, others remain static. Given the GPU-intensive nature of medical AI workloads, CPU affinity yields little performance gains. Although certification and safety guidelines often recommend CPU affinity to mitigate interference, its efficacy in improving determinism in medical AI contexts is a bit limited. Nevertheless, CPU affinity does not impair performance and is usable for regulatory purposes.

D. Do More GPUs Lead to Better Determinism?

Earlier sections showed consistent improvement in performance determinism with multi-GPU configurations. In this section, we demonstrate that the computational power of multi-GPUs is not the driver behind this performance gain.

We conducted an experiment using two RTX A4000 GPUs in two configurations: **1) Two A4000s (C+G):** Both GPUs are used for compute (C) and graphics (G) tasks, and each GPU is connected to a display monitor, **2) our design - A4000 (C) + A4000(G) + MPS Partition:** One A4000 is for compute tasks, the other is for graphics, with MPS partitioning enabled at a 15% thread percentage, making sure that the sum of maximum active thread percentage does not exceed 100% for up to 6 concurrent applications. *Figure 8 reveals a 35% average reduction in maximum latency and a 42% increase in GPU utilization with our optimized configuration.* These results indicate that our load-balancing design is instrumental in achieving predictable latency in a multi-GPU configuration.

Another experiment evaluated the A4000 and A6000 GPUs in four different configurations: **1)** one A4000 for both compute (C) and graphics (G), **2)** one A6000 for both C and G, and **our design**: **3)** A4000 for G and A6000 for C, **4)** A6000 for G and A4000 for G. In Figure 9, while our optimized configurations 3 and 4 do outperform the single-GPU setup like before, the key observation is negligible performance difference between the two of our configurations. This highlights the effectiveness of our system design, irrespective of the individual compute capacities of the GPUs.

E. Other Applications

Other Holoscan applications, such as ultrasound bone scoliosis segmentation and multi-AI cardiac ultrasound [33], also show performance improvements with the IMG-MPS and IMG-MPS-Pin configurations. The ultrasound segmentation performs spinal segmentation with a TensorRT inference model. Its data flow pipeline is similar to the endoscopy application, leading to comparable results (see Figure 10).

Figure 11a for the multi-AI ultrasound shows that the average E2E latency increases with more instances, unlike the other applications. As described in Section III-B, this application executes three parallel inference models, resulting in dramatically elevated GPU workload, and consequently, increased GPU resource contention. Our design improves both the mean and maximum E2E latency, respectively, by 4–27% and 19–33%. Furthermore, Figure 11c captures 36–56% average reduction in flatness using our techniques, peaking at 56% improvement through CPU affinity. This optimization minimizes extraneous Linux scheduling overhead among the three inference threads in this application, as the inferences are time-sliced in parallel CUDA streams on the GPU.

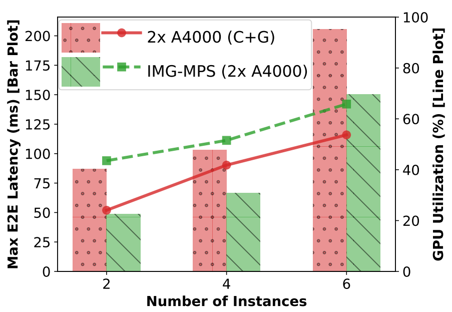


Fig. 8: Maximum E2E Latency and GPU Utilization with two A4000s

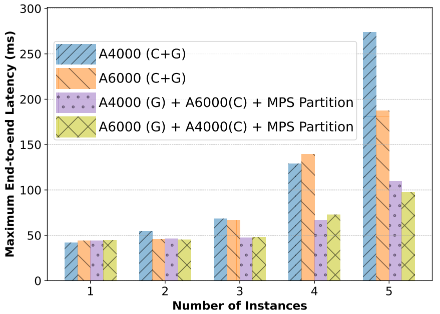


Fig. 9: Maximum E2E Latency for different GPU role combinations

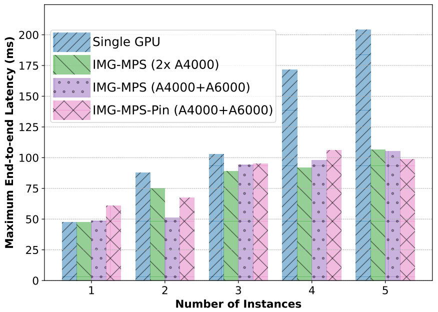
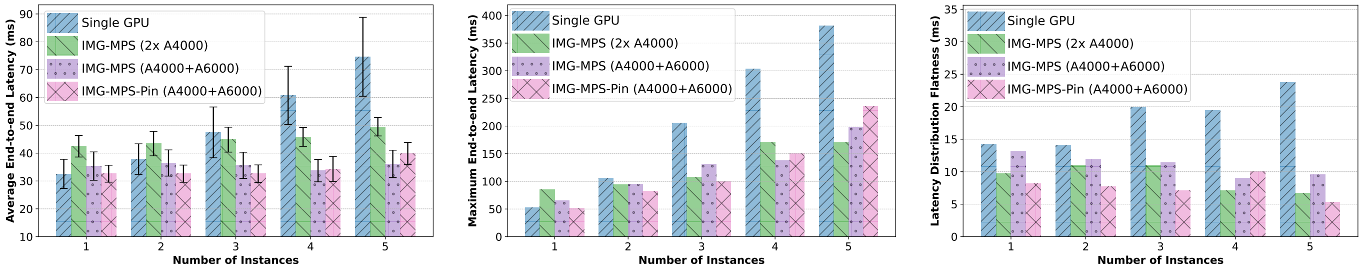


Fig. 10: Maximum E2E Latency in Ultrasound Segmentation



(a) Average End-to-end Latency **(b) Maximum End-to-end Latency** **(c) Latency Distribution Flatness**

Fig. 11: Multi-AI Ultrasound Application Performance

F. Heterogeneous Applications

We revisit the example in Section I where two different applications (endoscopy tool tracking and ultrasound segmentation) are concurrently executed. Table II show that our methods result in a maximum E2E latency reduction of 22–32%. This enables manufacturers to consolidate multiple AI applications on a single workstation, making devices like endoscopic ultrasound (EUS) [23], [40] more predictable, safe and affordable. Our techniques also offer a foundational step towards multi-function medical device development [19].

TABLE II: Maximum E2E Latency (ms) for concurrent Endoscopy Tool Tracking and Ultrasound Segmentation

Application	**One A4000**	**One A4000 + MPS**	**Two A4000s + IMG-MPS**
Endoscopy Tool Tracking	133.76	**91.24**	**90.57**
Ultrasound Segmentation	80.61	**62.49**	**55.23**

G. Discussion

Based on our empirical study, we suggest the following design guidelines for managing heterogeneous GPU workloads that include equally important compute and graphics tasks:

- In scenarios where external constraints prohibit the use of multiple GPUs - due to cost, energy, or other factors - CUDA MPS could be utilized to partition a single GPU, after a thorough resource profiling of the involved applications. However, benefits only with this setup are limited, as shown in Section VI-A, due to contention between graphics and multiple compute contexts.
- Deploying distinct GPUs for compute and graphics tasks greatly enhances performance predictability. For cost considerations, a less powerful GPU may handle graphics if it meets the application's memory needs. For instance, an IGX Orin with future support for inter-operable iGPU and discrete GPU would be a suitable platform.

VII. Related Work

Despite the rapid integration of AI technologies into medical devices, ensuring deterministic performance in these systems has received little attention. Existing endoscopy and colonoscopy AI application studies emphasize primarily live video processing and accuracy, neglecting end-to-end performance metrics [28], [30]. This work addresses the gap by investigating systems techniques to enhance time-critical characteristics of medical AI applications.

Prior research in real-time systems has explored priority-aware GPU task and interrupt scheduling in time-slicing mode [8], [17], [18], [24]–[26], [44], [51], [52]. Our work, however, focuses on the spatial isolation of heterogeneous workloads for greater predictability. Inference latency has also been studied and optimized in the cloud computing context [12], [13], [16], [20], [31], [43], but these works neglect latency as a safety-critical metric. Importantly, none of the previous works considered heterogeneous GPU workloads involving equally important compute and graphics tasks.

Our approach employs CUDA MPS and heterogeneous workload partitioning across multiple GPUs to achieve greater latency predictability. Unlike OS-based and device driver solutions [8], [18], [24]–[26], [44], [47], [48], our methodology is applicable to NVIDIA, AMD and other GPUs without requiring system software and driver modifications, making it particularly viable for the medical industry, which usually mandates long-term (3–10 years) software maintenance. Dhakal *et al.* explored CUDA MPS for inference performance but did not consider E2E latency determinism [16]. Zou et al. investigated a theoretical model of MPS and simulated task schedulability [56]. Our empirical study, in contrast, provides actionable design insights for predictable performance. Finally, we use the Holoscan SDK [34], a novel AI computing platform that resembles the programming model of real-time frameworks [49] like ROS [42] and Arduino [4], because it has extensive support for GPUs in the edge-computing domains, especially suitable for medical AI applications.

VIII. Conclusion and Future Work

This paper presents GPU workload partitioning and load-balancing techniques to improve end-to-end latency predictability in medical AI applications using the NVIDIA Holoscan platform. By leveraging CUDA MPS partitioning and multi-GPU configurations for hardware isolation between compute and graphics workloads, we demonstrate significant gains across various latency determinism metrics against single- and multi-GPU baselines. We expect to motivate further performance studies involving medical AI workloads.

Although our study focuses on Holoscan, it could be used as a design guide for other edge-computing systems with concurrent and heterogeneous GPU workloads. Future endeavors will investigate solutions for more predictable performance on single GPU platforms. Potential benefits of hardware virtualization technologies will also be considered in terms of predictability, safety, and cost-effectiveness.

References

[1] Activ Surgical, "Activ Surgical aims to bring real-time AI to their surgery platform using NVIDIA Clara Holoscan on newly-launched NVIDIA IGX," https://www.activsurgical.com/news/nvidia-launches-igx-edge-ai-computing-platform-for-safe-secure-autonomous-systems.

[2] AJA Video Systems, "KONA XM - Video I/O for AI/AR Medical Devices," https://www.aja.com/products/kona-xm.

[3] S. Ali, "Where do we stand in AI for endoscopic image analysis? Deciphering gaps and future directions," *npj Digital Medicine*, 2022.

[4] Arduino, "Arduino," https://www.arduino.cc/.

[5] R. Ausavarungnirun, V. Miller, J. Landgraf, S. Ghose, J. Gandhi, A. Jog, C. J. Rossbach, and O. Mutlu, "Mask: Redesigning the gpu memory hierarchy to support multi-application concurrency," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 503–518, 2018.

[6] J. Bakita and J. H. Anderson, "Hardware compute partitioning on nvidia gpus," in *IEEE RTAS*, 2023.

[7] T. Bradley, "Hyper-Q example," *NVIDIA. Whitepaper v1. 0*, 2012.

[8] N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru, "Deadline-based scheduling for GPU with preemption support," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 119–130.

[9] J. W. Catto, P. Khetrapal, F. Ricciardi, G. Ambler, N. R. Williams, T. Al-Hammouri, M. S. Khan, R. Thurairaja, R. Nair, A. Feber *et al.*, "Effect of robot-assisted radical cystectomy with intracorporeal urinary diversion vs open radical cystectomy on 90-day morbidity and mortality among patients with bladder cancer: a randomized clinical trial," *Jama*, vol. 327, no. 21, pp. 2092–2103, 2022.

[10] J. P. Cerrolaza, R. Obermaisser, J. Abella, F. J. Cazorla, K. Grüttner, I. Agirre, H. Ahmadian, and I. Allende, "Multi-core devices for safety-critical systems: A survey," *ACM Computing Surveys (CSUR)*, 2020.

[11] A. Cherubini and N. N. Dinh, "A Review of the Technology, Training, and Assessment Methods for the First Real-Time AI-Enhanced Medical Device for Endoscopy," *Bioengineering*, vol. 10, no. 4, p. 404, 2023.

[12] S. Choi, S. Lee, Y. Kim, J. Park, Y. Kwon, and J. Huh, "Serving heterogeneous machine learning models on multi-gpu servers with spatio-temporal sharing," in *USENIX ATC*, 2022, pp. 199–216.

[13] M. Chow, A. Jahanshahi, and D. Wong, "KRISP: Enabling Kernel-wise RIght-sizing for Spatial Partitioned GPU Inference Servers," in *IEEE HPCA*, 2023.

[14] Deltacast, "MIXED INTERFACE CARDS (SDI/HDMI/ASI)," https://www.deltacast.tv/products/developer-products/mixed-interface-cards.

[15] D. E. Denning, "A lattice model of secure information flow," *Communications of the ACM*, 1976.

[16] A. Dhakal, S. G. Kulkarni, and K. Ramakrishnan, "Gslice: controlled spatial sharing of gpus for a scalable inference platform," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 492–506.

[17] G. Elliott and J. Anderson, "Building a real-time multi-gpu platform: Robust real-time interrupt handling despite closed-source drivers," *24th ECRTS*, 2012.

[18] G. A. Elliott, B. C. Ward, and J. H. Anderson, "GPUSync: A framework for real-time GPU management," in *IEEE RTSS*, 2013.

[19] Food, D. Administration *et al.*, "Multiple function device products: Policy and considerations," 2021.

[20] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving DNNs like Clockwork: Performance predictability from the bottom up," in *14th USENIX OSDI*, 2020.

[21] iCardio.ai, "icardio.ai," https://www.icardio.ai/.

[22] imgui, "Dear ImGui: Bloat-free Graphical User interface," https://github.com/ocornut/imgui.

[23] Intuitive Surgical, "How Ion Works," https://www.intuitive.com/en-us/products-and-services/ion/how-ion-works.

[24] S. Jain, I. Baek, S. Wang, and R. Rajkumar, "Fractional GPUs: Software-based compute and memory bandwidth reservation for GPUs," in *RTAS*. IEEE, 2019.

[25] S. Kato, K. Lakshmanan, R. Rajkumar, Y. Ishikawa *et al.*, "Time-Graph:GPU Scheduling for Real-Time Multi-Tasking Environments," in *USENIX Annual Technical Conference*, 2011.

[26] S. Kato, M. McThrow, C. Maltzahn, and S. Brandt, "Gdev: First-Class GPU Resource Management in the Operating System," in *USENIX Annual Technical Conference*, 2012, pp. 401–412.

[27] C. Kubisch, "Life of a triangle-NVIDIA's logical pipeline," *NVIDIA Developer*, vol. 9, 2015.

[28] H. Luo, G. Xu, C. Li, L. He, L. Luo, Z. Wang, B. Jing, Y. Deng, Y. Jin, Y. Li *et al.*, "Real-time artificial intelligence for detection of upper gastrointestinal cancer by endoscopy: a multicentre, case-control, diagnostic study," *The Lancet Oncology*, vol. 20, no. 12, 2019.

[29] Medtronic, "Medtronic to boost AI innovation with new platform introduction," https://news.medtronic.com/2023-03-22-Medtronic-to-boost-AI-innovation-with-new-platform-introduction.

[30] Y. Mori, S.-e. Kudo, M. Misawa, Y. Saito, H. Ikematsu, K. Hotta, K. Ohtsuka, F. Urushibara, S. Kataoka, Y. Ogawa *et al.*, "Real-time use of artificial intelligence in identification of diminutive polyps during colonoscopy: a prospective study," *Annals of internal medicine*, vol. 169, no. 6, pp. 357–366, 2018.

[31] K. K. Ng, H. M. Demoulin, and V. Liu, "Paella: Low-latency Model Serving with Software-defined GPU Scheduling," in *SOSP*, 2023.

[32] J. Nickolls and D. Kirk, "Graphics and Computing GPUs," *Computer Organization and Design*, 2009.

[33] NVIDIA, "HoloHub," https://github.com/nvidia-holoscan/holohub.

[34] ——, "Holoscan," https://docs.nvidia.com/holoscan/sdk-user-guide/index.html.

[35] ——, "Holoscan SDK Operators," https://docs.nvidia.com/holoscan/sdk-user-guide/holoscan_operators_extensions.html.

[36] ——, "How to Overlap Data Transfers in CUDA C/C++," https://developer.nvidia.com/blog/how-overlap-data-transfers-cuda-cc/.

[37] ——, "MPS," https://docs.nvidia.com/deploy/mps/index.html.

[38] ——, "TensorRT," https://docs.nvidia.com/deeplearning/tensorrt/.

[39] ——, "Technical brief: Multi-instance gpu and nvidia virtual compute server," Tech. Rep., 2023.

[40] Olympus, "Universal Ultrasound Processor (EU-ME2)," https://medical.olympusamerica.com/products/universal-ultrasound-processor-eu-me2.

[41] N. Otterness and J. H. Anderson, "Exploring AMD GPU scheduling details by experimenting with "worst practices"," in *International Conference on Real-Time Networks and Systems*, 2021.

[42] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, 2009.

[43] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "{INFaaS}: Automated model-less inference serving," in *USENIX ATC*, 2021.

[44] C. J. Rossbach, J. Currey, M. Silberstein, B. Ray, and E. Witchel, "PTask: operating system abstractions to manage GPUs as compute devices," in *SOSP*, 2011.

[45] T. C. Schroeder, "Peer-to-peer & Unified Virtual Addressing," in *GTC, NVIDIA*, 2011.

[46] G. Sellers and J. Kessenich, *Vulkan programming guide: The official guide to learning Vulkan*. Addison-Wesley Professional, 2016.

[47] S. Sinha, "Towards a centralized multicore automotive system," Ph.D. dissertation, Boston University, 2022.

[48] S. Sinha and R. West, "Towards an Integrated Vehicle Management System in DriveOS," *ACM TECS*, vol. 20, no. 5s, 2021.

[49] ——, "End-to-end scheduling of real-time task pipelines on multiprocessors," *Journal of Systems Research*, vol. 2, no. 1, Aug. 2022.

[50] A. Steimers and M. Schneider, "Sources of risk of AI systems," *International Journal of Environmental Research and Public Health*, vol. 19, no. 6, p. 3641, 2022.

[51] I. Tanasic, I. Gelado, J. Cabezas, A. Ramirez, N. Navarro, and M. Valero, "Enabling preemptive multiprogramming on gpus," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 193–204, 2014.

[52] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo, "Simultaneous multikernel GPU: Multi-tasking throughput processors via fine-grained sharing," in *IEEE HPCA*, 2016.

[53] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra *et al.*, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM TECS*, vol. 7, no. 3, pp. 1–53, 2008.

[54] T. Yandrofski, J. Chen, N. Otterness, J. H. Anderson, and F. D. Smith, "Making powerful enemies on NVIDIA GPUs," in *IEEE RTSS*, 2022.

[55] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed Adaptive Deep Learning Inference on Resource-Constrained IoT Edge Clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[56] A. Zou, J. Li, C. D. Gill, and X. Zhang, "RTGPU: Real-time GPU scheduling of hard deadline parallel tasks with fine-grain utilization," *Transactions on Parallel and Distributed Systems*, 2023.

IX. Appendix

In this section, we include extra results from our experiments for clarification.

Figure 12 shows the difference of workload distribution with different solutions including the techniques introduced in this paper. The compute of 3 applications and graphics are color-coded. We imagine that every application needs 2 SMs for their compute. In a single GPU without MPS, graphics and compute of all applications share all the SMs. When MPS is enabled on single GPU, compute workload of an application is reserved for a number of SMs, but graphics workload is shared across all the SMs. Default multi-GPU setup is similar to single GPU but with two GPUs. In our proposed workload-isolated multi-GPU, graphics is confined to GPU1, but compute of all applications still share all the SMs. When MPS is enabled on the workload-isolated multi-GPU setup, graphics is isolated to GPU1, and compute workload of every app is restricted to a dedicated set of SMs in GPU2, enabling predictable performance for AI applications with real-time visualization capabilities. In this configuration, the unoccupied SMs could be opportunistically used by the GPU as necessary.

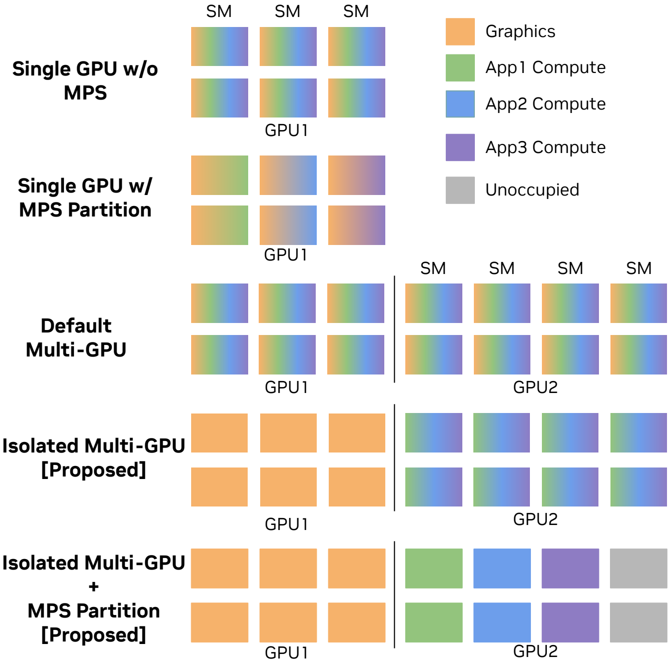


Fig. 12: Heterogeneous GPU Workload Partitioning

Figure 13 and Figure 14 show sample outputs from the Endoscopy Tool Tracking and Multi-AI ultrasound applications described in Section III.

A. Additional Evaluation

We report that for most comparisons between different configurations, we have observed a Z-score greater than 1.96 with a two-tailed Mann-Whitney U test with significance level 0.05, indicating the latency distribution difference. However, we do not include this data, as we have analyzed and quantified different components of latency distribution for an in-depth view on end-to-end latency predictability.

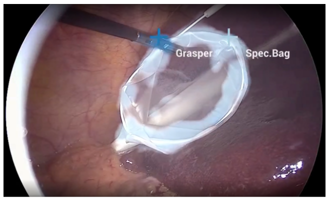


Fig. 13: Endoscopy Tool Tracking Sample

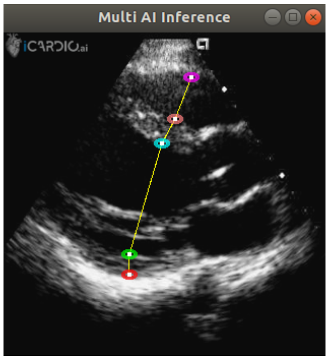


Fig. 14: Multi-AI Ultrasound Sample

1) Impact of Active Thread Percentage in CUDA MPS: To quantify the effects of active thread percentage value in CUDA MPS partitioning, we varied the active thread allocation per MPS client. The results, displayed in Table III, reveal that a 15% allocation counter-intuitively yields better maximum E2E latency and latency distribution tail for five instances. This improvement is due to the GPU scheduler's dynamic optimization of unallocated threads. The results underscore the importance of application-specific GPU resource profiling. Over-allocation could offer stable performance until GPU saturation, at which point it can negatively affect co-running processes, as seen with 25% allocation (5x25=125%). Thus, reserving a few GPU threads for dynamic allocation is recommended in CUDA MPS configurations.

TABLE III: Impact on E2E Latency for Variation on Active Thread Percentage of CUDA MPS for 5 Instances

Active Thread Percentage	Maximum	Tail	Flatness
15%	94.78	56.55	8.28
20%	109.58	70.82	9.28
25%	124.49	85.89	9.19

B. Additional Experimental Results

Figure 15 shows the latency distribution tail for the endoscopy tool tracking application against single GPU without and with CUDA MPS partitions, whose explanation is already covered in Section VI-A. Figure 16 depicts that workload separation in distinct GPUs improves latency distribution tail.

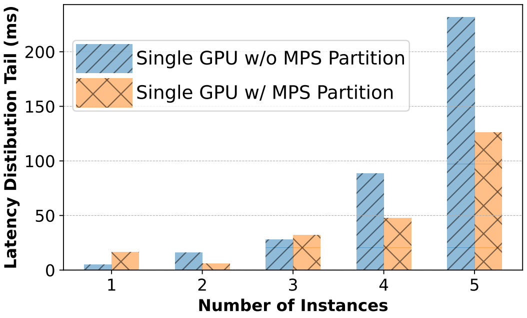


Fig. 15: Latency Distribution Tail against Single GPU w/o and w/ CUDA MPS Partition for Endoscopy

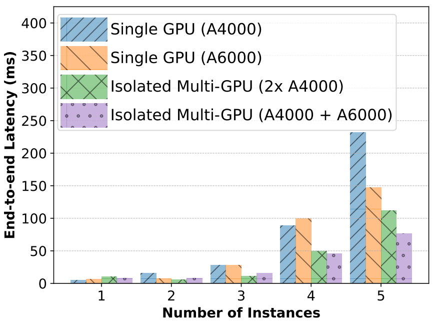


Fig. 16: Latency Distribution Tail with Endoscopy Tool Tracking for Single GPU vs. Multi-GPU Configurations

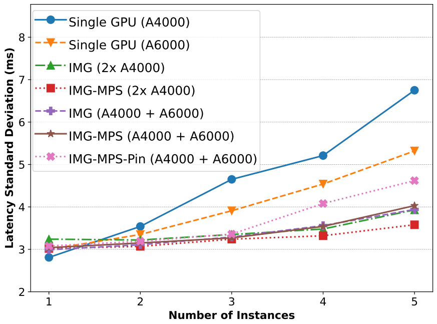


Fig. 17: Standard Deviation of Endoscopy Tool Tracking Latency for All Optimizations

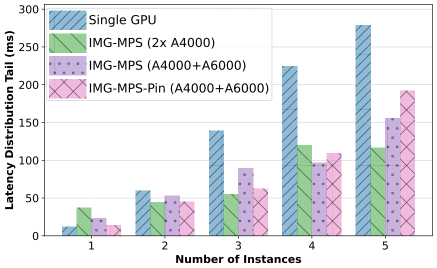
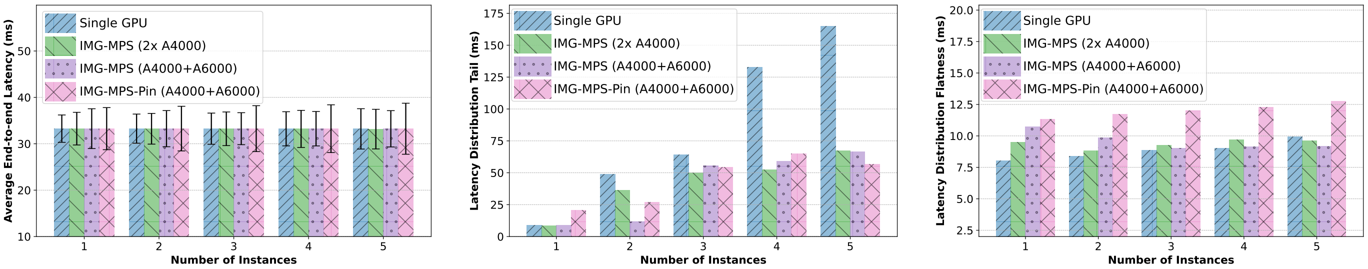


Fig. 18: Latency Distribution Tail for Multi-AI Ultrasound



(a) Average E2E Latency

(b) Latency Distribution Tail

(c) Latency Distribution Flatness

Fig. 19: Ultrasound Segmentation Application Performance - rest of the results

Figure 17 shows the standard deviation with all optimizations for the endoscopy tool tracking application.

Our design achieves 10–29% reduction in latency distribution tail for multi-AI ultrasound application, illustrated in Figure 18. Figure 19 demonstrates better standard deviation and latency distribution tail with our proposed design choices in the ultrasound bone scoliosis segmentation application, while the flatness in Figure 19c is comparable and stable between the setups.