ABSTRACT

Title of Dissertation: LONGITUDINAL DATA ANALYTICS OF HPC SYSTEMS AND APPLICATIONS

Onur Cankur

Dissertation Directed by: Professor Abhinav Bhatele
Department of Computer Science

Modern high-performance computing systems continuously collect large volumes of telemetry through always-on monitoring services. These data describe system behavior, application resource usage, and changing operating conditions over time, and they enable performance analysis, workload characterization, and proactive resource management. As supercomputers grow in scale and complexity, analyzing such longitudinal telemetry becomes increasingly important because it helps reveal how different scientific workloads use modern systems and how system operation can be improved to better support diverse applications. Without effective analysis methods, important patterns in system utilization remain hidden. However, analyzing such data is difficult because HPC systems are large, heterogeneous, and dynamic. In addition, the relationship between low-level monitoring data and application behavior is often indirect and complex. Traditional analysis methods often rely on application-specific instrumentation, pre-run benchmarking, or do not scale well in production environments with diverse workloads. This dissertation studies how longitudinal telemetry can be used to better understand and manage modern

HPC systems. It uses production monitoring data from a leadership-class supercomputer in two complementary ways. First, it combines system-wide monitoring data with scheduler metadata to examine GPU workload behavior. This analysis characterizes how production applications use GPU resources, how utilization varies across GPUs and over time, and how workload behavior relates to compute activity, memory activity, and resource imbalance. Second, it combines scheduler metadata with network telemetry to identify similar runs, model performance variation, and predict runs that are likely to perform unusually slowly relative to similar runs based on telemetry collected near the beginning of execution. This is done without application-specific profiling or prior knowledge of the code. This dissertation shows that always-on monitoring data can be used not only to understand workload behavior at scale, but also to anticipate performance problems and support more informed operational decisions.

LONGITUDINAL DATA ANALYTICS OF HPC SYSTEMS AND APPLICATIONS

by

Onur Cankur

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2026

Advisory Committee:
Professor Abhinav Bhatele, Chair/Advisor
Dr. Brian Austin
Professor Alan Liu
Professor Donald Yeung
Professor Abhinav Shrivastava

Benim manevî mirasım, bilim ve akıldır. Benden sonrakiler, bizim aşmak zorunda olduğumuz çetin ve köklü güçlükler önünde, belki amaçlara tamamen eremediğimizi, fakat asla ödün vermediğimizi, akıl ve bilimi rehber edindiğimizi onaylayacaklardır.

My spiritual legacy is science and reason. Those who come after me will affirm that, in the face of the severe and deep-rooted difficulties we had to overcome, perhaps we did not fully attain our goals, but we never compromised, and we took reason and science as our guide.

Mustafa Kemal Atatürk

Acknowledgements

I owe sincere thanks to the many people who have supported me throughout this journey. I am deeply grateful to all of them. Their encouragement and patience sustained me through these years, and I would not have reached this point without them.

My advisor, Dr. Abhinav Bhatele, has shaped my growth as a researcher in countless ways. He taught me how to think carefully about technical problems, how to present work clearly, and how to become a better researcher overall. I also learned from his example as a mentor. He has always been kind, patient, and thoughtful. I am especially grateful for the patience and understanding he showed me during the difficult parts of my PhD. When things did not go well, he responded with kindness and perspective.

I am also very grateful for the friends who have supported me during these years. My lab friends have made the lab a welcoming and supportive place to work and grow. I feel lucky to have spent this time surrounded by people who are not only smart, but also kind and generous. Friends outside the lab have also supported me in important ways through these years. We have shared so much together, and their presence has brought joy, comfort, and balance to my life.

My collaborators at Lawrence Berkeley National Laboratory have played an essential role in this work. Their willingness to share their data, time, and expertise made this dissertation possible. Working with them has been a privilege, and I am very grateful for their insight, collaboration, and support.

My undergraduate advisor, Dr. Adnan Ozsoy, has also had a lasting impact on my career. His guidance and support played an important role in my decision to come to the United States for graduate school. He encouraged me at a critical point in my life, and that support has con-

tinued ever since. We have stayed in touch over the years, and I remain grateful for his advice, encouragement, and continued support as I move forward in my career.

I owe my family more gratitude than I can express for their love and support through every stage of my life. They made sacrifices throughout this time alongside me, and this accomplishment is as much theirs as it is mine. The hardest part of my PhD was the distance and the longing that came with being away from them. I could not have made it through these years without their love, patience, and constant support.

Finally, during my PhD, the luckiest thing that happened to me was meeting Saadet, the woman I will soon marry. Her love, support, and constant belief in me made even the hardest parts of my PhD easier to endure. She has stood by me through the stress, uncertainty, and exhaustion, and she has kept encouraging me and reminding me to believe in myself when I struggled. Her presence has made these years not only easier, but also happier, fuller, and far more meaningful.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AI	Artificial Intelligence
CPU	Central Processing Unit
DCGM	Data Center GPU Manager
FP	Floating-point
GPU	Graphics Processing Unit
HBM	High-Bandwidth Memory
HPC	High Performance Computing
IQR	Interquartile Range
LDMS	Lightweight Distributed Metric Service
ML	Machine Learning
NERSC	National Energy Research Scientific Computing Center
NIC	Network Interface Card
ODA	Operational Data Analytics
PR-AUC	Area Under the Precision-Recall Curve

Chapter 1: Introduction

High-performance computing (HPC) systems are essential to modern science and engineering. They support large-scale simulations, data analysis, and artificial intelligence (AI) workloads that depend on a variety of system components, including Graphics Processing Units (GPUs), deep memory hierarchies, and high-speed interconnects. System administrators continuously monitor these components, which generates massive volumes of telemetry. These data record hardware and software behavior over time and provide an opportunity to study production behavior at system scale. Turning such routine monitoring data into useful system-level and job-level knowledge is a central goal of operational data analytics (ODA) in HPC [57].

Despite this opportunity, extracting actionable insights from HPC telemetry remains difficult. Modern supercomputers collect monitoring data at high frequency across many subsystems. These measurements are heterogeneous, noisy, and strongly dependent on context such as job size, placement, system load, and application behavior. A single hardware counter rarely provides a complete picture, and a small subset of analysis metrics often misses important interactions among subsystems. Because low-level telemetry provides only an indirect view of application execution, relating these measurements to workload behavior or performance outcomes requires careful integration with scheduler metadata and analysis at an appropriate level of abstraction.

These challenges are especially important on modern GPU-accelerated systems. As HPC architectures grow more heterogeneous, resource demands, execution behavior, and performance bottlenecks vary more widely across production applications. In this setting, coarse utilization summaries and limited application-specific profiling do not provide an adequate view of system behavior. Production-scale analysis methods must therefore work with routine monitoring data, accommodate diverse workloads, and reveal patterns that remain useful across a large and evolving workload mix.

This dissertation studies two complementary uses of longitudinal telemetry in HPC systems. The first is descriptive: it uses system-wide monitoring data collected over long time periods to characterize workload behavior. The second is predictive: it combines telemetry with scheduler metadata to identify signals associated with performance variability across comparable production runs. In both cases, the dissertation emphasizes methods that rely on routinely collected system data rather than intrusive instrumentation, application-specific knowledge, or pre-run benchmarking.

The first part of this dissertation combines system-wide monitoring data with scheduler metadata to construct job-level views of GPU activity in production jobs. This analysis examines how production jobs use GPU resources, how resource usage varies across jobs and over time, and how workload behavior relates to metrics such as utilization, compute activity, memory activity, and imbalance. This perspective makes it possible to study production GPU workloads at system scale while retaining job-level interpretability.

The second part of this dissertation focuses on run-to-run performance variability in production GPU jobs. Identical runs can exhibit different runtimes because system conditions change over time and jobs compete for shared resources. This variability matters to both users and sys-

tem operators, but it is difficult to study at scale without application-specific measurements. This dissertation uses scheduler metadata and network telemetry to group comparable runs, define slowdown relative to similar runs, and model whether a run is likely to experience slowdown. The goal is to show that routine monitoring data contain useful signals for predicting performance variability in production jobs, even without application-specific instrumentation or pre-run benchmarking.

The empirical analysis in this dissertation is centered on Perlmutter, a leadership-class supercomputer, with particular emphasis on its GPU partition and production monitoring infrastructure. This setting grounds the dissertation in a modern GPU-accelerated environment and allows the methods to be evaluated on real operational data. Although the empirical study focuses on a single system, the methods are designed to transfer to other HPC systems that provide similar monitoring data and scheduler metadata, both of which are commonly available in production environments.

1.1 Summary of Contributions

This dissertation addresses the challenges described above by developing methods that use routine monitoring data to analyze and predict performance behavior in production GPU systems. It combines system-wide telemetry data with scheduler metadata to produce job-level insight into GPU workload behavior and to model run-to-run performance variability under practical production constraints.

Chapter 4 presents several contributions. We develop a job-level characterization of GPU workloads on Perlmutter using a robust job-attribution pipeline and two complementary telemetry

datasets that span a long time range across 2023 and 2025. This design lets us study production GPU behavior at scale and over a broader range of workloads than a single short collection period would allow. We use a very rich set of GPU hardware counters to quantify multiple aspects of job behavior at job level. We label jobs as compute-bound or memory-bound and find that memory-bound jobs consume more energy than compute-bound jobs at comparable GPU-hours. We introduce a time-windowed spatial imbalance metric to quantify uneven GPU usage within a job and to detect load imbalance at minute scale. We classify jobs by floating-point pipeline activity and quantify differences in GPU utilization and in spatial and temporal imbalance across these classes. This analysis indicates that jobs with tensor activity tend to reach higher utilization. We also compare ML and non-ML jobs and identify differences in GPU usage behavior and workload characteristics. Finally, we use the broad counter set to quantify job-level relationships among GPU hardware counters and to identify the counters that track GPU utilization.

Chapter 5 presents a new approach to predict run-to-run performance variability for arbitrary production GPU applications using only system-wide monitoring data, without application profiling, code instrumentation, or pre-run benchmarking. To the best of our knowledge, this is the first variability prediction method that relies only on telemetry data and does not require instrumentation. We make variability prediction feasible for arbitrary production workloads by using job metadata to group similar runs that approximate repeated executions of the same application under the same launch configuration, and by defining a group-specific variability value. This design enables prediction without application-specific knowledge. We also quantify how telemetry scope and timing affect prediction performance by comparing allocated-node and all-node telemetry, as well as pre-start and post-start windows.

1.2 Outline of Dissertation

The remainder of this dissertation is organized as follows. Chapter 2 provides background on Perlmutter, the monitoring infrastructure, and the data sources used in this dissertation. Chapter 3 reviews prior work on resource usage analysis and performance variability in HPC systems. Chapter 4 presents a job-level characterization of GPU workloads using system-wide monitoring data. Chapter 5 presents a method for predicting run-to-run performance variability in production GPU jobs using scheduler metadata and network telemetry. Chapter 6 provides a summary of the contributions and directions for future work.

Chapter 2: Background

This dissertation uses system telemetry and scheduler metadata from Perlmutter, a leadership-class supercomputer at NERSC. This chapter provides the shared system and data context for the technical chapters that follow. It summarizes the relevant aspects of the Perlmutter architecture, describes the monitoring and scheduling infrastructure, and explains the structure of the telemetry and metadata used throughout this dissertation.

2.1 Perlmutter Architecture

Perlmutter is a heterogeneous supercomputer at NERSC with both CPU-only and GPU-accelerated nodes. The system contains 3,072 CPU-only nodes and 1,792 GPU nodes. This dissertation focuses on production jobs that run on the GPU partition. Each GPU node contains one AMD EPYC 7763 processor, four NVIDIA A100 GPUs, and 256 GB of DDR4 system memory. Among the GPU nodes, 1,536 use A100 GPUs with 40 GB of high-bandwidth memory (HBM) per GPU, and 256 use A100 GPUs with 80 GB of HBM per GPU.

The GPU partition uses the Slingshot 11 interconnect with HPE Cray Cassini network interface cards (NICs). Each GPU node has four NICs. The interconnect follows a three-level dragonfly topology. In the GPU partition, each compute cabinet contains eight chassis. Each chassis contains eight compute blades and four switch blades, and each GPU compute blade

contains two GPU nodes. The four NICs on each node connect to switch blades at the rear of the cabinet. Each cabinet contains 32 switch blades arranged as two groups of 16 switches.

Within a GPU node, the AMD EPYC 7763 processor connects to the four A100 GPUs through PCIe 4.0. The GPUs also connect to one another through NVLink. Each NVLink connection provides 25 GB/s of bidirectional bandwidth. The A100 GPUs provide up to 1,555 GB/s of memory bandwidth in the 40 GB configuration and 2,039 GB/s in the 80 GB configuration.

The computational capability of the A100 GPUs depends on precision and execution mode. In standard floating-point execution, each GPU provides 19.5 teraflops for FP32 and 9.7 teraflops for FP64. When tensor cores are used, the peak performance increases to 155.9 teraflops for TF32, 311.9 teraflops for FP16, and 19.5 teraflops for FP64 tensor-core execution. These architectural characteristics provide important context for the GPU and network analyses in the later chapters.

2.2 Monitoring and Scheduler Infrastructure

Perlmutter collects telemetry through the Lightweight Distributed Metric Service (LDMS) [2]. LDMS uses a hierarchical design. Lightweight samplers run on compute nodes and capture hardware counters at fixed intervals. Aggregators retrieve these sampled records and store them for later analysis. LDMS supports many plugins for different subsystems. This dissertation uses two telemetry sources collected through LDMS. The GPU workload analysis uses the Data Center GPU Manager (DCGM) plugin [59] to collect GPU telemetry. The variability analysis uses Cassini NIC counters to characterize network behavior.

Perlmutter uses Slurm [79] as its workload manager and scheduler. Slurm records metadata

such as job identifiers, step identifiers, start and end times, node allocations, requested resources, job state, and submission commands. This dissertation retrieves Slurm metadata through `sacct` and combines it with LDMS telemetry to attribute system measurements to running jobs or job steps.

2.3 Data Structure

The telemetry data in this dissertation have a time-series structure. Each record includes a timestamp, a node identifier, a device identifier, and counter values from a particular monitoring plugin. In the GPU analysis, the device identifier corresponds to a GPU. In the variability analysis, it corresponds to a NIC.

2.3.1 DCGM Counters

GPU telemetry from the DCGM plugin records per-GPU measurements, which provide a device-level view of accelerator behavior over time. The DCGM counters are sampled every 10 seconds. Table 2.1 lists the GPU counters used in this study. GPU_UTIL is the fraction of time at least one kernel was executing on the device. SM_ACTV is the fraction of time at least one warp was active (including stalled warps waiting on memory). FP16_ACTV, FP32_ACTV, FP64_ACTV, and TNSR_ACTV are activity fractions for the corresponding pipes; higher values indicate greater activity of that pipeline. DRAM_ACTV is the fraction of cycles issuing device-memory traffic. MEM_UTIL is a memory-copy / copy-engine utilization signal. NVLINK_TX,RX and PCIE_TX,RX report byte rates per second over NVLink (payload only) and PCIe (headers plus payload), respectively. HBM_USED is the absolute HBM footprint (MB).

Table 2.1: DCGM counters used in this thesis and their explanations.

Counter Name	Short Name	Description
DCGM_FI_DEV_GPU_UTIL	GPU_UTIL	The fraction of time during which at least one kernel was executing on the GPU.
DCGM_FI_DEV_MEM_COPY_UTIL	MEM_UTIL	The fraction of time during which device memory was read or written.
DCGM_PROF_DEV_DRAM_ACTIVE	DRAM_ACTV	The fraction of cycles where data was sent to or received from device memory.
DCGM_FI_PROF_PIPE_FP16_ACTIVE	FP16_ACTV	The fraction of cycles the FP16 (half-precision) pipes were active.
DCGM_FI_PROF_PIPE_FP32_ACTIVE	FP32_ACTV	The fraction of cycles the FP32 (single-precision and integer) pipes were active.
DCGM_FI_PROF_PIPE_FP64_ACTIVE	FP64_ACTV	The fraction of cycles the FP64 (double-precision) pipes were active.
DCGM_FI_PROF_PIPE_TENSOR_ACTIVE	TNSR_ACTV	The fraction of cycles the tensor pipes were active.
DCGM_FI_PROF_SM_ACTIVE	SM_ACTV	The fraction of time at least one warp was active, averaged over all multiprocessors.
DCGM_FI_PROF_NVLINK_{TX/RX}_BYTES	NVLINK_{TX/RX}	The rate of data transmitted/received over NVLink, not including protocol headers, in bytes per second.
DCGM_FI_PROF_PCIE_{TX/RX}_BYTES	PCIE_{TX/RX}	The rate of data transmitted/received over PCIe, including both protocol headers and data payloads, in bytes per second.
DCGM_FI_DEV_FB_USED	HBM_USED	Absolute amount of high-bandwidth memory (HBM) used (MB).
DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION	TOTAL_ENG	Total energy consumption for the GPU in mJ since the driver was last reloaded.
DCGM_FI_DEV_GPU_TEMP	GPU_TEMP	Current temperature readings for the device, in degrees Celsius.

TOTAL_ENG is the cumulative GPU energy (mJ) since the driver was last reloaded. GPU_TEMP is the current device temperature (Celsius).

2.3.2 NIC Counters

LDMS collects Cassini NIC hardware counters from GPU nodes. Unlike many GPU activity counters, the NIC counters used in this dissertation are monotonically increasing. Later analyses therefore use changes over time rather than raw counter values. Table 2.2 lists the counters we use in this study. The counters fall into several conceptual groups. Traffic volume counters report packets transmitted and received, including per-traffic-class breakdowns. Pause

counters capture flow control and can indicate congestion or injection throttling. Replay and timeout counters reflect retry behavior that can increase when links or buffers are under stress. Physical-layer codeword counters summarize link-layer error behavior. Posted and non-posted PCIe traffic counters, along with their blocked-event counters, provide a view of pressure at the host-to-NIC interface.

2.3.3 Slurm Job Metadata

Slurm is the system scheduler and workload manager. At the job-step level, it records metadata such as job identifiers, start and end times, requested resources, and the command line from the submission script. A job step is a unit of execution within a Slurm job, typically created by an `srun` launch in the job script. Each `srun` launch is therefore associated with a unique job-step identifier.

Slurm metadata differs from telemetry because it provides static records about jobs and job steps rather than time-series samples. These records include start and end times, node lists, resource requests, executable or submission-command information, and final state. The later chapters combine Slurm metadata with LDMS telemetry to construct job-aware datasets. Chapter 4 uses merged GPU telemetry and scheduler data to analyze workload behavior. Chapter 5 combines NIC telemetry and scheduler data to study run-to-run performance variability.

Subsequent chapters describe chapter-specific sampling intervals, selected counters, data retrieval steps, and preprocessing decisions. This chapter provides only the shared system and data context needed to understand those analyses.

Table 2.2: Telemetry counters used in this study. The conceptual-group column identifies the subsystem or behavior each counter represents. Colored cells highlight the six conceptual groups in the feature set.

Counter Name	Conceptual Group	Description
parbs_tarb_pi_non_posted_pkt	PCIe traffic	# of PCIe packets transferred on the non-posted path, which is typically associated with read-like host interface traffic.
parbs_tarb_pi_non_posted_blo	PCIe traffic	# of cycles in which the non-posted PCIe path is blocked. Large blocked-cycles per packet suggest higher host read latency.
parbs_tarb_pi_posted_pkts	PCIe traffic	# of PCIe packets transferred on the posted path, which is typically associated with write-like host interface traffic.
parbs_tarb_pi_posted_blocked	PCIe traffic	# of cycles in which the posted PCIe path is blocked. Large blocked-cycles per packet suggest possible congestion at this endpoint.
pct_sct_timeouts	Timeout	# of close requests that timed out before the matching response arrived.
pct_spt_timeouts	Timeout	# of ordinary requests that timed out before a response arrived.
pct_tct_timeouts	Timeout	# of timeout events associated with the TCT tracking or retry path.
hni_pkts_recv_by_tc_{0..7}	Traffic volume	# of packets received per traffic class (TC0–TC7).
hni_pkts_sent_by_tc_{0..7}	Traffic volume	# of packets transmitted per traffic class (TC0–TC7).
hni_rx_paused_std	Pause / flow control	# of cycles in which at least one PCP pause condition was present.
hni_rx_paused_{0..7}	Pause / flow control	# of cycles in which pause is applied on the receive path for a given traffic class. This indicates that incoming traffic is arriving faster than the endpoint can absorb it (TC0–TC7).
hni_tx_paused_{0..7}	Pause / flow control	# of cycles in which the NIC asserts pause on the transmit path for a given traffic class. This indicates the NIC is pushing back because host-side write or translation handling cannot keep up (TC0–TC7).
hni_pcs_good_cw	Codewords	# of received codewords with no detected error.
hni_pcs_corrected_cw	Codewords	# of received codewords that were corrected successfully.
hni_pcs_uncorrected_cw	Codewords	# of received codewords that could not be corrected.
pct_trs_replay_pend_drops	Replay behavior	# of drops while traffic is pending replay in the TRS retry path.
hni_llr_tx_replay_event	Replay behavior	# of link-level replays. Higher rates indicate the link reliability mechanism is actively retransmitting data to protect against link problems.

Chapter 3: Related Work

This chapter reviews prior work most closely related to this dissertation. It focuses on three areas: resource-usage analysis for workload characterization, performance variability in HPC systems, and the use of telemetry, logs, and scheduler records for prediction and system management.

3.1 Resource Usage Analysis

Several monitoring tools collect performance data from HPC systems, including LDMS [2], ParMon [15], and SuperMon [80]. These tools typically monitor compute nodes continuously and collect time-series data about the jobs that run on them. They can report CPU, GPU, memory, and I/O usage, as well as power and energy consumption. Perlmutter uses LDMS for system monitoring and provides GPU-level time-series data for every job that runs on the compute nodes.

Many studies characterize the performance and resource utilization of HPC and cloud systems through monitoring data. Most studies focus primarily on CPU and memory utilization to characterize workloads [3, 21, 73, 17, 51, 66, 67, 25, 40, 12]. For example, Peng et al. [64] analyze memory utilization on two HPC systems and investigate spatial and temporal imbalance in jobs. Similarly, several papers analyze temporal and spatial characteristics of jobs on Perlmutter by using counters such as CPU utilization, GPU utilization, and memory usage [46, 70].

Several studies also focus on I/O, storage, power, and energy usage [73, 61, 67, 3, 46, 36, 19]. Patel et al. [61] analyze I/O behavior in large-scale storage systems. Chu et al. [19] compare the energy profiles of machine learning and traditional workloads and find that machine learning jobs have higher power usage and failure rates. Ilager et al. [36] analyze energy and temperature in cloud data centers. Hu et al. [35] characterize deep learning workloads across multiple data centers and find that single-GPU jobs have a smaller effect on cluster usage than multi-GPU jobs.

These studies establish the value of monitoring data for workload characterization. However, most prior studies rely on coarse resource metrics or examine only a small subset of counters. As a result, they provide only a limited view of GPU workload behavior. This dissertation addresses that limitation by using a much richer set of GPU hardware counters across two telemetry datasets from 2023 and 2025. We use these counters to investigate multiple aspects of workload behavior, including energy use, compute-bound and memory-bound behavior, imbalance, floating-point pipeline activity, differences between ML and non-ML jobs, and relationships among counters and GPU utilization.

3.2 Performance Variability Analysis

Performance variability in HPC systems has been studied for many years. Early work identifies operating system noise as one source of variability [65]. Later studies report that shared network interference and congestion can strongly affect run-to-run behavior in communication-heavy applications [11, 37, 9]. Production studies on Aries and Cray systems relate slowdowns to network conditions and analyze their causes in practice [31, 20]. Other work uses system-wide telemetry, scheduler data, and placement or utilization context to study slowdown behavior at

scale [10, 43].

On modern GPU supercomputers, both GPU-level effects and network conditions can contribute to performance variability [77, 89]. Wei et al. [85] examine variability in GPU codes on production supercomputers and identify network conditions as a major source of slowdown. Nichols et al. [58] further show that historical telemetry can help predict variability, although their approach relies on application-specific knowledge. These studies establish that performance variability is a persistent property of production HPC systems and that shared system state often plays a central role.

These studies establish that performance variability is a persistent property of production HPC systems and that shared system state can affect run time. However, they do not make variability prediction practical for arbitrary production GPU workloads. Existing work either focuses on explaining slowdown or relies on application-specific knowledge. This dissertation addresses that gap by grouping similar runs with scheduler metadata, defining a group-specific variability value, and building prediction models for arbitrary production GPU runs.

3.3 Telemetry-based Prediction

Researchers use job records, logs, and monitoring data to predict many operational quantities, including runtime, I/O behavior, wall-time underestimation, and other resource demands [50, 1, 33, 84, 26, 78, 38, 69]. Of particular relevance to this dissertation, some studies use monitoring data to predict performance variability [58, 10]. Others use monitoring data for application fingerprinting and workload identification [45, 6, 72, 23, 34]. Additional work uses monitoring data to predict job runtime [50, 1, 22] and resource usage [26, 78, 29]. Monitoring data also sup-

port operational analytics, system characterization [29, 55, 57, 19], and anomaly detection [53, 56, 14, 71, 52, 60]. These studies illustrate the broader value of monitoring data for operational analytics in HPC [55, 57].

Several studies connect predictive models to scheduling and control. These approaches use runtime or resource predictions to guide placement, backfilling, co-location, or system-level adaptation [87, 81, 86, 62, 42, 88, 44, 13]. Uncertainty-aware models also appear in this literature because prediction error can strongly affect scheduling quality [32, 27, 83]. Related work also studies phase detection and phase-aware modeling to identify intervals with distinct resource-usage patterns and to support runtime prediction [18, 48, 63, 28, 49, 5].

These studies demonstrate that monitoring data can support a wide range of predictive and operational tasks. However, they do not establish whether scheduler metadata and system-wide telemetry together can predict run-to-run slowdown for arbitrary GPU applications at or shortly after run start. They also do not quantify how telemetry scope and timing affect this task. This dissertation addresses these gaps by building slowdown prediction models without code instrumentation, application profiling, or pre-run benchmarking, and by comparing allocated-node versus all-node telemetry data and pre-start versus post-start windows.

Chapter 4: Job-level Characterization of GPU Workloads from System-wide Monitoring Data

This chapter presents a job-level characterization of GPU workloads on Perlmutter using system-wide monitoring data. It synthesizes two studies that analyze telemetry collected in 2023 and 2025. The first study uses a longitudinal dataset that spans August 16 to December 13, 2023 and characterizes utilization and its spatial and temporal variability across a large population of production jobs. The second study analyzes a representative one-month dataset from March 1 to April 1, 2025 and extends the earlier analysis with diagnostics that relate job behavior to practical workload characteristics such as compute and memory limitations, energy consumption, and HBM capacity headroom. This chapter builds primarily on my paper [16] and includes findings from both studies.

Both studies use GPU telemetry collected on Perlmutter through LDMS [2] and its DCGM plugin [59]. The underlying data format and collection pipeline are the same in both studies, and each telemetry sample records a timestamp together with node and GPU identifiers and the selected DCGM counter values. The studies differ in time coverage and in the breadth of counters analyzed. The 2023 study focuses on utilization, SM activity, floating-point pipeline activity for FP16, FP32, FP64, and tensor pipes, and HBM usage over a longitudinal window of ~four months. The 2025 study uses the same data format and sampling cadence, but expands the

counter set to include additional signals such as NVLink and PCIe traffic, energy consumption, and temperature, and analyzes a representative one-month window. When the two studies address the same question and yield similar conclusions, this chapter reports the result from the later study and uses the earlier study when it provides additional context or a complementary perspective.

The chapter begins with an introduction to the problem setting and the main analytical themes. It then describes the monitoring data, data integration steps, and preprocessing choices that produce a reliable job-attributed telemetry dataset. Next, it presents the methodology used to characterize job behavior. The remaining sections summarize the main findings on overall workload behavior, job types, spatial and temporal behavior, and relationships among hardware counters, and then discuss their implications.

4.1 Introduction

General-purpose Graphics Processing Units (GPGPUs) have become pervasive in compute nodes on high-performance computing (HPC) systems. Understanding GPU resource usage can inform application optimization, facility operation, and future architecture design. System-wide monitoring datasets make this analysis possible because they capture GPU floating-point (FP) pipe activity, HBM usage, interconnect traffic, and energy consumption together with job metadata such as start and end times and GPU counts.

Although modern GPUs expose a rich set of hardware counters, operators and users rarely exploit this breadth. This underuse creates performance blind spots. Users struggle to diagnose underperformance and to choose effective optimizations for their jobs, and system operators lack the observability needed for informed placement, capacity planning, and inefficiency detection

at scale. This chapter addresses that gap by translating system-level hardware counter telemetry into interpretable, job-level insights that inform both users and operators.

Producing valid job-level insights from system-wide monitoring data requires a rigorous curation and analysis workflow. Reliable analysis begins with precise alignment of each GPU sample to its job metadata. Hardware counters also differ in semantics: some report fractional activity, others provide instantaneous readings, and others accumulate totals over time. The workflow must therefore enforce correct interpretations, handle missing values and outliers, and apply appropriate aggregations to produce job-level summaries that support statistically defensible comparisons.

While some prior work analyzes GPU monitoring data [19, 46, 41], most studies consider only a limited set of hardware counters. This chapter addresses that limitation by synthesizing two complementary studies of GPU-specific counters collected through the Lightweight Distributed Metric Service (LDMS) on Perlmutter, a flagship open-science supercomputer at the National Energy Research Scientific Computing Center (NERSC). The first study spans August 16 to December 13, 2023 and covers more than 300,000 jobs on the GPU partition. The second spans March 1 to April 1, 2025 and covers around 75,000 jobs. Both datasets are sampled every 10 seconds and include NVIDIA Data Center GPU Manager (DCGM) counters. The 2023 study focuses on a core set of signals, including utilization, streaming multiprocessor activity, floating-point pipe activity, and frame-buffer (HBM) usage. The 2025 study extends this set with additional counters such as NVLink and PCIe traffic, energy, and temperature.

This chapter analyzes how GPU workloads use resources across GPUs and over time. It quantifies how evenly work within a job is distributed across its assigned GPUs and how steadily each GPU is exercised during the job. It also classifies jobs as compute-bound or memory-

bound using a roofline-based criterion and relates these job types to utilization patterns, energy consumption, and imbalance. In addition, it examines usage of FP pipes (FP16, FP32, FP64, and tensor), HBM memory, and the NVLink and PCIe interconnects, and analyzes correlations between hardware counters and GPU utilization to identify factors associated with higher GPU use. The chapter also compares ML and non-ML jobs and concludes with implications for users and system operators.

This chapter makes the following contributions:

- It presents a system-wide characterization of GPU workloads on Perlmutter using two complementary telemetry datasets from 2023 and 2025 together with a robust job-attribution pipeline, which enables job-level analysis of diverse aspects of GPU usage.
- It labels jobs as compute-bound or memory-bound using a roofline criterion and observes that, at comparable GPU-hours, memory-bound jobs tend to consume more energy than compute-bound jobs.
- It introduces a time-windowed spatial imbalance metric that captures uneven resource usage across the GPUs assigned to a job and enables minute-scale detection of uneven load and more reliable job-level summaries.
- It classifies jobs by FP pipeline activity and quantifies how GPU utilization and spatial and temporal imbalance vary across job classes, and it demonstrates that tensor activity is typically associated with higher utilization.
- It compares ML and non-ML jobs to identify differences in GPU usage behavior and workload characteristics.

- It uses a broad set of system-wide DCGM counters to quantify job-level relationships among hardware counters and to identify which counters co-vary with GPU utilization and how these associations differ between compute-bound and memory-bound jobs.

4.2 Monitoring Data Used in this Chapter

The data format is the same in both studies synthesized in this chapter. Each LDMS record contains a timestamp, a node identifier, a GPU identifier, and the selected DCGM counter values. The studies differ in time coverage and in the breadth of counters analyzed. The 2023 study uses a longitudinal dataset that spans August 16 to December 13, 2023 and covers 345,154 jobs. The 2025 study uses a one-month dataset that spans March 1 to April 1, 2025 and covers 75,703 jobs. The 2023 study focuses on a core set of counters that capture GPU utilization, SM activity, floating-point pipeline activity, copy-engine activity, and HBM usage. The 2025 study uses the same core signals and expands the counter set. The facility configures this rate, and users cannot change it. This chapter uses only DCGM counters that Perlmutter staff validate for operational monitoring because hardware counters can be noisy [68].

4.2.1 Data Retrieval and Integration

This chapter retrieves LDMS data through the Prometheus API and Slurm job data through the `sacct` command and stores both in Parquet files. LDMS provides per-GPU hardware counter values together with timestamps, node identifiers, and GPU identifiers, but it does not provide job identifiers. Slurm provides job-level metadata, but it does not include hardware counter measurements.

To enable per-job GPU-level analysis, the workflow merges the datasets by aligning LDMS samples with Slurm job data. It matches LDMS timestamps to jobs running on the same node by using start and end times from Slurm. For each job, it assigns LDMS samples within the job time range to the corresponding job identifier and step. This process produces a dataset in which each LDMS sample is linked to a specific job and step.

4.2.2 Data Filtering and Preprocessing

Both studies apply a common set of filtering and preprocessing steps to ensure reliable analysis of the merged LDMS and Slurm datasets. First, the workflow removes LDMS entries that cannot be associated with any job during the merge. It excludes jobs that run on login nodes because they do not represent meaningful workloads, and it removes jobs that do not use the GPU partition. It also excludes jobs submitted by staff accounts because these jobs often involve maintenance, testing, or debugging rather than production workloads.

In addition, the workflow removes jobs with durations shorter than three minutes because they usually do not contain enough data for meaningful analysis. It applies counter-specific sanity checks to address data inconsistencies. For example, it removes jobs with counter values that exceed physical limits for the device, such as GPU_UTIL $>100\%$. These samples are extremely rare, accounting for less than 0.0001% of all records. They likely result from counter overflow, so the workflow removes them. Finally, it excludes jobs with a mean GPU_UTIL below 1%.

The 2023 study also identifies ML jobs through a coarse heuristic based on submit-line information in Slurm job scripts. It classifies a job as ML-related if its submit line contains any of 18 predefined keywords.[1] For the remainder of this chapter, we refer to these jobs as ML

[1] Keywords include ‘epoch‘, ‘training‘, ‘neural‘, ‘cnn‘, ‘rnn‘, ‘lstm‘, ‘transformer‘, ‘bert‘, ‘tensorflow‘, ‘pytorch‘,

jobs. This keyword-based labeling may undercount ML jobs that do not expose these terms in their submit lines and may overcount jobs that include these terms without actually using ML frameworks. Despite these limitations, the heuristic is sufficient for a coarse-grained ML versus non-ML comparison in the 2023 analysis.

4.3 Methodology

Our goal is to convert raw, per-sample GPU telemetry into interpretable job-level insights. We (i) classify job type (compute- versus memory-bound) using roofline-based sample labels aggregated to the job level, (ii) measure spatial imbalance to quantify how evenly work is distributed across allocated GPUs, and (iii) measure temporal imbalance to quantify how steadily a job sustains activity over time. We then relate these metrics to job-level counter means using heatmaps and Spearman correlations.

Together, these views support user diagnosis (e.g., properly sizing allocations) and operator decisions (e.g., placement under HBM headroom, expectations for power/thermals). The subsections that follow formalize each metric, define aggregation and normalization rules, and detail how we compute job-level summaries and correlations from the counters in Table 2.1.

4.3.1 Classifying Job Types

The roofline model identifies whether compute or memory activity limits a kernel. It plots each kernel with arithmetic intensity on the x-axis and achieved flop/s on the y-axis. Arithmetic intensity is FLOPs per byte moved from HBM. Achieved rate is the measured flop/s during execution. Two limit lines appear on the same chart: a flat line at peak flop/s and a rising line

‘keras‘, ‘deep‘, ‘inference‘, ‘autoencoder‘, ‘classification‘, ‘detection‘, ‘activation‘, ‘sklearn‘.

equal to peak HBM bandwidth times arithmetic intensity. Their intersection is the ridge. Points left of the ridge are memory-bound, and points at or right are compute-bound. At any x-value, the lower limit sets the attainable flop/s.

We perform a roofline-based analysis, building on [7], to label each telemetry sample as compute-bound or memory-bound and then aggregate these labels at the job level. The method uses the floating-point (FP) pipeline activity fractions (FP16_ACTV, FP32_ACTV, FP64_ACTV, TNSR_ACTV) and the memory activity fraction (DRAM_ACTV) listed in Table 2.1, together with peak device capabilities. We take peak flop/s and peak HBM bandwidth from the device specifications. Because the HBM peak depends on GPU capacity (40 GB or 80 GB), we infer the capacity from node identifiers.

For each sample, achieved flop/s and HBM bandwidth are estimated by scaling the architectural peaks with the observed activity fractions. For example, the achieved FP64 flop/s and HBM bandwidth are estimated as follows:

$$F64 = \text{FP64\_ACTV} \times \text{peak FP64 flop/s} \tag{4.1}$$

$$B = \text{DRAM\_ACTV} \times \text{peak HBM BW} \tag{4.2}$$

The per-sample arithmetic intensity is then

$$AI_{\text{fp64}} = \frac{F64}{B} \tag{4.3}$$

FP64 arithmetic intensity

achieved FP64 flop/s

achieved HBM bandwidth

We compare AI_{fp64} to the device balance point (ridge) and classify a sample as compute-bound if AI_{fp64} exceeds the balance point, and as memory-bound otherwise. The balance point is defined as the ratio of the device's peak compute throughput (flop/s) for the FP pipeline of interest to the device's peak memory bandwidth (bytes/s). Finally, we categorize each job by the fractions of its compute-bound and memory-bound samples.

4.3.2 Quantifying the Spatial Imbalance of Jobs

Spatial behavior refers to how a job's workload is distributed across its assigned GPUs. We extend a prior spatial imbalance metric [46, 64] with a time-windowed approach that captures usage throughout the job rather than over its full runtime at once. This time-windowed spatial imbalance metric is useful because it quantifies how evenly a job's GPU usage is distributed over time.

Let $TC(g, w) = \sum_{t=1}^{t_w} C_{g,t}$ be the sum of hardware counter values $C_{g,t}$ over all timestamps in time window w for GPU g. We define the spatial imbalance of job j in w as

spatial imbalance in window w

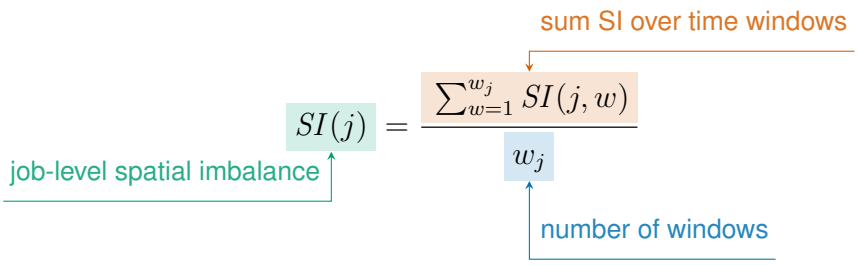
total activity in window w

$$SI(j, w) = 1 - \frac{\sum_{g=1}^{g_j} TC(g, w)}{g_j \times \max\limits_{1 \le g \le g_j} TC(g, w)} \tag{4.4}$$

max possible total given busiest GPU

The numerator in Equation 4.4 sums the per-GPU hardware counter values within time window w across all GPUs allocated to the job where g_j is the number of GPUs allocated to j. The denominator represents the maximum possible counter value if all GPUs matched the highest observed value in that window. An SI value near 0 indicates minimal imbalance; a value near 1

indicates significant imbalance. We define the spatial imbalance metric for a job over its entire runtime as the mean spatial imbalance across all time windows, where w_j is the total number of time windows:



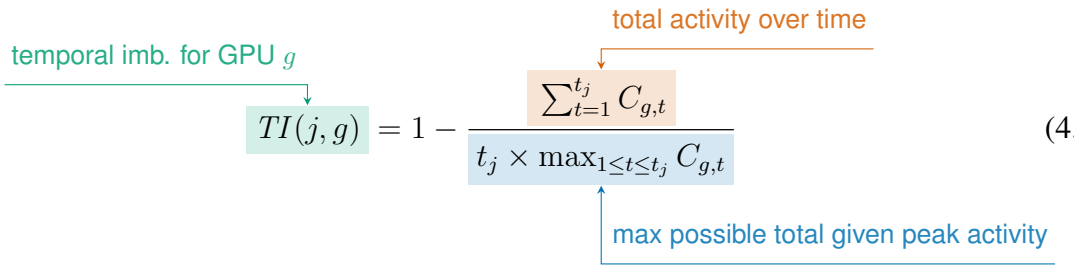
(4.5)

A longer time window smooths out short bursts by averaging them over a longer period. In contrast, a shorter window retains these quick changes and makes them more visible in the metric. Window length therefore sets the time scale of the imbalance we capture. We keep the window length fixed when comparing jobs so the metric is consistent across workloads.

4.3.3 Quantifying the Temporal Imbalance of Jobs

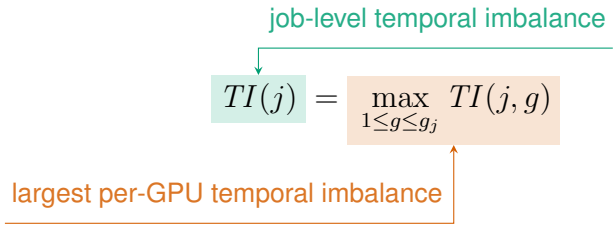
Temporal imbalance quantifies the variation in hardware counter values over a job's runtime. We adopt the definition from [64]. Here, $C_{g,t}$ is the hardware counter value for GPU g at time t. The numerator sums the counter values over time. The denominator represents the maximum possible value and is calculated by multiplying the job duration by the peak observed counter value for that GPU. Subtracting from 1 highlights the imbalance, where values near zero

indicate stable behavior and higher values reflect greater fluctuations.



(4.6)

Equation 4.6 defines the temporal imbalance metric for a single GPU. We define the temporal imbalance of a job as the maximum imbalance across all allocated GPUs, where g_j is the number of GPUs assigned to the job:



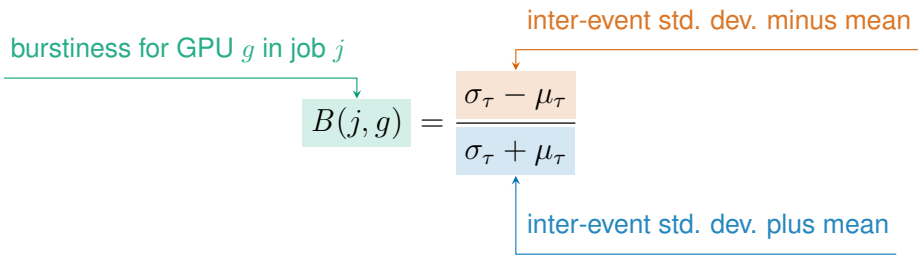
(4.7)

4.3.4 Quantifying Burstiness

Temporal imbalance compares average behavior to the peak observed value and can be high when a job has brief spikes but low average activity. To capture irregularity of sudden changes that is not reflected by temporal imbalance, the 2023 study uses a burstiness metric based on inter-event times [30].

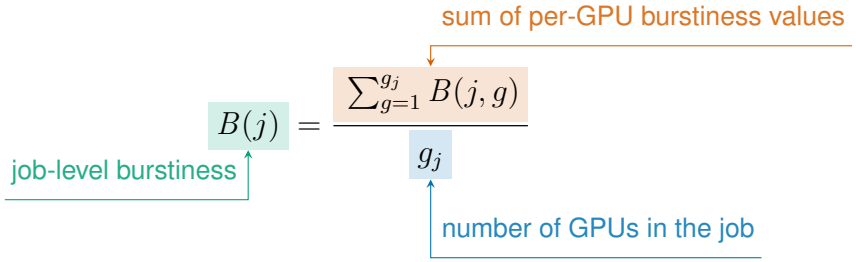
An event is defined as a counter value increasing by more than 15 percentage points within a 10-second interval. For example, if GPU_UTIL is 50% at a given timestamp, an event occurs if the next measurement exceeds 65%. Inter-event times are the time intervals between consecutive

events. Let τ denote the inter-event times for GPU g in job j. Burstiness is defined as



(4.8)

where σ_τ and μ_τ are the standard deviation and mean of the inter-event times. A value of -1 indicates perfectly regular behavior, a value of 0 indicates balanced fluctuations, and a value near 1 indicates highly irregular changes.

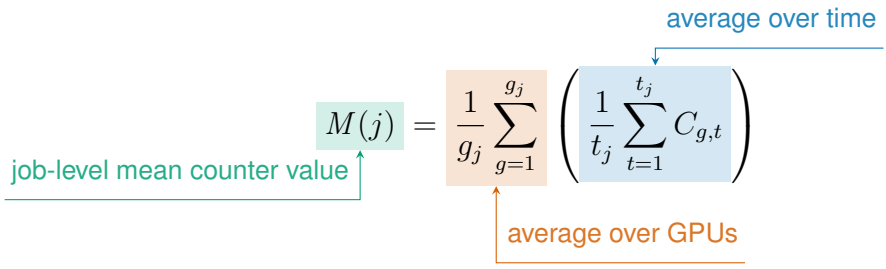


(4.9)

Equation (4.8) defines burstiness for a single GPU. The overall burstiness of a job is the mean across its GPUs.

4.3.5 Correlating Job-Level Counter Values

We analyze correlations among job-level means of hardware counters to characterize per-job resource-usage behavior. We compute the job-level mean, $M(j)$, by averaging the counter values over time, t_j, for each GPU in a job, and then taking the mean across all GPUs, g_j, assigned to that job:



$$M(j) = \frac{1}{g_j} \sum_{g=1}^{g_j} \left(\frac{1}{t_j} \sum_{t=1}^{t_j} C_{g,t} \right) \tag{4.10}$$

where $C_{g,t}$ is the counter value for GPU g at time t.

To quantify relationships without assuming linearity or normality, we use Spearman's rank correlation. We compute pairwise correlation across jobs and report the full matrix. We compute all correlations on the same job set used throughout this chapter.

4.4 Overview of GPU Workloads on Perlmutter

In this section, we present an overview of our monitoring data and the job-level characteristics of Perlmutter. Our analysis examines job size, utilization patterns, usage of different GPU FP pipelines, HBM capacity used, and interconnect activity. These aspects are central to understanding how workloads use system resources.

4.4.1 Overall Resource Usage

Job size is an important factor that shapes workload composition. On Perlmutter, most jobs are small, while large jobs consume a disproportionate share of resources. Relating job size to job count, duration and GPU_UTIL provides a baseline characterization of job behavior on the GPU partition.

The left plot in Figure 4.1 demonstrates that 64,183 jobs (85%) allocate four GPUs (a single node). Each GPU node has four GPUs on Perlmutter. There are no jobs in the first bin (1–2

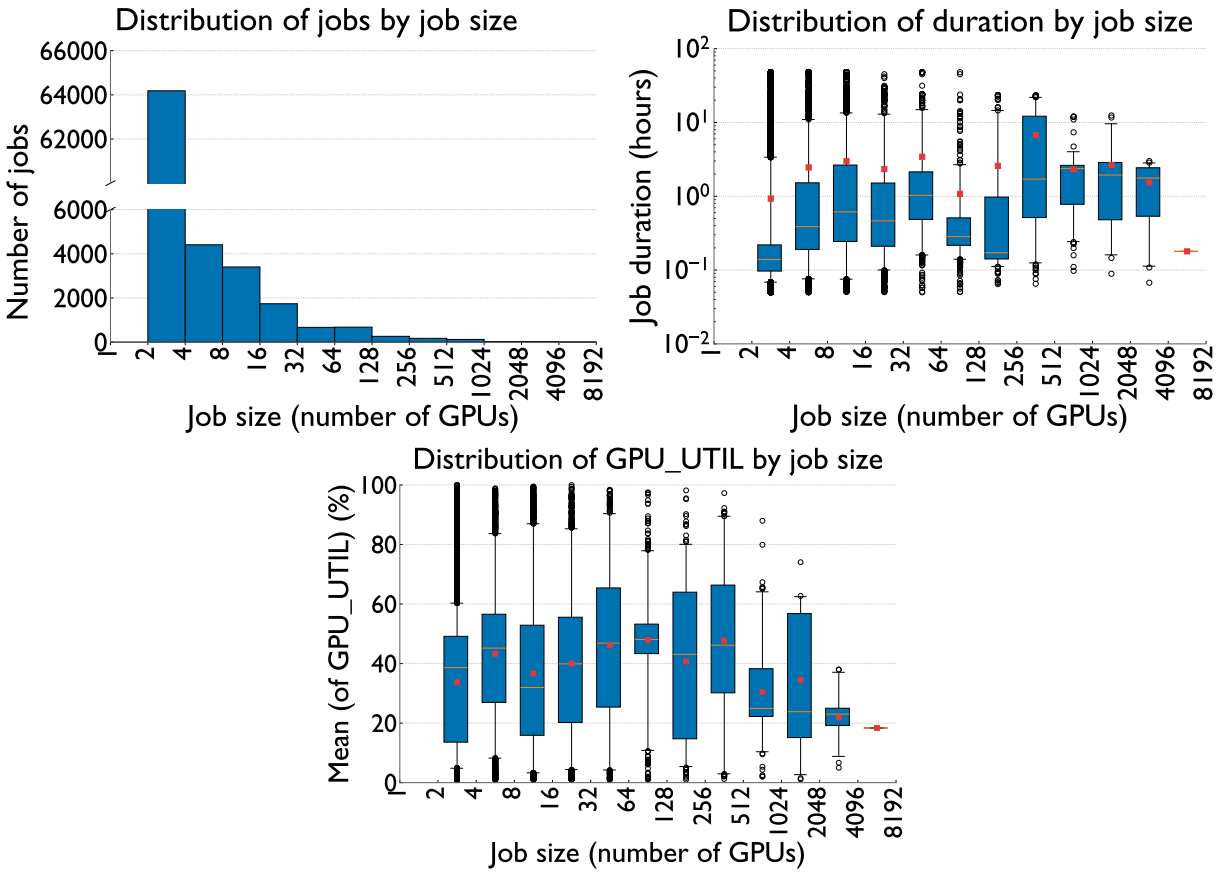


Figure 4.1: Left: number of jobs by job size (GPUs). A y-axis break is used to display the tall first bin while preserving readability of subsequent bins. Each bar represents a range of job sizes (e.g., the 8-16 bin includes all jobs that requested more than 8 and up to 16 GPUs). Middle: distribution of job duration (hours on log scale) by job size. Right: distribution of per-job mean GPU_UTIL (%) by job size. Red dots on the box plots indicate means, orange lines indicate medians, and open circles denote outliers.

GPUs) because even when users request fewer than four GPUs, Slurm allocates all four GPUs on the node on Perlmutter. The job size in the subsequent bins decreases as job size increases.

The middle plot in Figure 4.1 demonstrates the job durations for different ranges of job sizes. The circles in the box plot represent outliers. The area within the box represents the interquartile range (IQR), defined as the range between the 5th and 95th percentiles. Whiskers above and below the box extend to the last data point within $1.5 \times IQR$, and outliers are the points beyond the whiskers. This plot indicates that single-node jobs are shorter on average (~55

min) compared to the multi-node jobs. Multi-node jobs have varying durations, but most multi-node size ranges have a mean value of ~two hours. Across sizes, the IQR is roughly 1–3 hours. The absence of jobs longer than 48 hours reflects the maximum duration allowed by Perlmutter's queue policies.

The right plot in Figure 4.1 summarizes the per-job mean GPU_UTIL by job size, using the same box-plot conventions as the middle plot. The job count distribution skews toward small jobs (1–4 GPUs), yet means of GPU_UTIL improve at moderate scales and peak for 33–512 GPUs (~47–48%). At very large scales (≥512 GPUs), medians drop to ~20% with a reduced 95th percentile, plausibly due to extended idle time from communication and synchronization in larger jobs. Small jobs (1–4 GPUs) have lower medians than 5–8 GPUs (38% vs 45%), because exclusive placement leaves more allocated GPUs unused in the smallest bin.

> **Observations** Overall, we find that small node-count jobs dominate Perlmutter's workload. Median runtimes are shorter for single-node jobs compared to multi-node jobs. Typical durations for multi-node jobs are around 1–3 hours. Overall, GPU utilization is highest at intermediate job sizes (33–512 GPUs) and lower at the largest sizes (≥512 GPUs).

4.4.2 GPU Floating-Point Pipe Activity

Beyond overall GPU utilization, we analyze GPU FP pipe activity (FP16_ACTV, FP32_ACTV, FP64_ACTV, TNSR_ACTV) to assess how workloads drive the compute pipelines. We address the following research question:

> **RQ1** *Are there differences in how GPU jobs utilize FP16, FP32, FP64, and Tensor pipes?*

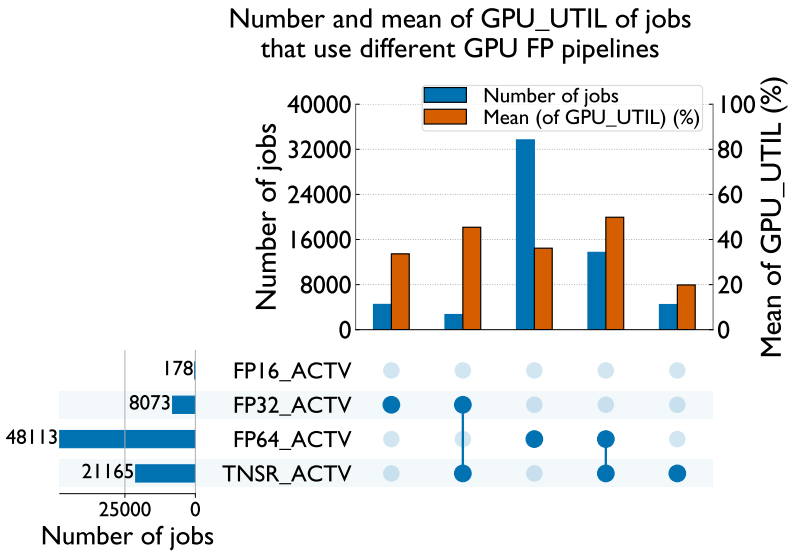


Figure 4.2: The number and mean GPU_UTIL of jobs using different combinations of GPU floating-point pipes used. Each bar represents a disjoint set of jobs. Filled circles mark which FP pipes are active in that set.

Figure 4.2 demonstrates how much GPU jobs use different FP pipelines and how they co-occur. The top bar chart reports the number and mean GPU_UTIL of jobs for each combination of active counters. The bottom matrix with filled circles and connecting lines denotes the counters present in each combination. Each top bar represents a disjoint set of jobs. To determine which FP pipe a job engages, we mark an FP pipe as used if the job's mean activity for that pipe exceeds a small threshold. The threshold is the larger of the 5th percentile of per-job means for that pipe and 0.005.

FP64 dominates Perlmutter's workload. The FP64-only intersection contains 33,675 jobs (44% of all jobs) with a mean GPU_UTIL of 36%. Tensor+FP64 is the next largest intersection with 13,721 jobs (18%) and a higher mean GPU_UTIL of 50%. FP32-only accounts for 4,485 jobs (6%) at 34% mean GPU_UTIL. Tensor+FP32 includes 2,694 jobs (4%) with 46% mean GPU_UTIL. The Tensor-only group appears with 4,463 jobs (6%) and mean GPU_UTIL of 20%. Because of our small threshold, very small means of activity can fall below the threshold for

some jobs and those jobs then surface as Tensor-only. Finally, FP16 is very rare on Perlmutter (178 jobs).

> **Answer to RQ1** The FP64-only group is the largest (44% of jobs), which indicates that many workloads utilize FP64 pipes on Perlmutter. In addition, jobs that utilize tensor pipes alongside FP64 or FP32 tend to sustain the highest GPU utilization (50% and 46%, respectively), which suggests that utilizing tensor cores tends to improve GPU utilization.

4.4.3 Peak HBM Usage on 40 GB and 80 GB GPUs

Memory (HBM) capacity is a critical constraint for GPU jobs. We analyze peak HBM usage to assess how close jobs run to hardware limits and whether 80 GB GPUs are fully utilized. We address the following research question:

> **RQ2** *Among the jobs that request 80 GB GPUs, how much HBM do they actually use?*

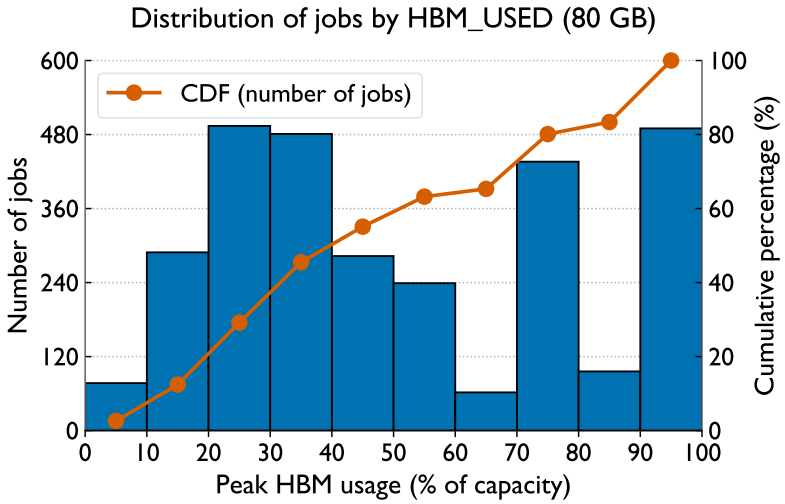


Figure 4.3: Distribution of peak HBM usage (normalized by capacity) for jobs that explicitly requested 80 GB GPUs. The orange line represents the CDF.

Figure 4.3 summarizes job-level peak HBM usage (HBM_USED) for jobs that explicitly requested 80 GB GPUs. For each GPU, we compute its peak usage as the maximum HBM_USED sample over the job's duration divided by the GPU's nominal capacity. The job-level peak is the maximum of these per-GPU peaks. The figure demonstrates a histogram of the resulting percentages with a CDF on the secondary y-axis.

On Perlmutter, jobs that do not request a specific memory capacity can run on either 40 GB or 80 GB GPUs. To reflect intent, we isolate the 2,947 jobs that explicitly requested 80 GB GPUs with `-C gpu&hbm80g`. Figure 4.3 reports the distribution of job-level HBM usage: 12% of jobs use less than 20% of capacity, 55% use less than 50%, 20% use at least 80%, and 17% fall in the 90–100% range. Because HBM_USED is sampled every 10 seconds, brief spikes may be missed, therefore a safety margin should be applied when interpreting peaks.

This analysis indicates that system administrators can use historical telemetry to help users request the right amount of memory. At submission time, the system can check a user's recent jobs. If a user requests 80 GB but prior runs on 80 GB GPUs peaked below 40 GB, the system can display an advisory prompt to consider a 40 GB GPU instead. This may reduce wait times on scarce 80 GB GPUs and improve overall throughput.

> **Answer to RQ2** Among jobs that explicitly request 80 GB GPUs, 55% peak at or below 50% of HBM capacity and should fit on a 40 GB GPU. Meanwhile, 20% of jobs exhibit good capacity use with HBM usage of at least 80%.

4.4.4 Comparison Between ML and non-ML Jobs

> **RQ3** *Is there a difference between machine learning and non-machine learning jobs in terms of their resource utilization?*

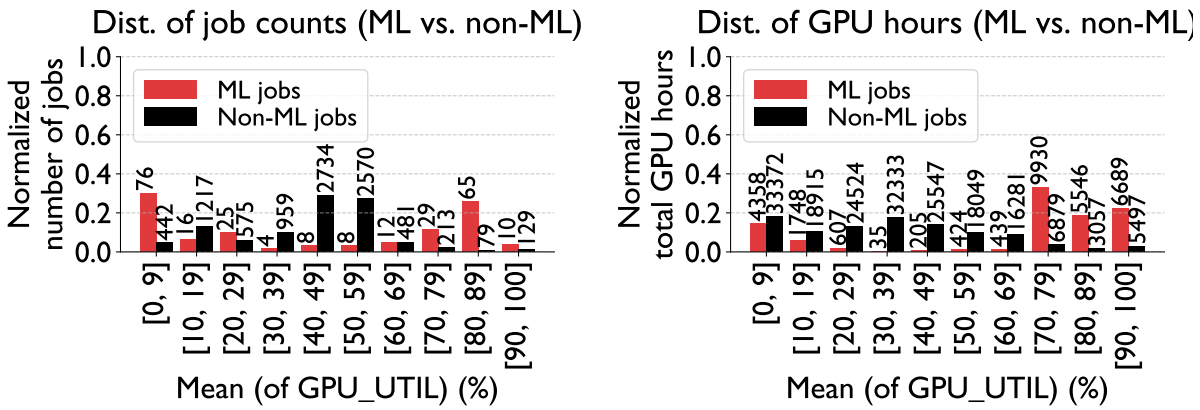


Figure 4.4: The plots compare the number (left) and GPU hours (right) of ML and non-ML jobs by mean of GPU_UTIL. ML jobs dominate high (70-100%) and the low (0, 9%) utilization bins. They account for more GPU hours. Absolute values are annotated above bars.

Figure 4.4 illustrates the distribution of normalized job counts and GPU hours for ML and non-ML jobs across mean of GPU_UTIL ranges. We normalize job counts and GPU hours separately to compare the relative distributions of ML and non-ML jobs. For job counts, we group jobs by mean of GPU_UTIL and compute each group's fraction relative to the total jobs in its category (ML or non-ML). For GPU hours, we sum the total GPU hours in each group and normalize by the total GPU hours in that category. The baselines are total jobs and total GPU hours per category. Absolute values are annotated above bars.

The left plot shows normalized job counts by mean GPU_UTIL ranges. Non-ML jobs dominate the lower- and mid-utilization ranges (except [0-9%]), but decline significantly at higher utilizations. ML jobs are most common in high-utilization ranges (70-100%) and the lowest

utilization bin. ML jobs comprise 2.62% of all jobs. The right plot in Figure 4.4 presents the normalized GPU hours. ML jobs dominate the 70-100% range, while non-ML jobs contribute most in 0-39%. ML jobs are underrepresented in the 30-70% GPU utilization range, likely because they either involve minimal computation (e.g., tuning or small-scale experiments) or exhibit high computational intensity that drives utilization above 70%. In contrast, mid-utilization jobs are likely traditional HPC workloads, though further metrics are needed to confirm this.

> **Answer to RQ3** ML jobs are a small fraction of total jobs (2.62%) but account for a disproportionately large share of GPU hours at very high utilization (70–100%). In normalized job counts, non-ML jobs dominate the low and mid utilization bins (except the 0–9% bin), while ML jobs concentrate in the 70–100% bins and also appear prominently in the lowest utilization bin.

4.5 Characterizing Job Types

In this section, we examine the distribution of compute- versus memory-bound samples on Perlmutter and assess how job-level total energy consumption differs by job type. This section addresses the following questions:

> **RQ4** *What share of jobs are classified as compute-bound versus memory-bound under the roofline criterion, and how do these job types differ in total energy consumption?*

Figure 4.5 presents per-sample arithmetic-intensity distributions for the FP64 and tensor pipes on NVIDIA A100 GPUs with 40 GB and 80 GB HBM. We omit the FP32 distribution for brevity because it matches FP64. Each plot reports the fraction of samples per arithmetic-

intensity bin on a logarithmic axis. Balance points (ridges) and roofline ceilings are annotated. Using the FP64-specific arithmetic intensity, we classify 88% of samples as memory-bound and 12% as compute-bound. We assign job-level labels by the majority class among a job's samples. When aggregated by job, 81% of jobs are memory-bound and 19% are compute-bound. We observe similar proportions across the other FP pipes, with ~20% of jobs classified as compute-bound and ~80% as memory-bound.

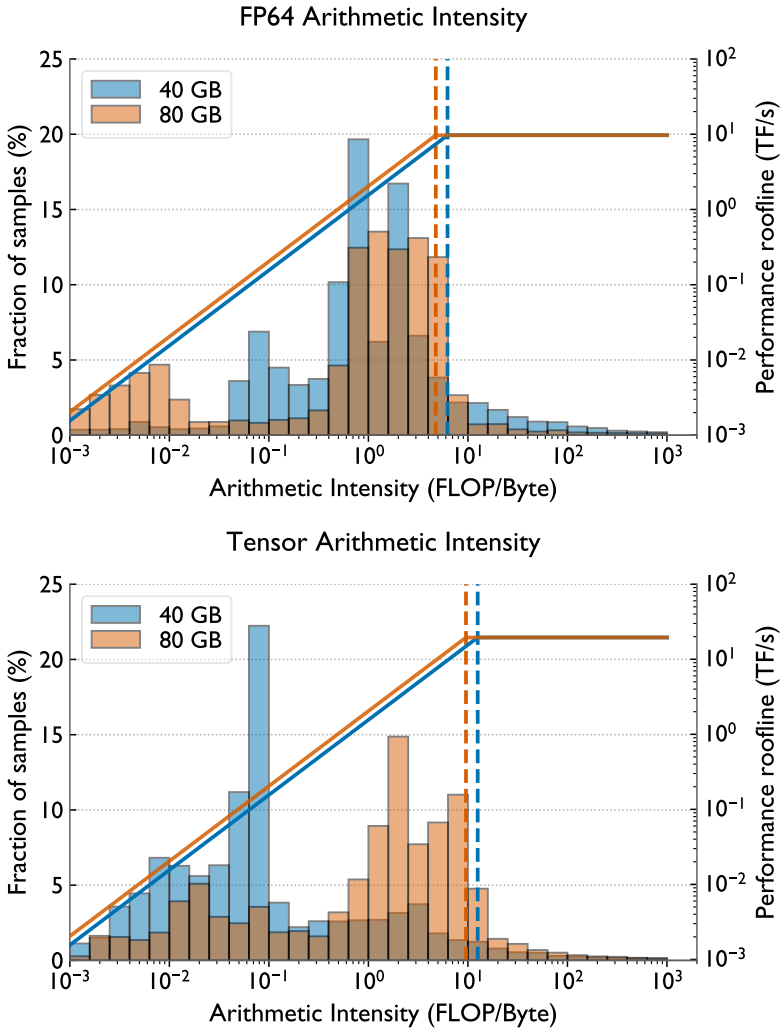


Figure 4.5: Distribution of arithmetic intensity by GPU floating-point pipes (40 GB versus 80 GB GPUs). Bars represent the fraction of samples for FP64 (left) and tensor (right). Vertical lines mark the capacity-specific balance points (ridges), and the right axis overlays the corresponding rooflines.

The separation between 40 GB and 80 GB is most visible for the tensor pipe. Tensor pipe activity is mostly associated with high compute reuse (e.g., matrix multiplication). The extra HBM capacity on 80 GB GPUs supports larger batches or models, which results in more sustained tensor flops per byte.

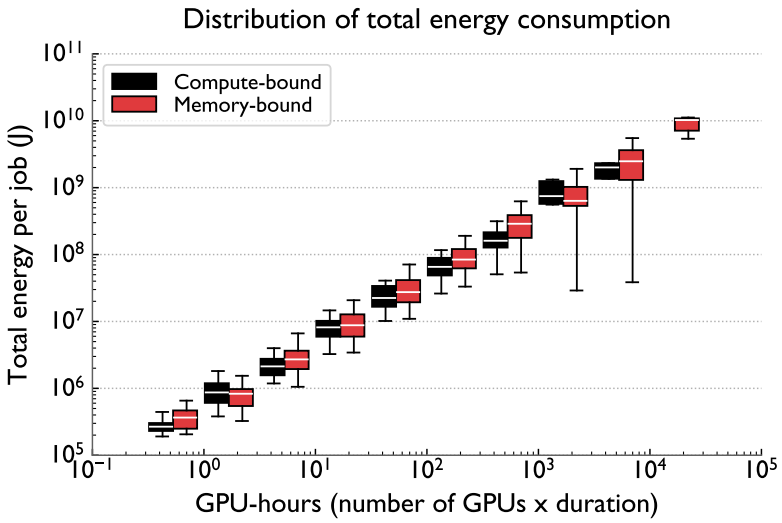


Figure 4.6: Distribution of total energy per job (J) versus GPU-hours with log scales on both axes. For each GPU-hour bin, side-by-side box plots compare compute-bound and memory-bound jobs classified using FP64 arithmetic intensity. GPU-hours = number of GPUs $\times$ duration of the job. White lines in boxes represent median and IQR with whiskers (outliers suppressed).

We next compare total energy consumption of compute- and memory-bound jobs (classified by FP64 arithmetic intensity). In Figure 4.6, each box plot demonstrates the distribution of total energy per job (J) within a GPU-hours bin. For every bin on the x-axis, we place two box plots side by side: one for compute-bound jobs and one for memory-bound jobs so that we can compare two classes at the same GPU-hours scale. Here, we define GPU-hours as the number of GPUs allocated multiplied by the job's runtime in hours.

Two patterns emerge. First, as GPU-hours increase, the entire energy distribution shifts upward and broadens as expected from longer and larger runs. Second, at comparable GPU-hours, memory-bound jobs tend to consume more total energy than compute-bound jobs at a

given GPU-hour scale (their boxes sit higher in the same bins). Antepara et al. empirically observe that off-chip HBM traffic carries a much higher energy cost than arithmetic, which can translate into higher average power for memory-bound workloads and aligns with our observation of higher total energy at comparable GPU-hours [4].

> **Answer to RQ4** The jobs on Perlmutter are overwhelmingly memory-bound (~80% of all jobs). Memory-bound jobs typically consume more energy compared to compute-bound jobs.

4.6 Analyzing the Spatial Behavior of Jobs

This section analyzes spatial imbalance in GPU jobs. We use a fixed one-minute window to define the metric, after comparing one, five, 15, and 30 minutes. With 15–30 minute windows, a brief spike on a single GPU can dominate the entire interval because the metric compares every sample in that interval to the most active GPU. This tends to overstate imbalance in bursty, low-utilization jobs. Shorter windows (1–5 minutes) attribute high imbalance only to the periods when it actually occurs. We therefore use a 1-minute window. We group jobs by mean GPU_UTIL into low (<30%), medium (30–70%), and high (>70%) categories, and we also examine the effects of FP pipeline usage and job type. This section addresses the following questions:

> **RQ5** *Do jobs that utilize multiple GPUs use them evenly, and how does spatial imbalance vary with overall GPU utilization, FP pipes used, and job type (compute- versus memory-bound and ML vs non-ML)?*

Figure 4.7 demonstrates the spatial imbalance distribution of GPU_UTIL across three uti-

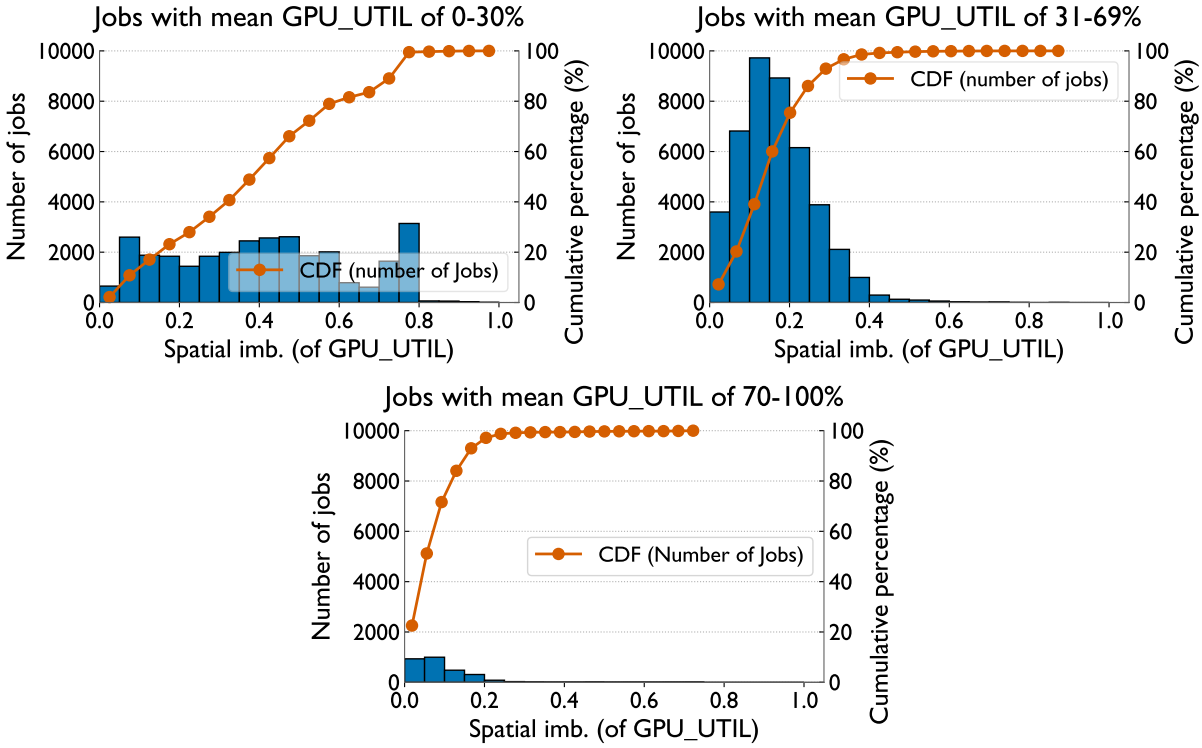


Figure 4.7: The plots demonstrate the distributions of spatial imbalance of GPU_UTIL for jobs grouped by mean GPU_UTIL (0–30%, 31–69%, 70–100%)

lization ranges. Low-utilization jobs (left) exhibit the highest imbalance (peaking at 0.78), which indicates a significant unevenness in resource distribution. We observe that only 34% of jobs in this category fall below 0.30, and 16.5% are at or above 0.70. Medium-utilization jobs (middle) are more balanced (86% fall below 0.30). High-utilization jobs (right) have less spatial imbalance and most of them concentrate in the lower range: a total of 97.1% fall below 0.30.

For further exploration, we analyze the coefficient of variation (CV) of GPU_UTIL across GPUs for jobs with a spatial imbalance of GPU_UTIL greater than 0.5. CV quantifies relative variability and enables comparison across jobs with different mean of GPU_UTIL levels. It is defined as: $CV = \frac{\sigma}{\mu} \times 100$, where σ is the standard deviation and μ is the mean.

The left plot in Figure 4.8 shows the CV of total GPU_UTIL across GPUs for each job. Among jobs with high spatial imbalance (>0.5), 52.4% have a CV exceeding 100%. This indi-

cates substantial variation, where some GPUs are more utilized than others.

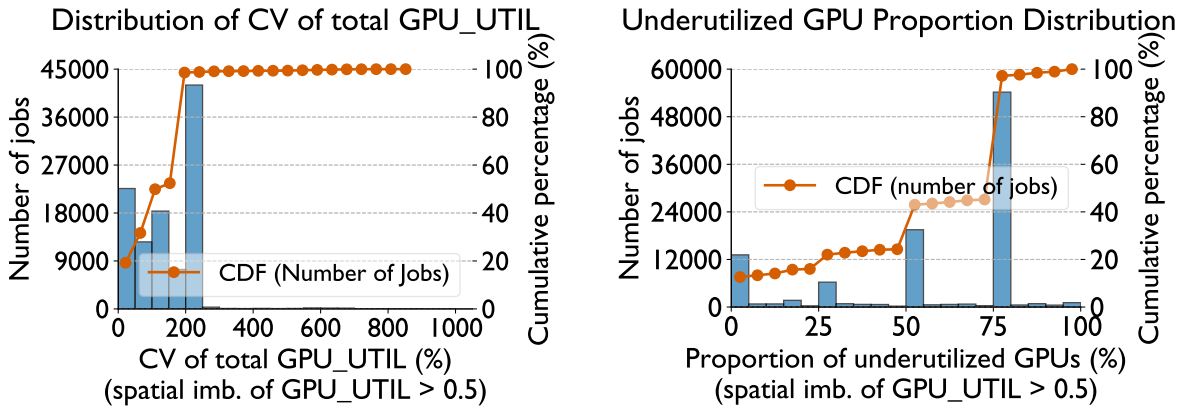


Figure 4.8: The plots demonstrate CV of total GPU_UTIL (left) and underutilized GPU proportions (right) for the jobs with spatial imb. of GPU_UTIL value of greater than 0.5. 52.4% exceeded a CV of 100%, and 51% of jobs have 75% underutilized GPU, mostly allocating four GPUs but using only one.

As a complementary analysis, we examine the proportion of underutilized GPUs in jobs with high spatial imbalance (>0.5). A GPU is considered underutilized if its total utilization is at least 50% lower than the maximum total utilization among GPUs assigned to the same job. The right plot in Figure 4.8 shows that 51% of jobs have 75% underutilized GPU. We also found that among these jobs, 96% allocate four GPUs (one node) but actively use only one. This occurs because the scheduler allocates entire, non-shareable GPU nodes on Perlmutter, even when only one GPU is needed.

Figure 4.9 relates spatial imbalance of GPU_UTIL to FP pipeline usage (left plot) and job type (right plot). We normalize the density of each violin so that the total area remains consistent across all violins. White lines mark quartiles. The left plot reports spatial imbalance of GPU_UTIL for exclusive FP pipe group derived from Figure 4.2. The right plot reports spatial imbalance of GPU_UTIL by job type.

Figure 4.9 demonstrates that Tensor-only jobs exhibit the greatest spatial imbalance (me-

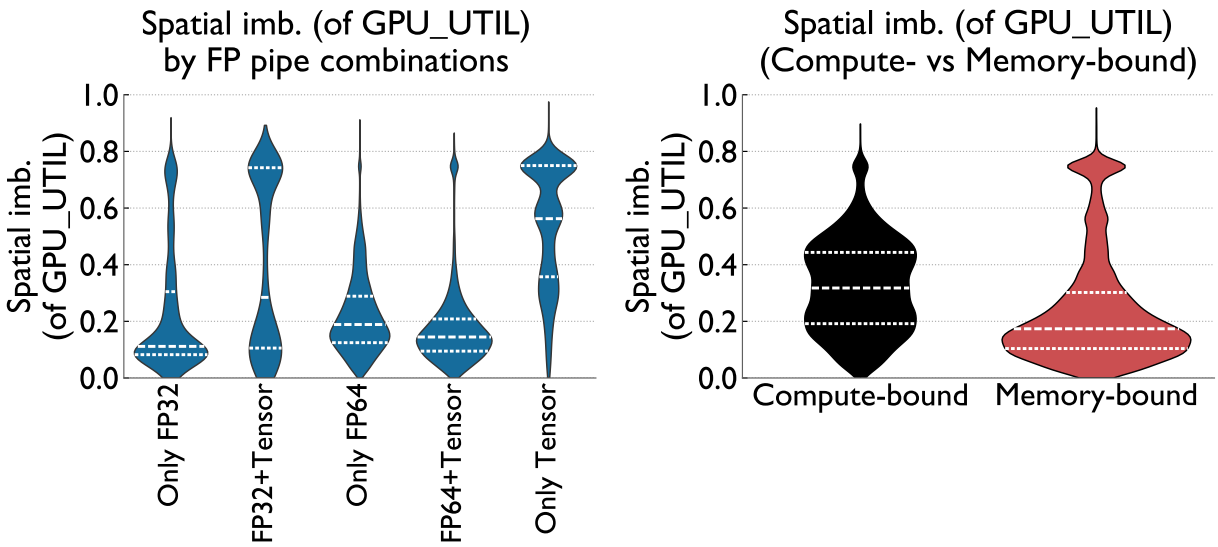


Figure 4.9: The plots demonstrate spatial imbalance by FP pipes used and by job type. Left: spatial imbalance of GPU_UTIL grouped by FP pipes used. Right: spatial imbalance of GPU_UTIL by job type (memory-bound vs compute-bound). Each violin illustrates the job-level distribution and is area-normalized (width encodes density); internal lines mark quartiles.

dian 0.56, IQR 0.36–0.75; p95 0.75). FP32+Tensor is also elevated (median 0.28, IQR 0.10–0.74). By contrast, FP32-only (median 0.11) and FP64-dominated combinations are lower: FP64+Tensor (median 0.14, IQR 0.09–0.20) and FP64-only (median 0.18, IQR 0.12–0.28).

Using FP64-derived arithmetic intensity, compute-bound jobs exhibit higher typical imbalance than memory-bound jobs (median values of 0.31 and 0.17, respectively). Overall, tensor-heavy groups and compute-bound workloads typically have higher spatial imbalance, whereas memory-bound jobs tend to be lower on average.

Figure 4.10 presents the spatial imbalance of GPU_UTIL for ML vs non-ML jobs. The plot compares the spatial imbalance between ML and non-ML jobs. ML jobs exhibit a flatter spatial imbalance distribution, while non-ML jobs mostly cluster around 0.5 imbalance, but extend up to 1.0. In contrast, ML jobs reach up to 0.9 imbalance.

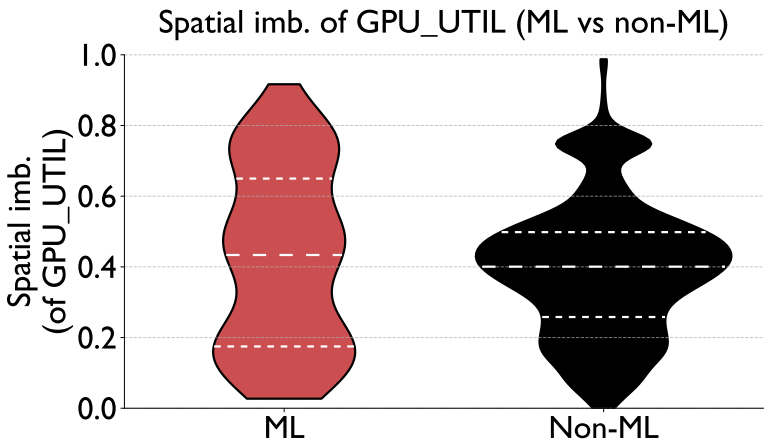


Figure 4.10: The plot illustrates spatial imbalance of GPU_UTIL across job type (ML vs non-ML). ML jobs have a flatter distribution, while non-ML jobs cluster around 0.5 imbalance.

> **Answer to RQ5** Jobs that sustain higher overall GPU utilization distribute work more uniformly across their GPUs, whereas low-utilization jobs typically exhibit higher spatial imbalance. Jobs with tensor-heavy combinations (e.g., Tensor-only, FP32+Tensor) tend to be more imbalanced than those that are FP64-dominated or FP32-only. Compute-bound jobs exhibit higher spatial imbalance than memory-bound jobs.

4.7 Analyzing the Temporal Behavior of Jobs

In this section, we analyze the distribution of temporal imbalance of GPU_UTIL and quantify its dependence on FP pipes used and job types. Temporal imbalance captures how consistently a job sustains GPU activity over its runtime. This section addresses the following question:

> **RQ6** *Are GPUs used consistently over time within a single job, and how does temporal imbalance vary with overall GPU utilization, FP pipes used, and job type (compute- versus memory-bound and ML vs non-ML jobs)?*

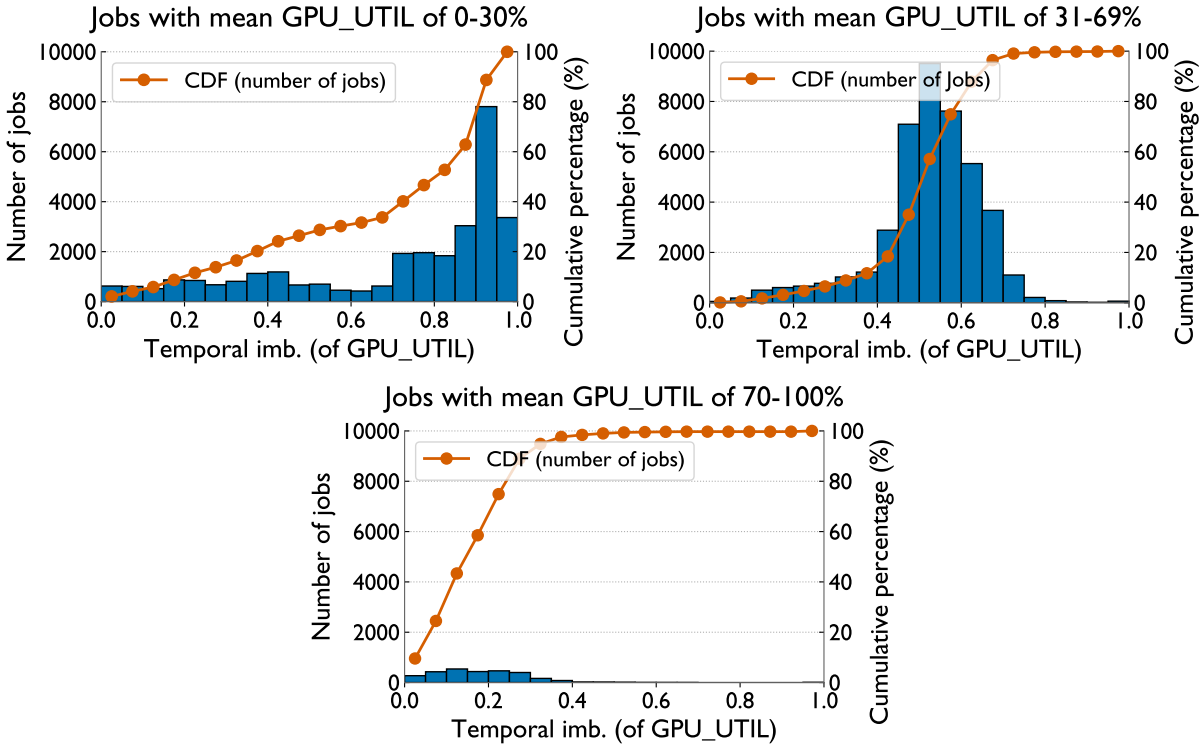


Figure 4.11: Distributions of temporal imbalance of GPU_UTIL for jobs grouped by mean GPU_UTIL (0–30%, 31–69%, 70–100%). Low-utilization jobs concentrate at high imbalance, while high-utilization jobs are tightly clustered at low imbalance.

4.7.1 Temporal Imbalance

Figure 4.11 demonstrates the distribution of temporal imbalance of GPU_UTIL for jobs grouped by mean GPU_UTIL: low (<30%), medium (31%-69%), and high (>70%). Low-utilization jobs (left) are broadly spread, with most clustered at high imbalance: only 13.7% fall below a temporal imbalance value of 0.30, while 66.3% exceed 0.70. Medium-utilization jobs (middle) are more balanced. Their distribution shifts lower and tightens around 0.5, with 6.5% below 0.30 and 3.6% above 0.70. Finally, high-utilization jobs (right) concentrate at low imbalance and 88.9% of high-utilization jobs fall below temporal imbalance value of 0.30, and only 0.3% exceed 0.70.

Figure 4.12 demonstrates temporal imbalance distributions by FP pipe used (left plot) and

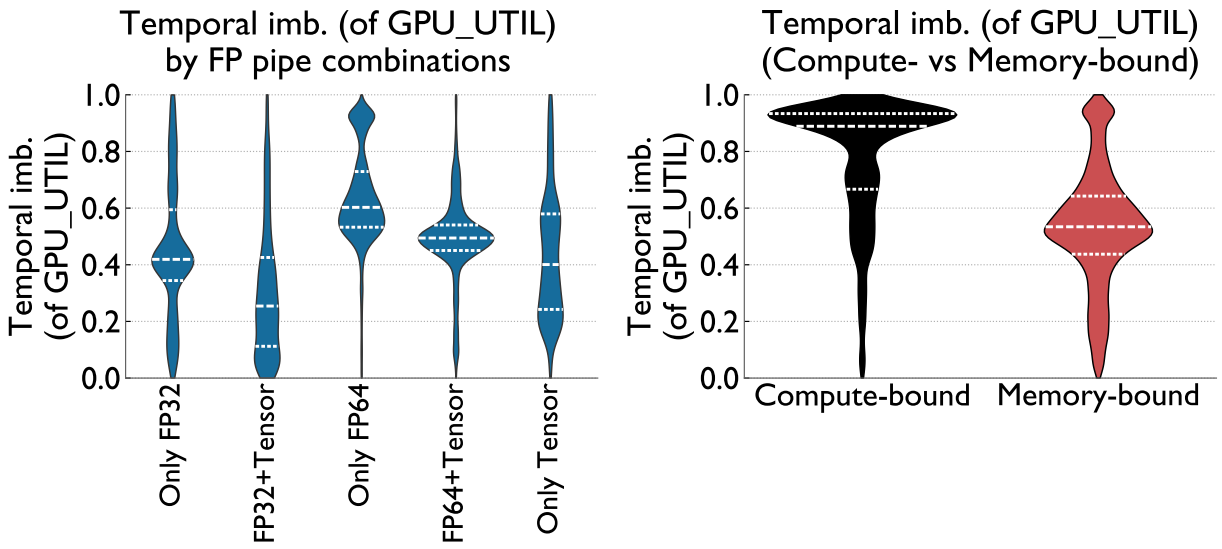


Figure 4.12: The plots demonstrate temporal imbalance by FP pipes used and by job type. Left: temporal imbalance of GPU_UTIL grouped by FP pipes used. Right: temporal imbalance of GPU_UTIL by job type. Each violin illustrates the job-level distribution and is area-normalized (width encodes density); internal lines mark quartiles.

job type (right plot). Each violin is an equal-area density with quartiles, as in Figure 4.9. Temporal imbalance varies with both FP pipes (left plot) and job type (right plot). FP64-only exhibits the highest and relatively tight temporal imbalance (median 0.60, IQR 0.53–0.73). FP64+Tensor (median 0.50, IQR 0.45–0.54), FP32-only (median 0.42, IQR 0.34–0.60) and Tensor-only (median 0.40, IQR 0.24–0.58) are moderate, while FP32+Tensor is lowest (median 0.25, IQR 0.11–0.43). By job type, compute-bound jobs exhibit markedly higher temporal imbalance than memory-bound jobs (medians 0.89 vs. 0.53; IQRs 0.67–0.93 vs. 0.44–0.64).

Figure 4.13 shows the temporal imbalance distribution of GPU_UTIL for different job types. Non-ML jobs have a higher density at high imbalance values (~0.9). In contrast, ML jobs have a higher density around ~0.2. This suggests that non-ML jobs exhibit more variability in GPU_UTIL over time.

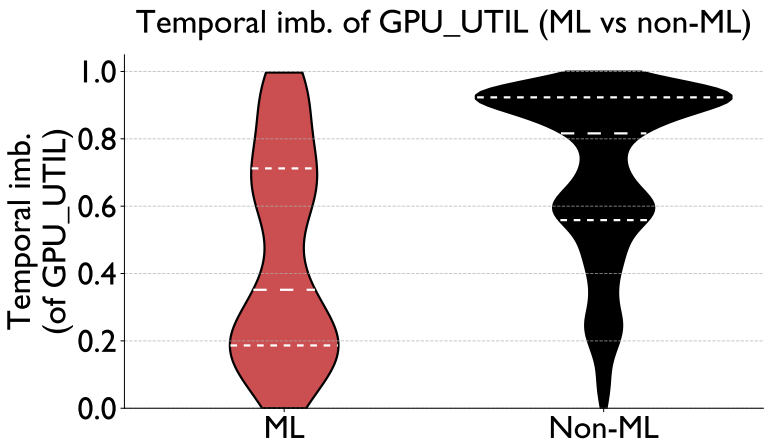


Figure 4.13: The plots show temporal imbalance of GPU_UTIL across job type (ML vs non-ML). ML jobs exhibit lower temporal imbalance.

> **Answer to RQ6** Greater mean GPU_UTIL correlates with lower temporal imbalance. Low-utilization jobs often exhibit high temporal imbalance, whereas high-utilization jobs maintain consistently high GPU_UTIL (low temporal imbalance). FP64-only jobs have higher temporal imbalance. Compute-bound jobs exhibit more temporal imbalance than memory-bound jobs.

4.7.2 Burstiness over Time

In this section, we analyze burstiness for jobs with low(<30%), medium (30%-70%), and high (>70%) mean of GPU_UTIL. To characterize burstiness, we define bursts as an increase of a counter value exceeding 15% from the previous value, as detailed in Section 4.3.4.

> **RQ7** *How bursty is GPU utilization over time, and how does burstiness relate to utilization level and temporal imbalance?*

Figure 4.14 presents burstiness distributions of GPU_UTIL across the three mean of GPU_UTIL

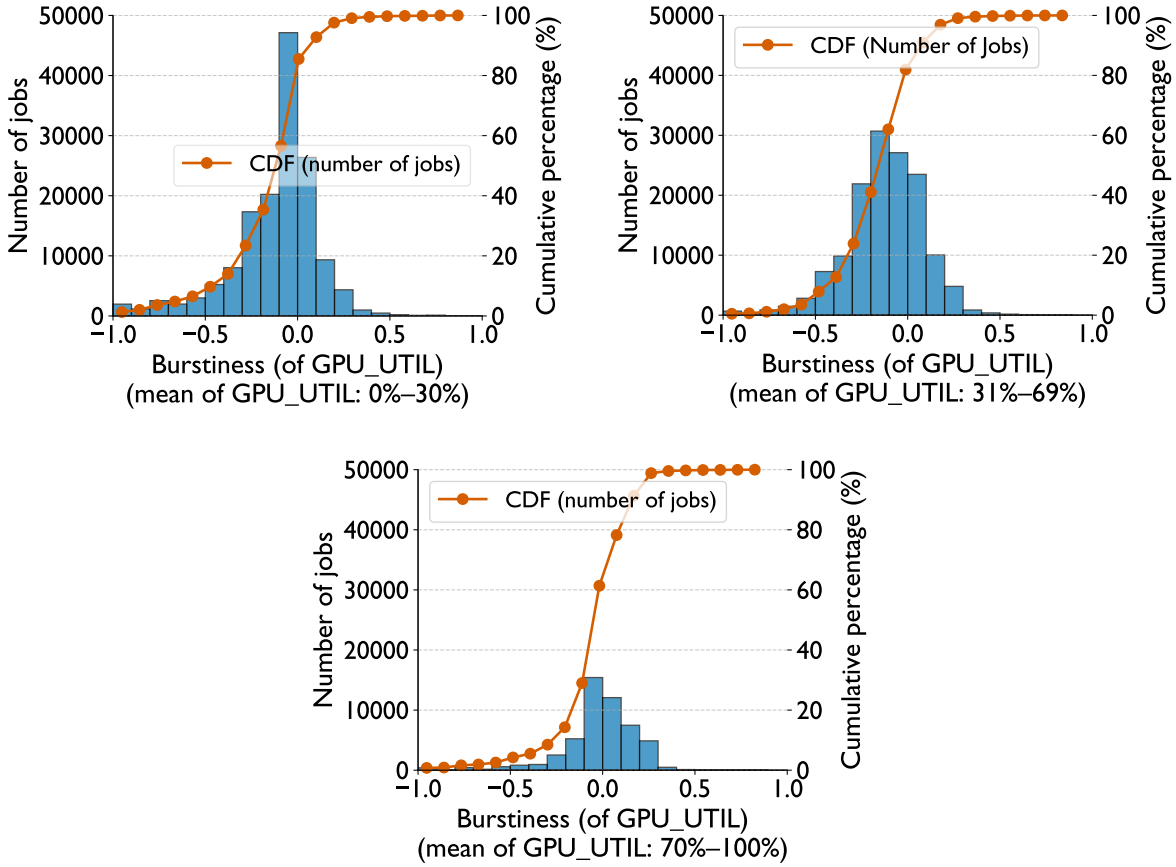


Figure 4.14: The plots show distribution of burstiness of GPU_UTIL for jobs grouped by mean GPU_UTIL ranges (0–30%, 31–69%, 70–100%, left to right). High-utilization jobs show more irregular bursts compared to low- and medium-utilization jobs.

ranges. The left plot shows that 85.5% of jobs with low mean of GPU_UTIL have a burstiness value below 0. This indicates that jobs in this category exhibit mostly regular fluctuations. Jobs with values near -1 show consistent, steady inter-event times with regular GPU usage or idling patterns. The lack of positive burstiness suggests these jobs do not exhibit irregular bursts of GPU activity, likely due to their light workload demands.

Medium-utilization jobs (middle plot) have a more spread-out distribution but still remain centered around 0. Among medium-utilization jobs, 82% still have a burstiness value below 0. It indicates a mix of steady and moderately irregular bursts. High-utilization jobs (right plot) show

a slight skew towards positive burstiness. Among high-utilization jobs, 39% have a burstiness value above 0. This suggests a greater irregularity among high-utilization jobs.

Our spatial imbalance metric is averaged across time windows. Therefore, it allows us to apply burstiness to measure irregular fluctuations in spatial imbalance over a job's lifetime. A burst in spatial imbalance is defined as a change in imbalance exceeding 0.15 from the previous time window. We analyze burstiness in spatial imbalance of GPU_UTIL for jobs with high spatial imbalance (>0.5).

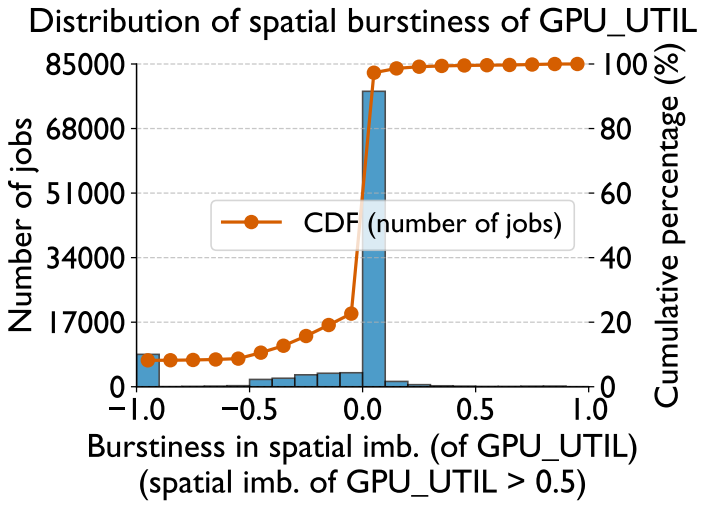


Figure 4.15: The plot illustrates burstiness in spatial imbalance of GPU_UTIL for jobs with spatial imbalance value of greater than 0.5. Bursts in spatial imbalance is mostly regular.

Figure 4.15 shows that burstiness in spatial imbalance peaks around 0, and 8.2% of jobs are close to -1. This indicates regular fluctuations. Although some GPUs are heavily loaded compared to others (Figure 4.8), significant changes in spatial imbalance occur at regular intervals rather than irregularly.

Table 4.1: Categorization of jobs by temporal imbalance and burstiness of GPU_UTIL (low and high). The table includes the total number of jobs, total GPU hours, and the proportion of jobs with low (<30%), medium (30%-70%), and high (>70%) mean of GPU_UTIL in each category.

Burstiness	**Temporal Imb. of GPU_UTIL**	**# of Jobs**	**Total GPU Hours**	**Job Ratio (Low/Med./High)**
Low (≤ 0)	Low (≤ 0.5)	104,730	1,462,583	32/42/26%
	High (> 0.5)	145,971	2,657,522	54/45/1%
High (> 0)	Low (≤ 0.5)	55,639	2,865,362	16/41/43%
	High (> 0.5)	38,603	3,534,739	75/24/1%

4.7.3 Categorizing Jobs by Burstiness and Temporal Imbalance

We further investigate the interaction between burstiness and temporal imbalance of GPU_UTIL by categorizing jobs into a matrix based on burstiness (low or high) and temporal imbalance (low or high) metrics, as shown in Table 4.1. We also examine the proportion of jobs with low (<30%), (30%-70%), and high (>70%) mean of GPU_UTIL in each category. This analysis provides deeper insights into the temporal behavior of GPU workloads.

Low Burstiness(≤ 0)**, Low Imbalance**(≤ 0.5) Jobs in this category exhibit steady usage patterns and regularly stay near to their maximum counter value. These jobs are generally desirable for HPC systems when mean of GPU_UTIL is relatively high, as they consistently push the GPU. The table shows that there are a total of 104,730 jobs in this category and 32% of jobs have low mean of GPU_UTIL, 42% have medium, and 26% have high mean of GPU_UTIL. The jobs with low mean represent consistent underutilization. This may be due to inefficient GPU usage or the minimal demand of the jobs.

Low Burstiness(≤ 0)**, High Imbalance**(> 0.5) Jobs in this category are relatively steady but it's generally far below the maximum they can reach. Most jobs, with a total of 145,971, fall into this category. When the mean of GPU_UTIL is low (54% of the jobs in this category), these

jobs often underutilize GPU resources with some minor irregular bursts. This underutilization also might occur due to low resource demands of the jobs.

High Burstiness(> 0)**, Low Imbalance**(≤ 0.5) Jobs in this category exhibit irregular bursts (high burstiness) but still remain close to their maximum (low imbalance). For example, the jobs with high mean of GPU_UTIL fluctuate between high and very high counter values, such as oscillating between 80% and 100% with irregular changes. This category is not a major concern since the total number of jobs in this category is relatively low (64,433) and the most jobs have medium (41%) and high (43%) mean of GPU_UTIL.

High Burstiness(> 0)**, High Imbalance**(> 0.5) This category represents the most problematic jobs and possibly indicates more anomalies. The bursts are irregular, and the jobs rarely stay near their peak. The jobs that have low mean of GPU_UTIL allocate GPU resources for a few intense bursts while leaving it underutilized most of the remaining runtime. This category has fewer jobs (38,603) but most of them have low mean of GPU_UTIL (75%).

> **Answer to RQ7** Low- and medium-utilization jobs mostly exhibit regular fluctuations, while high-utilization jobs show more irregular bursts in GPU utilization. The combination of high burstiness and high temporal imbalance is the most undesirable pattern and is dominated by low-utilization jobs. In contrast, burstiness in spatial imbalance is usually near zero, which indicates mostly regular changes in imbalance over time.

4.8 Analyzing Relationships between Hardware Counters

We examine how mean GPU_UTIL changes with FP64_ACTV and DRAM_ACTV for compute- versus memory-bound jobs (via job-level heatmaps), and we compute a Spearman cor-

relation matrix over job-level means to summarize how all counters relate to one another. This section addresses the following research questions:

> **RQ8** *How do compute and memory activity counters relate to overall GPU utilization across compute-bound and memory-bound jobs, and which counter pairs exhibit the strongest correlations?*

4.8.1 GPU Utilization of Compute- versus Memory-Bound Jobs

We measure how job-level mean GPU_UTIL changes with compute and memory activity. For compute, we use FP64_ACTV because it is the most commonly used pipeline. For memory, we use DRAM_ACTV, which tracks the achieved bandwidth on the A100 at a fixed HBM clock.

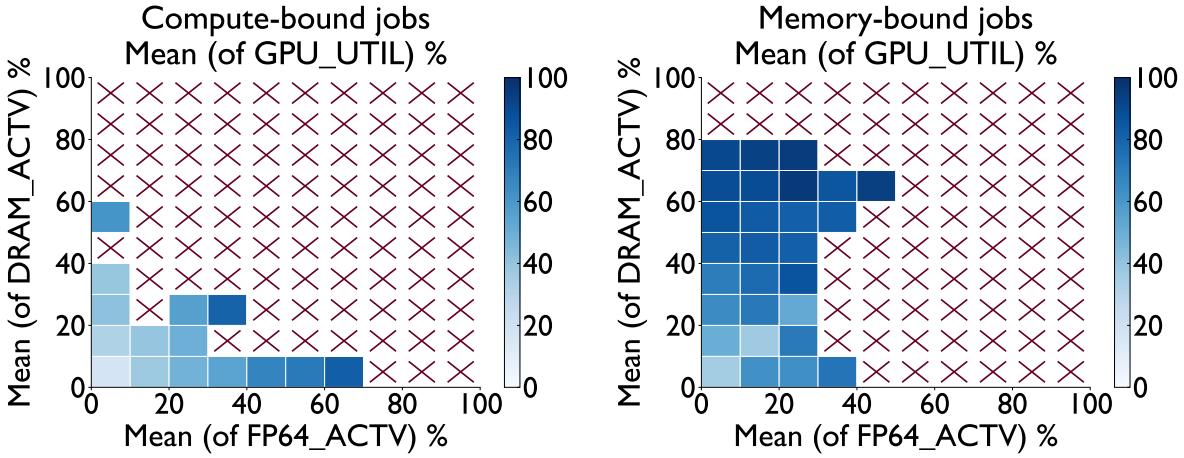


Figure 4.16: The plots demonstrate the job-level heatmaps of mean GPU_UTIL as a function of FP64_ACTV and DRAM_ACTV. We bin jobs by their mean FP64_ACTV (x-axis) and mean DRAM_ACTV (y-axis); color encodes the mean GPU_UTIL of jobs in each bin. Left: compute-bound jobs. Right: memory-bound jobs. Crosses indicate empty bins.

Figure 4.16 plots the resulting heatmaps. For compute-bound jobs (left), mean GPU_UTIL increases primarily with FP64_ACTV across a wide range of DRAM_ACTV, with corresponding correlations of 0.508 for FP64_ACTV and 0.102 for DRAM_ACTV. In memory-bound jobs

(right), mean GPU_UTIL increases with both axes. Bins with DRAM_ACTV >40% sustain higher GPU utilization even when mean FP64_ACTV is low. This pattern aligns with job-level correlations of mean GPU_UTIL with FP64_ACTV (0.513) and DRAM_ACTV (0.570).

These patterns are consistent with roofline reasoning. The performance of the compute-bound jobs (to the right of the ridge) is limited by peak flop/s. Therefore, incremental changes in delivered HBM bandwidth have little leverage. In the memory-bound regime (left of the ridge), sustained throughput improves either by delivering more bandwidth (higher DRAM_ACTV) or by increasing arithmetic intensity (more useful flops per byte), which typically co-moves with FP pipe activity. We emphasize that these are associations, not causal proofs. The exact magnitudes depend on factors such as kernel mix and cache behavior.

4.8.2 Relationship between SM_ACTV, MEM_UTIL and GPU_UTIL

We analyze the relationship between SM_ACTV, MEM_UTIL and GPU_UTIL to understand how memory transfer (MEM_UTIL) and computation (SM_ACTV) components contribute to GPU_UTIL. Figure 4.17 visualizes GPU_UTIL as a function of mean (left), spatial imbalance (middle), and temporal imbalance (right) metric values of SM_ACTV and MEM_UTIL. Colors represent the corresponding metric for GPU_UTIL. Cells with no jobs shown in orange.

In these heatmaps, each cell in the plots represents a binned value range for SM_ACTV and MEM_UTIL. For instance, in the left plot, we divide the mean counter values into 10 bins. Jobs with mean counter values falling within a specific bin are grouped into the corresponding cell. The color of each cell represents the mean GPU_UTIL of all jobs within that bin.

The left plot in Figure 4.17 shows that high mean of GPU_UTIL (dark blue) occurs when

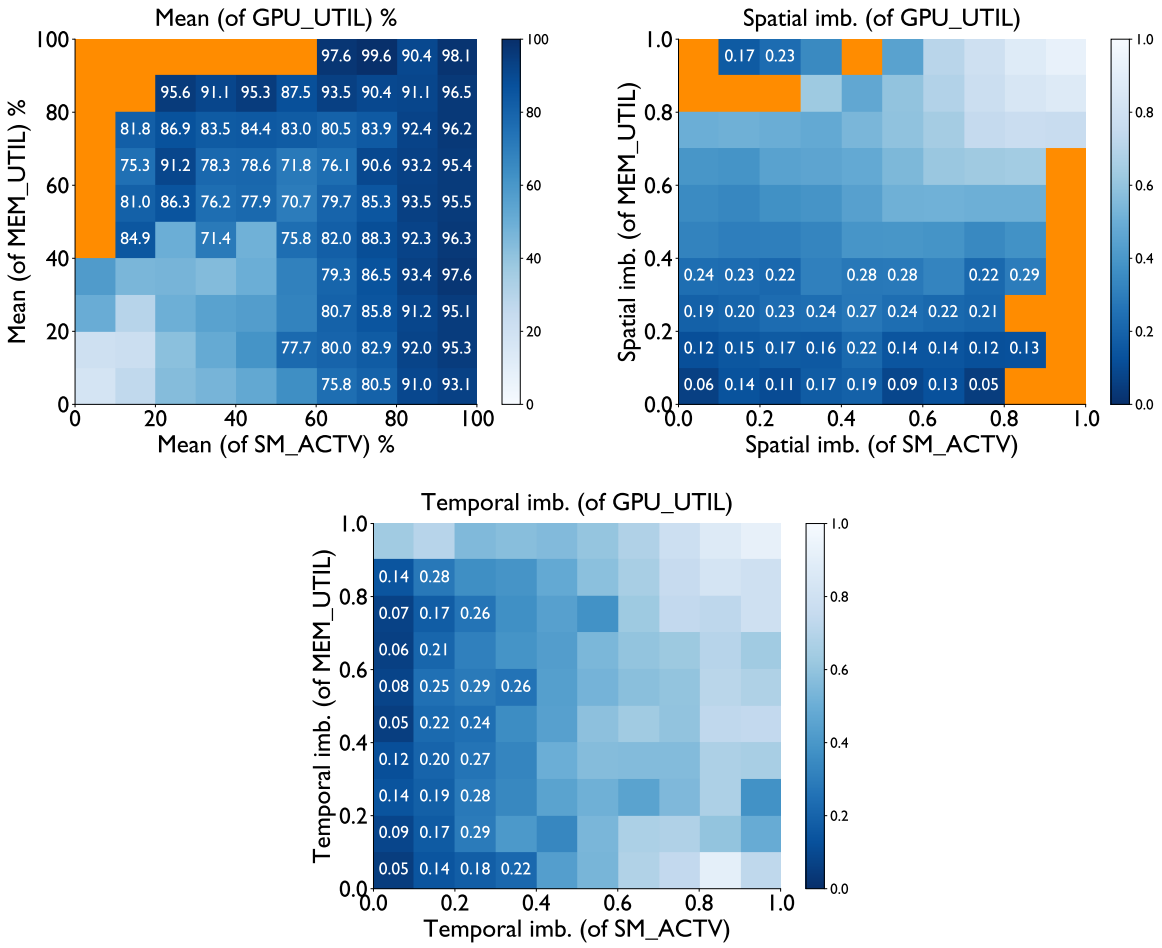


Figure 4.17: The plots show relationships between SM_ACTV, MEM_UTIL, and GPU_UTIL. High mean of GPU_UTIL occurs when both SM_ACTV and MEM_UTIL are high (left). Spatial imb. of MEM_UTIL has a stronger impact on spatial imbalance of GPU_UTIL (middle). Temporal imb. of SM_ACTV has a stronger impact on temporal imbalance of GPU_UTIL (right). Cells with no jobs are shown in orange.

both mean SM_ACTV and MEM_UTIL are high, which indicates efficient use of both compute and memory transfer resources. The cells with mean GPU_UTIL value of greater than 70% are annotated.

The middle plot in Figure 4.17 presents spatial imbalance using the same axes. Annotated cells represent spatial imbalance of GPU_UTIL values below 0.3. Unlike the middle plot, the brighter cells are mostly clustered around the low spatial imbalance of MEM_UTIL values (<0.2)

with a few exceptions on top left. As spatial imbalance of MEM_UTIL increases (moving up the y-axis), spatial imbalance in GPU_UTIL decreases, even when spatial imbalance in SM_ACTV remains low. However, when the spatial imbalance of MEM_UTIL is low ($<$0.2), the spatial imbalance of GPU_UTIL remains relatively more stable as the spatial imbalance of SM_ACTV increases. This suggests that spatial imbalance of MEM_UTIL has a stronger effect on spatial imbalance of GPU_UTIL.

The right plot in Figure 4.17 visualizes temporal imbalances of SM_ACTV and MEM_UTIL, colored by temporal imbalance of GPU_UTIL. Cells with mean temporal imbalance of GPU_UTIL values below 0.3 are annotated. As SM_ACTV increases (moving right along the x-axis), the temporal imbalance of GPU_UTIL also increases (darker colors), regardless of MEM_UTIL. However, increasing temporal imbalance of MEM_UTIL (moving up) has a weaker effect on GPU_UTIL. This suggests that temporal imbalance of SM_ACTV has a stronger influence on GPU_UTIL.

4.8.3 Correlation among Hardware Counters

To understand how hardware counter values co-vary at the job level, we compute a Spearman correlation matrix over job-level means of the counters (Eq. 4.10). We order the heatmap by the absolute correlation with GPU_UTIL.

Figure 4.18 demonstrates that mean GPU_UTIL correlates most strongly with GPU_POWER (0.78), and GPU_TEMP (0.77), and SM_ACTV (0.76). We calculate GPU_POWER by dividing the total energy consumption per job (TOTAL_ENG) by GPU count and job duration. These counters co-vary strongly across jobs: higher active compute is accompanied by higher power

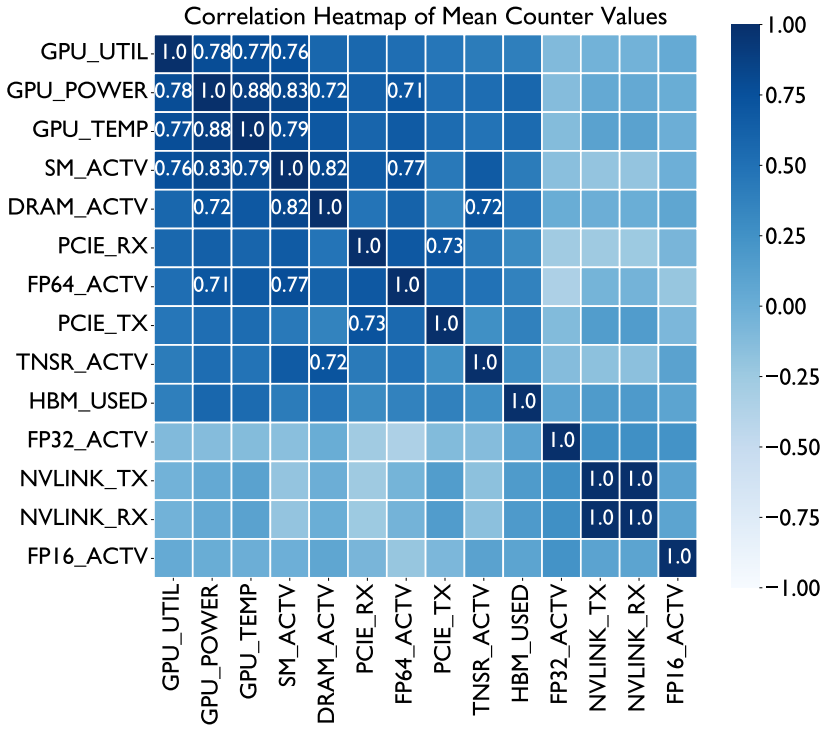


Figure 4.18: The plot presents a Spearman correlation heatmap of job-level mean counters. We order cells by absolute correlation with GPU_UTIL. Only strong correlations ($|\rho| \geq 0.7$) are annotated.

draw and device temperature. DRAM_ACTV correlates positively as well (0.58), consistent with memory activity supporting sustained utilization.

SM_ACTV correlates strongly with DRAM_ACTV (stalled warps are counted as active) and with FP64_ACTV, while having a weak negative association with FP32_ACTV (−0.11). This negative sign reflects workload composition: FP64-dominant and FP32-dominant jobs are largely disjoint groups, so raising one pipe's activity tends to lower the other at the job-level mean, yielding a small inverse relationship.

NVLINK_TX and NVLINK_RX are nearly symmetric (0.99), consistent with bidirectional data exchange in intra-node collectives. In contrast, PCIE_RX and PCIE_TX are less symmetric (0.73) and only moderately related to GPU_UTIL (0.57 and 0.46), likely because PCIe traffic is

often directional (e.g., inter-node communication, host-to-device staging, checkpoints).

> **Answer to RQ8** In compute-bound jobs, GPU utilization is most strongly associated with compute activity, while DRAM activity contributes weakly. In memory-bound jobs, both compute and DRAM activity are positively associated with utilization. Across all jobs, overall GPU utilization aligns most closely with SM activity, power usage, and GPU temperature.

4.9 Discussion

This chapter synthesizes two telemetry studies of GPU workloads on Perlmutter: a longitudinal 2023 study centered on utilization and imbalance, and a 2025 study that extends the analysis with energy, temperature, and interconnect counters. These studies offer a job-level, GPU-centric characterization of workload behavior using previously underutilized DCGM counters. Spatial and temporal imbalance expose uneven and unsteady use of allocated GPUs; the roofline-based job characterization clarifies differences in energy consumption; FP pipeline analysis differentiates job behavior; and counter correlations indicate which signals move together. Collectively, these results support practical choices about optimization targets and scheduling policies on Perlmutter, and the methodology is applicable to other GPU-accelerated systems.

Leveraging tensor cores. Tensor pipe activity is typically associated with higher overall GPU utilization. While this relationship is associative rather than causal, it motivates adopting or upgrading tensor-enabled libraries/kernels where appropriate. For FP64 workloads (the most common on Perlmutter), targeted mixed-precision can improve device activity.

Job type determines the effective optimization. Our roofline-based analysis indicates that compute-bound jobs align primarily with FP pipe activity, whereas memory-bound jobs benefit from increases in both DRAM activity and FP pipe activity that raises arithmetic intensity. Practically, for compute-bound jobs, optimization should prioritize kernel efficiency, instruction-level parallelism, and increased floating-point issue. For memory-bound jobs, optimization should prioritize memory throughput and reuse (tiling, caching, reduced precision where valid) to shift arithmetic intensity rightward.

Energy as a scheduling signal. At comparable GPU-hour scales, memory-bound jobs tend to consume more total energy than compute-bound jobs. For operators, this suggests energy-aware placement and capping policies when memory-bound workloads dominate a node. For users, reducing memory traffic (and increasing reuse) can yield both performance and energy benefits.

Imbalance diagnostics for users and operators. Spatial and temporal imbalance concentrate in low-utilization jobs and in specific FP pipe mixes. Exposing spatial/temporal dashboards in user-facing job monitors can help identify load imbalance (rebalance partitions, adjust batch sizes, or request fewer GPUs), while operators can use this information to guide co-scheduling.

Properly sizing allocations for HBM requests. Peak HBM usage is typically below device capacity for many jobs that explicitly request 80-GB GPUs. A lightweight submission-time advisory could inspect a user's recent per-job peak HBM usage and recommend an appropriate memory tier (e.g., 40-GB vs 80-GB) before the scheduler queues the job. Such guidance may reduce wait times on scarce 80-GB nodes and improve overall throughput, without enforcing any policy.

Interpreting low utilization carefully. Low GPU utilization should not always be interpreted

as evidence of poor application implementation. Some low-utilization and highly imbalanced jobs may reflect allocation granularity or conservative job requests rather than purely computational inefficiency. This distinction matters when using telemetry to guide optimization or policy decisions.

Implications for scheduling and accounting. The prevalence of small jobs and the evidence of partial GPU use suggest that scheduler policy can affect observed efficiency as much as application behavior does. When allocation granularity forces users to reserve more GPUs than they actively use, utilization statistics may partly reflect system policy rather than intrinsic workload demand. This motivates allocation-aware interpretations of telemetry and, where feasible, more flexible resource-request guidance.

Limitations. This chapter combines a 2023 dataset spanning roughly four months with a 2025 dataset spanning one month. Both datasets rely on job-level aggregation. The sampling interval is set by system administrators and cannot be changed by users, so brief spikes may be missed. Our analysis is intentionally limited to what can be inferred from production monitoring without code-level profiling, so function call paths and thread-level behavior are not captured. While the methodology is broadly applicable, comparable job-level telemetry is hard to obtain across sites due to access permissions and monitoring differences. Higher-frequency traces, cross-site validation, and complementary code-level profiling are valuable directions for future work.

Chapter 5: Telemetry-driven Early Prediction of Performance Slowdowns on GPU-based Systems

This chapter presents a telemetry-driven method for predicting run-to-run performance variability in production GPU applications on Perlmutter. The goal is to determine whether an arbitrary run on the system will be unusually slow relative to similar runs by using only information that is routinely available in production systems.

5.1 Introduction

Run-to-run performance variability is an age-old problem on HPC clusters and supercomputers [65, 11, 37, 9, 31, 20, 10, 89]. Even when a job runs the same application in an identical configuration, the job can run fast or slow depending on the state of the overall system. Such variability often arises due to contention for shared resources such as the network and parallel filesystem [11]. In practice, users often compensate for this uncertainty by requesting extra time in their job submissions, which increases queue wait times for everyone. Performance variability also reduces overall system throughput and leads to energy waste.

Prior work suggests that historical telemetry data can help predict variability, but such methods often rely on application-specific knowledge [58]. In this work, our goal is to predict whether a GPU job will be unusually slow relative to similar jobs while using only minimal infor-

mation available at job start or shortly thereafter. This design goal reflects operational constraints in production systems, where continuous profiling, pre-run benchmarking and deep knowledge of individual jobs are impractical.

Predicting the performance of arbitrary GPU applications in a production job queue is difficult for several reasons. A practical method cannot rely on application-specific knowledge because privacy constraints often prevent access to detailed job information. In addition, performance variability is only meaningful relative to repeated executions of the same application under the same execution setting. We therefore need an automated way to identify runs that likely correspond to the same application under the same configuration. In production data, exact input files and internal algorithmic settings are usually unavailable, so this grouping must rely on metadata available at job start. The prediction method must also use only information available at or shortly after run start, without offline profiling or pre-run benchmarking. Finally, it must remain robust across different job sizes.

We develop a novel method to predict run-to-run variability for any production GPU application, without any knowledge of what the application is (other than the executable name), near the start of its execution. Our method uses only scheduler logs and network telemetry data routinely collected on production HPC systems. We use data collected on a flagship supercomputer over the period of two months, but this approach is not limited to a specific hardware or architecture. We use these historical data to group similar jobs and to train group-specific machine learning models that predict whether an arbitrary job will be slow relative to its group. The ML models learn relationships between system telemetry data and performance variability.

We conduct a variety of experiments to evaluate the efficacy of these models along various dimensions – group size, job duration, length of telemetry data used for prediction, and whether

all telemetry data is used or a subset is used affect prediction performance. The results indicate that our approach predicts whether a job will be slow relative to similar jobs with high accuracy. The strongest overall performance comes from telemetry collected on the job's allocated nodes after job start. The results also indicate that about 8 minutes of early-run telemetry is sufficient to predict slowdown for many jobs that continue running much longer. Under this setting, and across the two slowdown definitions introduced later, 9 of the 15 retained groups achieve slow-class F1 of at least 0.95 under one definition, and 10 of the 15 retained groups achieve F1 of at least 0.90 under the other. We also analyze feature importance to identify the most informative telemetry signals, and we validate the overall methodology using ground-truth repeated application runs.

Our contributions are as follows:

- We present a new approach to predict run-to-run performance variability for any production GPU application using system-wide monitoring data, without application profiling or pre-run benchmarking. To the best of our knowledge, this is the first variability prediction model that relies solely on telemetry data and requires no code instrumentation.

- We make variability prediction feasible for arbitrary production workloads by using job metadata to group similar runs that approximate repeated executions of the same application and launch configuration, and by defining a group-specific variability value. This enables prediction without application-specific knowledge.

- We quantify how telemetry scope and timing affect prediction performance by comparing allocated-node versus all-node telemetry and pre-start versus post-start windows.

5.2 Data Collection and Curation

We construct two datasets in this study. The first is a dataset that combines system-wide NIC telemetry data with Slurm job metadata. We use this dataset to train machine learning models that predict whether a job will run unusually slowly relative to similar jobs. The second dataset consists of controlled repeated runs of known applications that we execute on Perlmutter. We use this dataset to validate our grouping and labeling methodology. We collect both datasets on Perlmutter during two month-long periods: July 18–August 20, 2025 and November 15–December 17, 2025. Below, we describe how we collect these datasets and prepare them for analysis.

5.2.1 NIC Telemetry Data for Training

We build the training dataset by combining system-wide NIC telemetry data with Slurm job-step metadata. On Perlmutter, LDMS records Cassini NIC counters on GPU nodes at 1-second intervals, and we retrieve job-step metadata from Slurm through `sacct`. We use Slurm job steps launched with `srun` as the unit of analysis and refer to each such step as a *run*. Starting from these records, we apply a preprocessing pipeline to obtain a consistent set of comparable runs for variability analysis and prediction across diverse workloads.

Because our goal is to predict performance variability in GPU applications, we exclude CPU-only runs. We also retain only successfully completed runs, since our objective is to model slowdowns rather than failures, cancellations, or preemption. Finally, we exclude single-node runs because they do not involve inter-node communication and are therefore less relevant to network-related variability.

For each retained run, we use its start time and allocated node list from Slurm metadata to associate the corresponding NIC telemetry samples by node and timestamp. This produces a run-level view of network activity that serves as the basis for the methodology described in Section 5.3, where we group similar runs, define variability labels within each group, construct features, and train prediction models.

Table 2.2 lists the counters used in this study. We organize them into several conceptual groups. Traffic volume counters measure packets sent and received across traffic classes. Pause counters capture flow control activity. Timeout and replay counters reflect request timeouts and link-level replay behavior under communication stress. Codeword counters summarize link-level transmission quality. PCIe traffic and backpressure counters characterize packet movement and blocked conditions on the PCIe path.

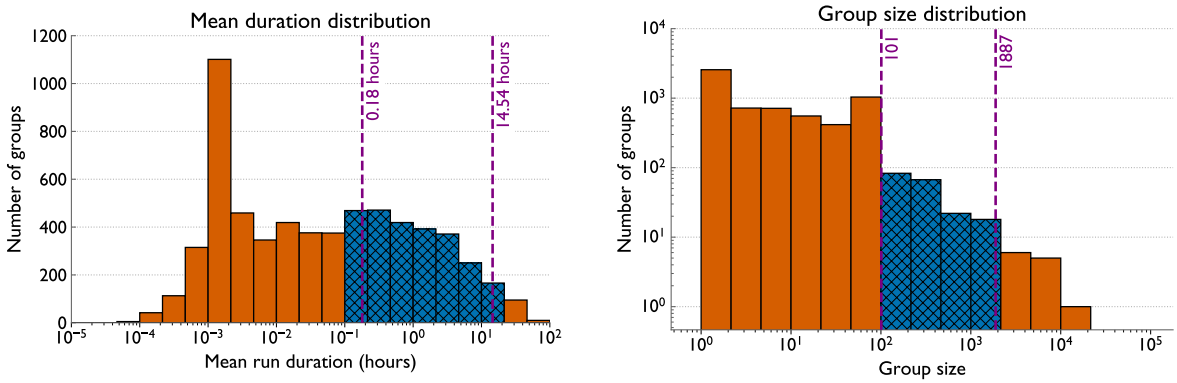


Figure 5.1: Overview of the production GPU workload after the initial filtering stage. The left plot shows the distribution of mean run duration in hours. The right plot shows the distribution of group size in number of runs. The blue bars indicate the range of groups retained for the final modeling dataset.

Figure 5.1 summarizes the filtered workload that forms the basis of the modeling dataset. Here, each group denotes a set of comparable runs defined by the grouping procedure described in Section 5.3.1. The left plot presents the distribution of group-level mean run duration in hours.

Mean duration spans several orders of magnitude, and many groups lie near the short-duration end. For modeling, we retain groups with mean duration of at least 10 minutes, shown by the blue bars. In the final analysis set, mean duration extends up to 14.54 hours.

The right plot reports the distribution of group size. Most groups contain few runs, while a smaller number contain hundreds or thousands. In the final modeling dataset, group size ranges from 101 to 1887 runs. Section 5.3.2 describes our methods to obtain this final analysis set. After applying all filters, 21 groups containing 13,158 runs remain and we use this fixed set of 21 groups in the rest of the chapter. This final analysis set covers a broad range of runtime scales and sample counts.

5.2.2 Control Jobs Dataset

We construct a dataset of controlled repeated runs to validate our grouping and labeling strategies under production conditions. We run several GPU-enabled applications. For each one, we fix the input, GPU count, and software configuration, and then repeat the same run many times. These repeated runs let us examine whether the group-specific baseline and variability values we compute capture slowdowns that are visible in practice. Our suite covers two kinds of workloads. On the HPC side, we run AMG2023 [47], MILC [8], LAMMPS [82], PSDNS [54], and BGW4 [24]. We choose these applications because each spends most of its runtime in a different communication primitive. For instance, AMG2023 is dominated by `all-reduce`, while PSDNS relies heavily on `all-to-all`. Together, they reflect workloads that typically run on production supercomputers.

Because AI training now accounts for a growing share of supercomputer workloads, we

also include three deep learning applications: nanoGPT with AxoNN [75, 76], DeepCAM [39], and Megatron-LM [74] Mixture of Experts (MOE) training. These cover common deep learning training patterns and rely on large message `all-reduce` and `all-to-all` communication. These controlled applications allow us to observe variability under production conditions and to validate our grouping and labeling methodology.

We collect the controlled jobs data during the same two periods as the NIC dataset. During each period, we submit three short jobs (three to five minutes) per day for each application. For DeepCAM, nanoGPT, LAMMPS, and PSDNS, we also submit two long jobs (30 minutes) per day. We spread submissions throughout the day to sample a range of system conditions and contention levels. In total, our suite contains 945 runs. All applications run on 64 GPU nodes.

5.3 Methodology for Slowdown Prediction

In this chapter, we develop a method to predict whether an arbitrary run will be unusually slow relative to previously observed comparable runs. Figure 5.2 provides an overview of the approach. First, we need to curate data to train machine learning models for slowdown prediction. To determine whether a run will be slower than "usual", we first need to define what usual means. We do this by identifying runs that are "similar" or comparable in their execution characteristics. This first step is Grouping in Figure 5.2. Next, we determine when a run should be considered slow relative to other runs in the same group. Within each group of similar runs, we create slowdown labels by comparing each run's duration with the minimum duration in that group. These labels allow us to train a binary classification model. Then, we curate features from NIC data for the labeled runs to create a training dataset and train a binary classifier to predict whether

a run will be unusually slow relative to its group.

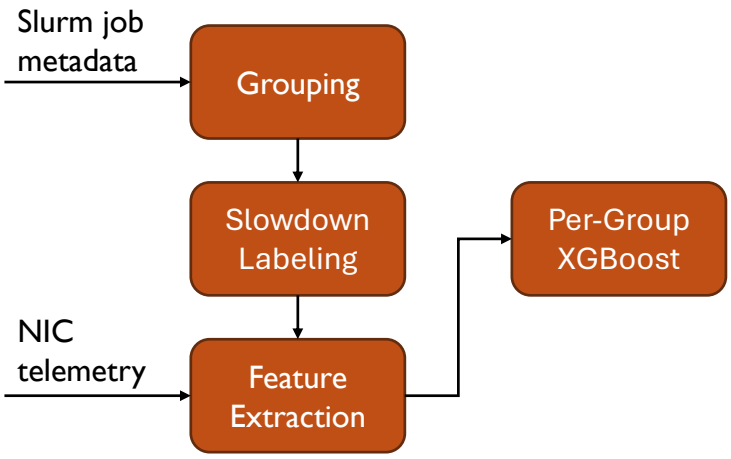


Figure 5.2: Overview of the proposed workflow. We group runs using Slurm metadata. Then, we create slowdown labels for training. Finally, we extract NIC features and train a separate XGBoost model for each group.

5.3.1 Identifying Comparable Runs from Slurm Metadata

Production runs differ widely in scale and behavior, so comparing all runs directly does not provide a meaningful notion of slowdown. We therefore group runs that share the same scheduler-visible execution settings. After the initial filtering described in Section 5.2.1, we assign each run to a deterministic group using three fields from Slurm metadata: the number of nodes `NNodes`, the job time limit `TimeLimit`, and the executable name obtained from the submission command line `SubmitLine`. We use these fields because they are consistently available in Slurm metadata and together capture the main aspects of how the run was launched.

We parse the SubmitLine field, which records the srun command used in the job script, to identify the application executable name. We then combine the executable name with the node-count and time-limit fields to form a deterministic group identifier. This procedure groups together runs that launch the same executable with the same requested resources and time limit.

The grouping rule is intentionally conservative because it uses only metadata that is available at production scale. Runs within the same group may still differ in ways that Slurm metadata does not expose, such as input data or internal algorithmic choices, but the group key still provides a practical and deterministic approximation of run similarity.

We validate the grouping rule with the control jobs dataset described in Section 5.2.2. In that dataset, we fix the input, node count, software configuration, and launch settings for each experiment, so that repeated executions represent the same execution setting. Under these conditions, the method groups repeated runs correctly. We use this grouping key as an approximation of run similarity.

5.3.2 Defining Slowdown Labels and Training Data

After we group similar runs, we assign binary slowdown labels within each group using the minimum Slurm elapsed time as the reference duration. We label a run as slow when its Slurm elapsed time exceeds a chosen multiple of this reference duration. Otherwise, we label it as normal. This rule defines slowdown relative to the fastest observed execution in the same group. In the results section, we report two representative thresholds, 1.05 and 1.3, which capture milder and more pronounced slowdowns.

Next, we define the training dataset. We remove runs shorter than 10 minutes because startup effects and small absolute timing fluctuations can dominate their runtime. We also remove runs whose runtime exceeds 10 times the group minimum because these cases usually reflect extreme outliers or imperfect grouping. For each threshold, we exclude groups with no slow runs, fewer than 100 remaining runs, or too few samples from both classes for stratified cross-

validation. Because these filters depend on the threshold, the usable set of groups can vary across threshold settings. These filters determine which runs enter the prediction task, but they do not change how we later construct NIC data features.

We validate this labeling strategy using the control jobs dataset described earlier. For each control jobs group, we measure the main `for` loop and assign a ground label when the loop runtime exceeds 1.15 times the group minimum. We use 1.15 in this setting because these workloads exhibit limited performance variability. We then identify the same runs in the Slurm data, group them with the same procedure, and assign Slurm labels from Slurm elapsed time using the same threshold. We compare the two labeled sets run by run to assess whether Slurm elapsed time preserves the slowdown behavior captured by in-application timing. Section 5.4.1 reports the agreement between these labels and discusses the remaining differences.

5.3.3 Constructing Features from NIC Data

For each run in the final training dataset, we construct features from NIC data under an experiment configuration defined by node scope and time window. The node scope is either *all nodes*, which includes data from all GPU nodes in the selected interval, including idle nodes, or *job nodes*, which includes data only from nodes allocated to the run by Slurm. The time window is either a *pre-start* interval immediately before run start or a *post-start* interval immediately after run start. We evaluate window lengths of 1, 2, 3, 5, 8, and 10 minutes.

For each NIC counter, we compute the counter increase for each NIC over the selected time window. Because these counters are cumulative, we measure the change in value within the interval. If a counter resets and its value drops, we add the restarted value after the reset so that

the total increase within the interval is preserved.

We aggregate these values in two stages. We first sum the NIC-level increases across NICs on each node to obtain a node-level value for each counter. We then aggregate these node-level values across the selected node scope. For each counter, we retain two summaries across nodes, the mean and the maximum. The mean captures the overall activity across the selected node scope, whereas the maximum captures hot spots that may reflect contention. We concatenate these summaries over all counters to form a fixed-length feature vector for each run and experiment configuration.

For the traffic-class specific counter families in Table 2.2, we also aggregate adjacent traffic classes in pairs, namely classes 0 and 1, 2 and 3, 4 and 5, and 6 and 7, to reduce dimensionality while preserving coarse communication structure. In our environment, classes 0 and 1 correspond to MPI message traffic, classes 2 and 3 are associated with I/O traffic, classes 4 and 5 are associated with TCP/IP traffic, and classes 6 and 7 appear to capture bulk data transfer.

We compute these features from the full NIC data rather than only from the filtered modeling subset. This choice preserves the actual system state observed by the run at that time. The filtering steps described earlier determine only which runs enter the prediction task.

5.3.4 Training and Evaluating Per-Group Classification Models

We train and evaluate one classifier for each group and each combination of node scope and time window. For each group, we use the NIC feature vectors described above to train an XGBoost binary classifier that predicts the slowdown label. We keep the model configuration fixed across all groups. This design enables direct comparison of node scopes and time windows

under a common modeling setup.

We evaluate each model with nested repeated stratified five-fold cross-validation. We first divide the data into five stratified folds and repeat this procedure five times, which gives 25 test evaluations for each group and each combination of node scope and time window. In each evaluation, we keep one fold aside for final testing and use the other four folds for model development. When the slowdown class is underrepresented in the training portion, we set the class-imbalance weight to the ratio of negative to positive examples in that portion.

The classifier outputs a probability for the slowdown class, so we must choose a probability threshold to convert these scores into binary labels. We do not choose this threshold using the held-out test fold. Instead, after setting one fold aside for final testing, we use the remaining four folds to determine the threshold by cross-validation. We select the threshold that gives the best F1 score for the slowdown class on this training portion. After selecting the threshold, we train the classifier on the full training portion and evaluate it on the held-out test fold. We then convert the predicted probabilities on the test fold into binary labels using the selected threshold.

To summarize performance, we first average the metrics across the five folds within each repeat and then report the mean and standard deviation across the five repeats. This protocol keeps model fitting, threshold selection, and final evaluation separate, and it enables fair comparison across node scopes and time windows under a common grouping rule and a common labeling rule.

5.3.5 Metrics for Evaluating Classification Models

Our main goal is to identify slow runs reliably, so we focus on metrics that emphasize the slowdown class. For a given decision threshold, precision and recall for the slowdown class are defined as

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}, \tag{5.1}$$

where TP, FP, and FN denote the numbers of true positives, false positives, and false negatives, respectively. We report the F1 score of the slowdown class after threshold selection. F1 is the harmonic mean of precision and recall,

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{5.2}$$

and summarizes their tradeoff in a single value. This metric is useful because it evaluates the final binary slowdown predictions produced by the selected threshold.

We also report AUPRC. AUPRC is the area under the precision-recall curve obtained by sweeping over all probability thresholds. It measures how well the model assigns higher slowdown scores to truly slow runs than to normal runs before a final threshold is applied. This metric is useful because it evaluates ranking quality independently of the particular probability cutoff used to produce binary predictions.

5.4 Results

We evaluate the proposed approach on production GPU runs from Perlmutter using system-wide NIC data and Slurm job-step metadata. We first validate the slowdown labels on controlled repeated runs by comparing labels from Slurm elapsed time with labels from in-application timing. We then analyze how node scope, timing relative to run start, and window length affect prediction performance on the production dataset. Finally, we report per-group performance under the selected setting and examine which NIC features are most informative.

5.4.1 Validating Variability Label Construction

Using the labeling procedure described in Section 5.3.2, we compare slowdown labels from Slurm elapsed time with labels from in-application timing for the controlled repeated-run workloads in Section 5.2.2. For this controlled study, we use a threshold of 1.15 because these workloads exhibit limited performance variability.

Table 5.1 reports agreement between slowdown labels from Slurm elapsed time and in-application timing for the controlled repeated-run workloads. Agreement ranges from 0.83 to 1.00 across workloads, and most workloads have agreement of at least 0.95. Agreement also remains high for both short and long variants of the same application.

The remaining mismatches are consistent with the difference between the two timing sources. In-application timing covers only the main `for` loop, while Slurm elapsed time covers the full run, including initialization, setup, and other work outside the main loop. To understand these cases, we manually inspected raw output logs for mismatched runs. In AI training workloads such as `deepCAM_short` and `nanoGPT_short`, some runs have larger variability under Slurm

Table 5.1: Agreement between slowdown labels from Slurm elapsed time and in-application timing for the controlled repeated-run workloads. # Runs is the number of runs in each workload. p_G^{slow} and p_S^{slow} are the fractions of runs labeled slow by in-application timing and Slurm elapsed time. Acc. is the fraction of runs for which the two labels agree.

Application	**# Runs**	p_G^{slow}	p_S^{slow}	**Acc.**
PSDNS_short	93	0.00	0.02	0.98
PSDNS_long	62	0.00	0.00	1.00
AMG2023	184	0.03	0.03	1.00
lammps_short	91	0.02	0.02	1.00
lammps_long	61	0.017	0.02	1.00
MOE	96	0.01	0.08	0.93
BGW4	91	0.00	0.02	0.98
MILC	93	0.14	0.04	0.88
deepCAM_short	115	0.04	0.20	0.83
deepCAM_long	60	0.00	0.03	0.97
nanoGPT_short	182	0.05	0.10	0.95
nanoGPT_long	62	0.03	0.05	0.98

elapsed time because they spend unusually long in initialization before training begins. In these runs, dataset loading and preprocessing inflate Slurm elapsed time without affecting the main computation loop, which suggests that filesystem load or storage contention contributes to the mismatch.

Overall, these results support our use of Slurm elapsed time to construct slowdown labels in the production dataset. For most workloads, Slurm elapsed time preserves the same slowdown behavior captured by in-application timing. The remaining differences are limited and largely reflect effects outside the current NIC-based feature set.

5.4.2 Analyzing the Effect of Node Scope

We examine how node scope affects prediction when the NIC data window is fixed at 8 minutes. As defined in Section 5.3.3, the *job-nodes* scope uses only the nodes allocated to the run, while the *all-nodes* scope uses all GPU nodes in the system during the selected interval.

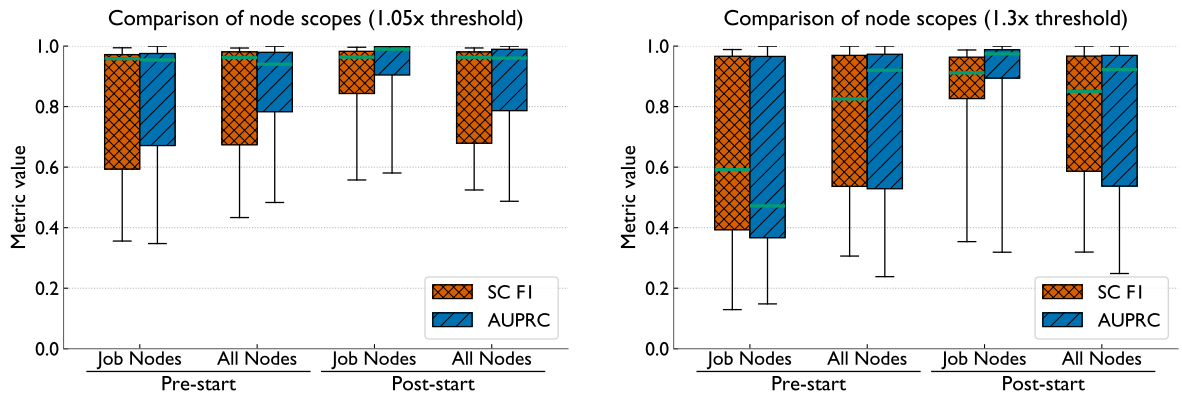


Figure 5.3: Prediction performance for the four 8-minute settings defined by time relative to run start and node scope. We compare pre-start and post-start NIC data using either job nodes or all nodes. We report the results for each slowdown threshold, 1.05 (left) and 1.3 (right). SC F1 denotes F1 for the slow class, and we summarize prediction performance using SC F1 and AUPRC.

We evaluate this comparison separately for the 1.05 and 1.3 slowdown thresholds. Within each threshold, all four settings are evaluated on the same set of groups.

Figure 5.3 compares the four combinations of node scope and timing relative to run start. Across both thresholds, NIC data collected after run start is more informative than NIC data collected before run start. The strongest overall results come from the post-start job-nodes setting. At the 1.05 threshold (left), the difference among settings is modest because all four settings achieve high median scores. The contrast is clearer at the 1.3 threshold (right). Median F1 increases from 0.59 for pre-start job nodes to 0.91 for post-start job nodes, and median AUPRC increases from 0.47 to 0.97. Results are also generally higher and more stable at the 1.05 threshold.

The preferred scope depends on whether the run has started. Before run start, all-nodes outperforms job-nodes at both thresholds. This difference is especially clear at the 1.3 threshold, where median F1 increases from 0.59 to 0.83 and median AUPRC increases from 0.47 to 0.92

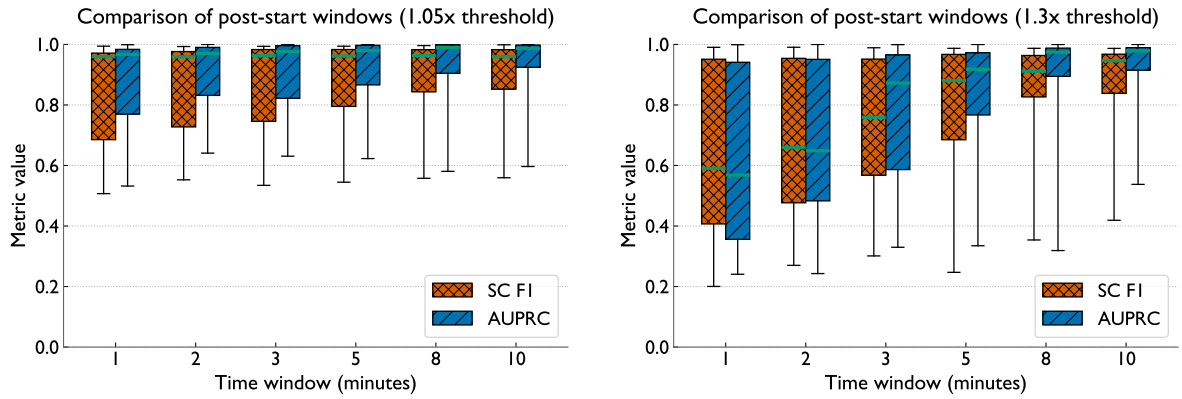


Figure 5.4: Prediction performance for post-start job-node NIC data using time windows of 1, 2, 3, 5, 8, and 10 minutes. We report the results for both slowdown thresholds, 1.05 (left) and 1.3 (right). SC F1 denotes F1 for the slow class, and we summarize prediction performance using SC F1 and AUPRC.

when we move from pre-start job nodes to pre-start all nodes. Before launch, the future allocation has not yet developed a job-specific NIC activity pattern, so the allocated nodes alone provide limited signal. System-wide NIC data better captures the surrounding contention state. After run start, the ordering reverses. Job-nodes outperforms all-nodes because NIC data from the allocated nodes reflects the behavior of the running job more directly. We therefore fix the node scope to job nodes in the following analyses and next examine the effect of post-start window length.

5.4.3 Analyzing the Effect of Time Window Length

We next examine how prediction quality changes with the length of the post-start NIC data window. Based on the previous subsection, we fix the node scope to job nodes and vary only the window length. We consider 1, 2, 3, 5, 8, and 10 minute windows. Within each threshold, all six windows are evaluated on the same set of groups, so the comparison isolates the effect of window length.

Figure 5.4 compares the six post-start job-node windows. Longer windows improve pre-

Table 5.2: Per-group dataset statistics for the selected 8-minute post-start job-node setting. Groups G1–G21 are sorted by mean duration. N is the number of runs, $\bar{d}$ is the mean run duration in minutes, and p^{slow} is the percentage of runs labeled slow. The last two columns report this percentage under the 1.05x and 1.3x labeling rules. A dash indicates that the group is not present under the corresponding labeling rule.

GID	N	$\bar{d}$ (mins)	$p^{\text{slow}}_{1.05x}$ (%)	$p^{\text{slow}}_{1.3x}$ (%)	GID	N	$\bar{d}$ (mins)	$p^{\text{slow}}_{1.05x}$ (%)	$p^{\text{slow}}_{1.3x}$ (%)	GID	N	$\bar{d}$ (mins)	$p^{\text{slow}}_{1.05x}$ (%)	$p^{\text{slow}}_{1.3x}$ (%)
G1	357	11.0	–	19.3	G8	566	33.0	–	40.8	G15	101	144.0	–	95.0
G2	294	13.0	–	72.4	G9	342	41.0	91.8	–	G16	255	144.6	25.5	–
G3	451	14.1	74.1	8.0	G10	817	57.4	92.4	20.9	G17	840	173.8	15.8	7.1
G4	1002	23.1	96.7	20.4	G11	131	58.6	21.4	6.1	G18	1710	195.0	92.6	25.2
G5	809	23.9	95.1	–	G12	1887	112.3	–	16.1	G19	356	208.2	98.3	98.3
G6	146	28.6	–	79.5	G13	200	118.5	–	33.5	G20	296	289.2	37.2	–
G7	519	31.8	99.0	38.3	G14	1874	140.4	41.8	–	G21	205	872.5	95.6	–

diction performance at 1.05 (left) and 1.3 (right). The effect is modest at 1.05 because even the 1-minute window already has high median scores. At this threshold, longer windows mainly improve consistency across groups. Median F1 remains near 0.96 across all windows, while the lower quartile rises from 0.69 at 1 minute to 0.85 at 10 minutes. At 1.3, the improvement is much larger. Median F1 increases from 0.59 to 0.95, and median AUPRC from 0.57 to 0.98. The largest gains occur between 1 and 5 minutes, after which improvements become smaller.

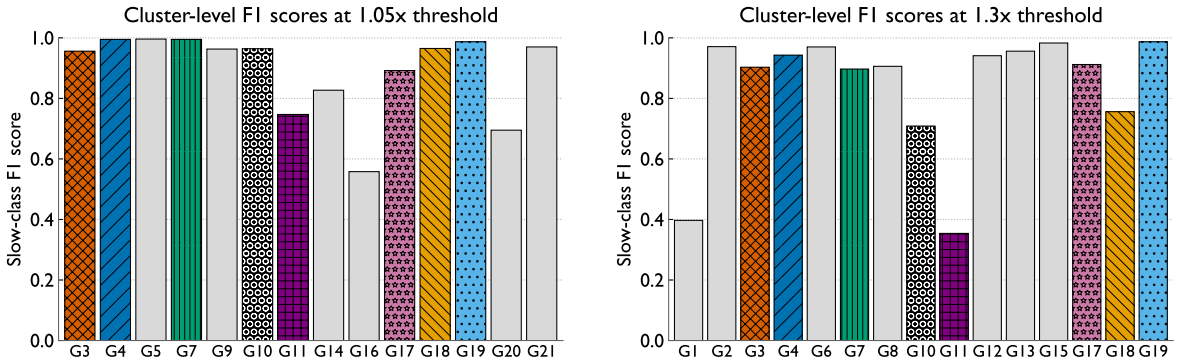


Figure 5.5: Slow-class F1 for the selected 8-minute post-start job-node setting under the 1.05x and 1.3x labeling rules. We order the groups, G1–G21, by mean duration. The same color-hatch combination denotes the same group in both plots. Most groups achieve strong performance under at least one rule, but a small number remain difficult to predict.

These results indicate that short post-start windows already contain useful NIC signal, but

longer windows capture more of the communication behavior associated with slowdown. The 10-minute window achieves the highest mean performance, especially at the 1.3 threshold. However, the 8-minute window already reaches nearly the same prediction performance level at both thresholds. We therefore use the 8-minute post-start job-node setting in the remaining analyses because it preserves most of the predictive benefit while enabling earlier prediction.

5.4.4 Evaluating Per-Group Classification Performance

We evaluate prediction performance at the group level under the selected 8-minute post-start job-node setting. Table 5.2 summarizes the 21 groups retained under the 1.05 and 1.3 labeling rules, and Figure 5.5 reports per-group slow-class F1. These groups are highly heterogeneous: size ranges from 101 to 1887 runs, mean duration from 11.0 to 872.5 minutes, and the fraction of slow runs from 15.8% to 99.0% under 1.05 and from 6.1% to 98.3% under 1.3. This variation supports group-specific baselines, labels, and models rather than a single global treatment.

Despite this heterogeneity, the selected setting performs well for most groups. Under the 1.05 rule, 9 of 14 groups achieve slow-class F1 of at least 0.95. Under the 1.3 rule, 10 of 15 groups achieve F1 of at least 0.90.Several groups remain strong under both rules. G3, G4, G17, and G19 all achieve F1 around 0.90 or above across thresholds. At the same time, a small number of groups remain difficult to predict. Weak groups appear at both short and long durations, while some long-duration groups are predicted very well. Prediction quality therefore does not follow a simple trend with runtime or sample count alone.

Among the eight groups retained under both rules, the labeling rule can change prediction performance substantially because it changes the fraction of runs labeled slow. G10 drops from

0.964 at 1.05 to 0.708 at 1.3 as the slow-run fraction decreases from 92.4% to 20.9%. G18 follows the same pattern, dropping from 0.965 to 0.756 as the slow-run fraction decreases from 92.6% to 25.2%. In contrast, G19 has the same slow-run fraction under both rules, 98.3%, and its F1 remains 0.987. These comparisons indicate that threshold choice affects prediction not only by changing the slowdown definition, but also by reshaping class balance within each group.

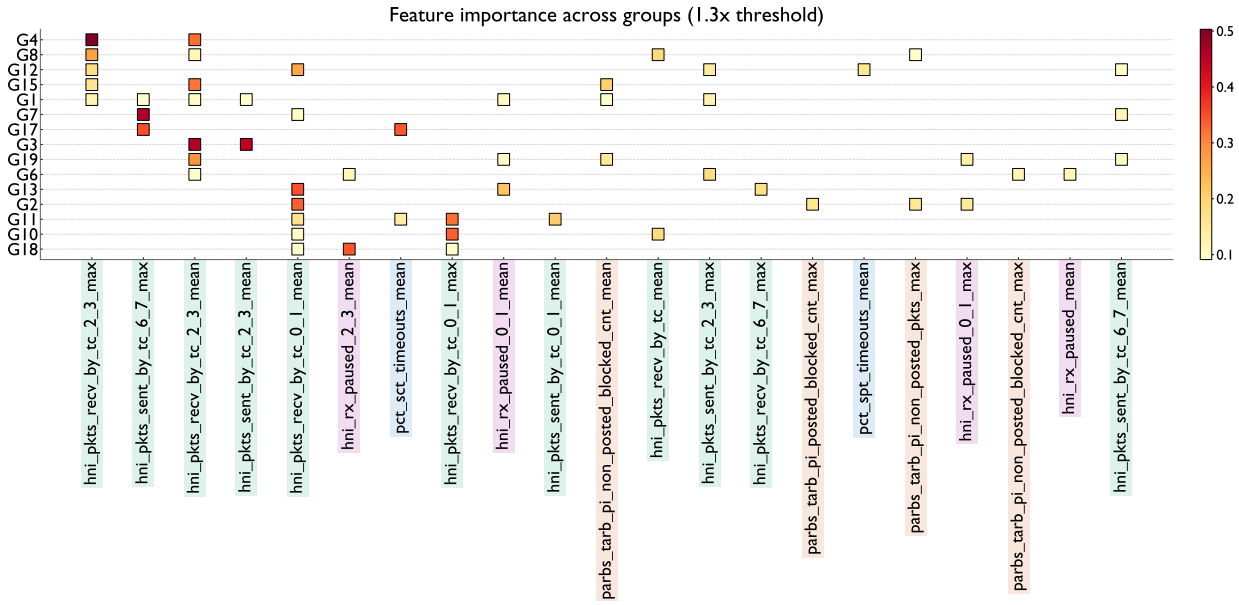


Figure 5.6: Feature importance for the selected 8-minute post-start job-node setting. Each row corresponds to a group and each column corresponds to a feature included in the heatmap. Cell values report mean feature importance across folds.

Beyond class balance, manual inspection suggests that several weak groups reflect residual heterogeneity within the grouping key. In G11, the submit line records only a launcher command and log directory, so it does not expose the underlying application or configuration. Other weak groups contain runs from the same scientific workload class but with different input files. For example, G10 includes runs with different XML inputs, and G14, G16, and G20 include lattice QCD calculations with different gauge configurations, source files, and momentum settings. G1 reflects a similar issue in materials-science applications, where one code can support substantially

different calculation setups. These cases indicate that scheduler-visible launch information does not always isolate a homogeneous workload.

5.4.5 Analyzing Feature Importances

We next examine which NIC features are most informative under the selected post-start Job Nodes setting at the 1.3 slowdown threshold. For each group, we apply recursive feature elimination to the full NIC feature set, retain 10 features, and then average XGBoost feature importance across the outer evaluation folds. Figure 5.6 reports these mean importance.

Figure 5.6 indicates substantial variation across groups, but it also reveals a clear pattern. Timeout and pause counters are not the dominant features in many groups, which differs from our expectation based on prior work that identified timeouts and pauses as important contributors to variability [10]. Instead, the most important features in many groups fall under send and receive packet counters. This pattern is visible in groups such as G3, where send and receive counters dominate, whereas timeout, pause, and PARBS counters are most prominent in smaller subsets of groups such as G17, G18, and G2.

Manual inspection of controlled applications helps explain this result. Jobs with variability almost always contain more timeouts than normal jobs, but the presence of timeouts does not reliably separate variable runs from normal ones. In that sense, timeout appears necessary but not sufficient for variability. By contrast, send and receive packet rates are often substantially lower in variable jobs than in normal jobs, which we believe is the main reason the model can predict variability. More broadly, network congestion can degrade communication performance even without clear timeout or pause signals. This interpretation is also consistent with our traffic-

class aggregation result. When we aggregate traffic classes, F1 decreases, which indicates that fine-grained send and receive packet structure carries useful predictive information.

5.5 Discussion

In this section, we discuss the main methodological and operational implications of our results for predicting performance variation in arbitrary GPU applications around launch time.

The importance of grouping comparable runs. A good grouping strategy is essential because slowdown only has meaning relative to comparable runs. Production applications differ in node count, runtime, input configuration, and launch behavior, so a single global baseline would mix fundamentally different applications. Our per-group results and manual inspection support this point. At the same time, the current grouping method provides only a scheduler-visible approximation, and residual within-group heterogeneity still reduces prediction performance in some cases.

NIC data and Slurm elapsed time as practical signals. Our results indicate that NIC data is informative for predicting network-related performance variation in GPU applications, as shown by the strong prediction results across diverse groups. Additionally, Slurm elapsed time is informative for label construction. For most controlled applications, labels from Slurm elapsed time agree closely with labels from timing the main computation loop. The remaining disagreements mainly reflect variation in initialization, data loading, and preprocessing phases, which can inflate Slurm elapsed time without changing the main loop. Predicting such cases would require additional telemetry data from other system components such as the filesystem or storage system.

When NIC data is most informative. The timing and scope analysis clarifies when NIC data is

most useful. After a run starts, NIC data from the allocated job nodes is more informative than NIC data from all nodes because it reflects the communication behavior of the running application more directly. Before a run starts, the future allocation has not yet developed a job-specific activity pattern, so all-node data is often more informative than job-node data. These results suggest that capturing application behavior after launch is more informative than observing the whole system alone, although system-wide NIC data remains useful for some groups.

Early prediction from short post-start windows. The time-window analysis indicates that the first few minutes of execution already contain substantial predictive signal, even for long-running applications. Prediction performance improves rapidly early in execution, and the gains become smaller for longer windows. This pattern suggests that early communication behavior often reflects the performance state that persists later in the run. From an operational perspective, the 8-minute window achieves nearly the same performance as the 10-minute window while providing an earlier prediction point. For long jobs, this still leaves time for a warning, a scheduling adjustment, or additional diagnosis.

Threshold choice changes the prediction problem. Threshold selection matters because it changes which runs are labeled slow. A lower threshold captures modest slowdowns, while a higher threshold isolates more pronounced slowdowns. This change affects class balance and prediction difficulty. Additionally, applications with low run-to-run variability may benefit from a lower threshold, while applications with higher inherent variability may require a larger threshold to separate meaningful slowdowns from normal variation. In practice, threshold choice should reflect both application behavior and the operational goal.

5.6 Conclusion and Future Work

We present a method to predict whether a GPU run will exhibit slowdown relative to runs with similar scheduler-visible execution settings. The method combines system-wide NIC data with Slurm job-step metadata, groups comparable runs, constructs slowdown labels from Slurm elapsed time, and trains group-specific models from NIC data collected around run start. Because it relies only on production monitoring and scheduler data, it applies to arbitrary applications without application-specific profiling. On Perlmutter, it achieves strong prediction performance, especially with post-start NIC data from the allocated job nodes.The method is broadly applicable to HPC systems that provide comparable scheduler metadata and network monitoring data.

The main limitations come from grouping and the scope of the available telemetry data. The current feature set focuses on NIC counters, so it does not capture slowdowns driven by other components such as I/O or storage. Future work can improve grouping with additional post-start information, incorporate telemetry data from other components, refine threshold selection, and extend the method into a deployable framework for warning, diagnosis, and scheduling support.

Chapter 6: Conclusion

This dissertation has examined how longitudinal telemetry can be used to better understand and manage modern high-performance computing systems. Motivated by the growing scale, heterogeneity, and operational complexity of GPU-accelerated supercomputers, it has focused on a central question: what can routinely collected monitoring data reveal about workload behavior and performance variability when they are analyzed at the right level of abstraction and integrated with scheduler metadata? The results show that always-on telemetry is more than a record of low-level counters. When attributed to jobs carefully and interpreted in context, it becomes a practical basis for workload characterization, diagnosis, and prediction in production environments.

The first major contribution of this dissertation is a job-level characterization of GPU workloads on Perlmutter using system-wide DCGM telemetry and Slurm metadata. This analysis showed that the GPU partition is dominated in job count by small allocations, especially single-node jobs, while larger jobs consume a disproportionate share of system resources. It further showed that GPU behavior varies substantially across workloads in ways that are not captured by a single utilization summary. FP64-only jobs make up the largest share of the workload, tensor-pipe activity is associated with higher utilization, and many jobs that explicitly request 80 GB GPUs use well below that capacity. Together, these findings illustrate that production telemetry can reveal practical information about how applications use accelerator resources and where

inefficiencies or overprovisioning may exist.

Another contribution of the workload study is the development of job-aware metrics that quantify how GPU activity is distributed across devices and over time. The results showed that low-utilization jobs are more likely to exhibit high spatial imbalance and high temporal imbalance, whereas highly utilized jobs tend to distribute work more evenly and sustain activity more consistently. The roofline-based classification of jobs as compute-bound or memory-bound further clarified how utilization relates to resource bottlenecks. Most jobs on Perlmutter are memory-bound, and at comparable GPU-hour scales these jobs tend to consume more energy than compute-bound jobs. At the same time, correlations among utilization, SM activity, DRAM activity, power, and temperature demonstrated that broad sets of hardware counters can be combined to form coherent job-level views of workload behavior. These results move beyond simple resource accounting and toward interpretable performance diagnostics for both users and operators.

The second major contribution of this dissertation is a telemetry-driven method for predicting run-to-run performance variability in arbitrary production GPU applications. This part of the work addressed a difficult operational problem: similar runs can experience substantially different runtimes because they execute under changing shared-system conditions, yet application-specific profiling is rarely available at production scale. By clustering runs with scheduler-visible metadata, defining slowdown relative to a cluster-specific baseline, and constructing features from NIC telemetry collected near run start, this dissertation showed that useful predictive signal exists in routine monitoring data alone. The methodology was validated on controlled repeated runs and then evaluated on production data from Perlmutter. Under the best configuration, telemetry from the allocated job nodes during the first eight minutes after run start achieved strong predic-

tion quality across 14 retained clusters. These results demonstrate that telemetry can support not only retrospective analysis but also early warning of likely performance problems.

The descriptive and predictive parts of this dissertation indicate that longitudinal telemetry becomes valuable when it is made job-aware. Careful attribution and aggregation make them useful for answering higher-level questions about workload behavior, resource efficiency, and slowdown risk. This perspective suggests a practical path forward for production HPC environments. For users, telemetry-driven analysis can inform optimization, reveal load imbalance, and help choose more appropriate resource requests. For operators, it can support capacity planning, energy-aware decision making, and earlier identification of problematic runs or inefficient workload classes.

This work also has important limitations. The empirical analysis is centered on a single leadership-class system, and although the methods are designed to generalize, site-specific counter availability, access constraints, and workload composition will affect how easily they transfer. The available telemetry is sampled at fixed intervals and cannot capture all short-lived behavior. Some analyses rely on coarse approximations, such as keyword-based ML labeling or scheduler-visible executable signatures that do not fully resolve input-level or algorithmic differences.

Several directions follow naturally from this dissertation. Higher-frequency telemetry and richer combinations of GPU, network, storage, and scheduler data could improve both descriptive fidelity and predictive accuracy. In the variability study, improved clustering strategies could reduce hidden heterogeneity for generic executable signatures and make slowdown prediction more robust. At the system level, the methods developed here could be integrated into user-facing dashboards, advisories at submission time, or scheduler services that flag likely slow runs,

highlight underused resources, or recommend better-fit allocations.

In conclusion, this dissertation has shown that always-on telemetry enables understanding how production workloads use complex resources, and anticipating when comparable runs are likely to perform poorly. By combining system-wide monitoring data with scheduler metadata and analyzing the resulting job-level views, it has demonstrated a practical approach to longitudinal data analytics that is scalable, non-intrusive, and operationally relevant. As HPC systems continue to grow in scale and heterogeneity, such telemetry-based methods will become increasingly important for turning routine monitoring data into actionable understanding and more effective system management.

Bibliography

[1] Omar Aaziz, Jonathan Cook, and Mohammed Tanash. "Modeling expected application runtime for characterizing and assessing job performance". In: *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2018, pp. 543–551.

[2] A. Agelastos et al. "The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications". In: *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2014, pp. 154–165.

[3] George Amvrosiadis et al. "On the diversity of cluster workloads and its impact on research results". In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 2018, pp. 533–546.

[4] Oscar Antepara et al. "Benchmark-driven Models for Energy Analysis and Attribution of GPU-Accelerated Supercomputing". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '25. New York, NY, USA: Association for Computing Machinery, 2025, 888–904. ISBN: 9798400714665. DOI: 10.1145/3712285.3759815. URL: https://doi.org/10.1145/3712285.3759815.

[5] Francesco Antici et al. "MCBound: An Online Framework to Characterize and Classify Memory/Compute-bound HPC Jobs". In: *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2024, pp. 1–15.

[6] Emre Ates et al. "Taxonomist: Application detection through rich monitoring data". In: *Euro-Par 2018: Parallel Processing: 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings 24*. Springer. 2018, pp. 92–105.

[7] Brian Austin et al. "System-Wide Roofline Profiling-a Case Study on NERSC's Perlmutter Supercomputer". In: *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2024, pp. 1398–1404.

[8] Claude Bernard et al. "Scaling tests of the improved Kogut-Susskind quark action". In: *Physical Review D* 61 (2000).

[9] Abhinav Bhatele et al. "Identifying the Culprits behind Network Congestion". In: *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*. IPDPS '15. IEEE Computer Society, May 2015. URL: http://doi.ieeecomputersociety.org/10.1109/IPDPS.2015.92.

[10] Abhinav Bhatele et al. “The Case of Performance Variability on Dragonfly-based Systems”. In: *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*. IPDPS ’20. IEEE Computer Society, May 2020.

[11] Abhinav Bhatele et al. “There goes the neighborhood: performance degradation due to nearby jobs”. In: *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’13. IEEE Computer Society, Nov. 2013. URL: http://doi.acm.org/10.1145/2503210.2503247.

[12] Abhinav Bhatele et al. “Understanding Application Performance via Micro-benchmarks on Three Large Supercomputers: Intrepid, Ranger and Jaguar”. In: *Int. J. High Perform. Comput. Appl.* 24.4 (Nov. 2010), pp. 411–427. URL: http://hpc.sagepub.com/content/24/4/411.

[13] Joshua Dennis Booth et al. “Phase detection with hidden markov models for dvfs on many-core processors”. In: *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE. 2015, pp. 185–195.

[14] Andrea Borghesi et al. “Anomaly detection and anticipation in high performance computing systems”. In: *IEEE Transactions on Parallel and Distributed Systems* 33.4 (2021), pp. 739–750.

[15] Rajkumar Buyya. “PARMON: a portable and scalable monitoring system for clusters”. In: *Software: Practice and Experience* 30.7 (2000), pp. 723–739.

[16] Onur Cankur et al. “Characterizing Production GPU Workloads using System-wide Telemetry Data”. In: *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*. IPDPS ’26. IEEE Computer Society, May 2026.

[17] Yanpei Chen et al. “Analysis and lessons from a publicly available google cluster trace”. In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95* 94 (2010).

[18] Ghislain Landry Tsafack Chetsa et al. “A user friendly phase detection methodology for HPC systems’ analysis”. In: *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. IEEE. 2013, pp. 118–125.

[19] Xiaoyu Chu et al. “Generic and ML Workloads in an HPC Datacenter: Node Energy, Job Failures, and Node-Job Analysis”. In: *arXiv preprint arXiv:2409.08949* (2024).

[20] Sudheer Chunduri et al. “Run-to-run Variability on Xeon Phi Based Cray XC Systems”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’17. Denver, Colorado: ACM, 2017.

[21] Eli Cortez et al. “Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms”. In: *Proceedings of the 26th Symposium on Operating Systems Principles*. 2017, pp. 153–167.

[22] Ian J. Costello and Abhinav Bhatele. *Analytics of Longitudinal System Monitoring Data for Performance Prediction*. 2020. eprint: arXiv:2007.03451.

[23] Orianna DeMasi, Taghrid Samak, and David H Bailey. “Identifying HPC codes via performance logs and machine learning”. In: *Proceedings of the first workshop on Changing landscapes in HPC security*. 2013, pp. 23–30.

[24] Jack Deslippe et al. “BerkeleyGW: A massively parallel computer package for the calculation of the quasiparticle and optical properties of materials and nanostructures”. In: *Computer Physics Communications* 183.6 (June 2012), 1269–1289. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2011.12.006. URL: http://dx.doi.org/10.1016/j.cpc.2011.12.006.

[25] Sheng Di, Derrick Kondo, and Walfredo Cirne. “Characterization and comparison of cloud versus grid workloads”. In: *2012 IEEE International Conference on Cluster Computing*. IEEE. 2012, pp. 230–238.

[26] Joseph Emeras et al. “Evalix: classification and prediction of job resource consumption on HPC platforms”. In: *Job Scheduling Strategies for Parallel Processing: 19th and 20th International Workshops, JSSPP 2015, Hyderabad, India, May 26, 2015 and JSSPP 2016, Chicago, IL, USA, May 27, 2016, Revised Selected Papers 19*. Springer. 2017, pp. 102–122.

[27] Eric Gaussier et al. “Improving backfilling by using machine learning to predict running times”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2015, pp. 1–10.

[28] Neha Gholkar, Frank Mueller, and Barry Rountree. “Uncore power scavenger: A runtime for uncore power conservation on hpc systems”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2019, pp. 1–23.

[29] Alfredo Giménez et al. “ScrubJay: Deriving Knowledge from the Disarray of HPC Performance Data”. In: *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’17. LLNL-CONF-735962. IEEE Computer Society, Nov. 2017. URL: http://doi.acm.org/10.1145/3126908.3126935.

[30] K-I Goh and A-L Barabási. “Burstiness and memory in complex systems”. In: *Europhysics Letters* 81.4 (2008), p. 48002.

[31] T. Groves, Y. Gu, and N. J. Wright. “Understanding Performance Variability on the Aries Dragonfly Network”. In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. 2017, pp. 809–813.

[32] Francesc Guim, Ivan Rodero, and Julita Corbalan. “The resource usage aware backfilling”. In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 2009, pp. 59–79.

[33] Jian Guo et al. “Machine learning predictions for underestimation of job runtime on HPC system”. In: *Asian Conference on Supercomputing Frontiers*. Springer. 2018, pp. 179–198.

[34] Mohamed S Halawa, Rebeca P Diaz Redondo, and Ana Fernández Vilas. “Unsupervised kpis-based clustering of jobs in hpc data centers”. In: *Sensors* 20.15 (2020), p. 4111.

[35] Qinghao Hu et al. “Characterization and prediction of deep learning workloads in large-scale gpu datacenters”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2021, pp. 1–15.

[36] Shashikant Ilager et al. “A data-driven analysis of a cloud data center: statistical characterization of workload, energy and temperature”. In: *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*. 2023, pp. 1–10.

[37] Nikhil Jain et al. “Maximizing Throughput on a Dragonfly Network”. In: *SC ’14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2014, pp. 336–347. DOI: 10.1109/SC.2014.33.

[38] Sunggon Kim et al. “Towards HPC I/O Performance Prediction through Large-scale Log Analysis”. In: *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*. HPDC ’20. Stockholm, Sweden: Association for Computing Machinery, 2020, 77–88. ISBN: 9781450370523. DOI: 10.1145/3369583.3392678. URL: https://doi.org/10.1145/3369583.3392678.

[39] Thorsten Kurth et al. *Exascale Deep Learning for Climate Analytics*. 2018. arXiv: 1810.01993 [cs.DC]. URL: https://arxiv.org/abs/1810.01993.

[40] Edgar A. Leon et al. “Characterizing Parallel Scientific Applications on Commodity Clusters: An Empirical Study of a Tapered Fat-tree”. In: *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’16. LLNL-CONF-681011. IEEE Computer Society, Nov. 2016. URL: http://doi.ieeecomputersociety.org/10.1109/SC.2016.77.

[41] Baolin Li et al. “AI-enabling workloads on large-scale GPU-accelerated system: Characterization, opportunities, and implications”. In: *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE. 2022, pp. 1224–1237.

[42] Boyang Li et al. “Mrsch: Multi-resource scheduling for hpc”. In: *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2022, pp. 47–57.

[43] Boyang Li et al. “The Effect of System Utilization on Application Performance Variability”. In: *Proceedings of the 9th International Workshop on Runtime and Operating Systems for Supercomputers*. ROSS ’19. Phoenix, AZ, USA: Association for Computing Machinery, 2019, 11–18. ISBN: 9781450367554. DOI: 10.1145/3322789.3328743. URL: https://doi.org/10.1145/3322789.3328743.

[44] Jiangtian Li et al. “Machine learning based online performance prediction for runtime parallelization and task scheduling”. In: *2009 IEEE international symposium on performance analysis of systems and software*. IEEE. 2009, pp. 89–100.

[45] Jie Li, Brandon Cook, and Yong Chen. “ARcode: HPC Application Recognition Through Image-encoded Monitoring Data”. In: *arXiv preprint arXiv:2301.08612* (2023).

[46] Jie Li et al. “Analyzing resource utilization in an HPC system: A case study of NERSC’s Perlmutter”. In: *International Conference on High Performance Computing*. Springer. 2023, pp. 297–316.

[47] Ruipeng Li and Ulrike M. Yang. *AMG2023*. [Computer Software] https://doi.org/10.11578/dc.20230413.1. 2023. DOI: 10.11578/dc.20230413.1. URL: https://doi.org/10.11578/dc.20230413.1.

[48] Erika Susana Alcorta Lozano and Andreas Gerstlauer. "Learning-based phase-aware multi-core cpu workload forecasting". In: *ACM transactions on design automation of electronic systems* 28.2 (2022), pp. 1–27.

[49] Samuel Maloney et al. "Analyzing HPC Monitoring Data With a View Towards Efficient Resource Utilization". In: *2024 IEEE 36th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE. 2024, pp. 170–181.

[50] Ryan McKenna et al. "Machine Learning Predictions of Runtime and IO Traffic on High-End Clusters". In: *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. 2016, pp. 255–258. DOI: 10.1109/CLUSTER.2016.58.

[51] Asit K Mishra et al. "Towards characterizing cloud backend workloads: insights from google compute clusters". In: *ACM SIGMETRICS Performance Evaluation Review* 37.4 (2010), pp. 34–41.

[52] Martin Molan et al. "Graafe: Graph anomaly anticipation framework for exascale hpc systems". In: *Future Generation Computer Systems* 160 (2024), pp. 644–653.

[53] Martin Molan et al. "RUAD: Unsupervised anomaly detection in HPC systems". In: *Future Generation Computer Systems* 141 (2023), pp. 542–554.

[54] Mikael Mortensen and Hans Petter Langtangen. "High performance Python for direct numerical simulations of turbulent flows". In: *Computer Physics Communications* 203 (June 2016), 53–65. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2016.02.005. URL: http://dx.doi.org/10.1016/j.cpc.2016.02.005.

[55] Alessio Netti et al. "A conceptual framework for HPC operational data analytics". In: *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2021, pp. 596–603.

[56] Alessio Netti et al. "A machine learning approach to online fault classification in HPC systems". In: *Future Generation Computer Systems* 110 (2020), pp. 1009–1022.

[57] Alessio Netti et al. "Operational data analytics in practice: experiences from design to deployment in production HPC environments". In: *Parallel Computing* 113 (2022), p. 102950.

[58] Daniel Nichols et al. "Resource Utilization Aware Job Scheduling to Mitigate Performance Variability". In: *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*. IPDPS '22. IEEE Computer Society, May 2022.

[59] NVIDIA Corporation. *NVIDIA Data Center GPU Manager (DCGM)*. https://developer.nvidia.com/dcgm. Accessed: 2024-10-15. 2023.

[60] Gence Ozer et al. "Characterizing HPC performance variation with monitoring and unsupervised learning". In: *High Performance Computing: ISC High Performance 2020 International Workshops, Frankfurt, Germany, June 21–25, 2020, Revised Selected Papers 35*. Springer. 2020, pp. 280–292.

[61] Tirthak Patel et al. “Revisiting I/O behavior in large-scale storage systems: The expected and the unexpected”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2019, pp. 1–13.

[62] Archit Patke et al. “Delay sensitivity-driven congestion mitigation for hpc systems”. In: *Proceedings of the 35th ACM International Conference on Supercomputing*. 2021, pp. 342–353.

[63] Indrani Paul et al. “Coordinated energy management in heterogeneous processors”. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 2013, pp. 1–12.

[64] Ivy Peng et al. “A holistic view of memory utilization on HPC systems: Current and future trends”. In: *Proceedings of the International Symposium on Memory Systems*. 2021, pp. 1–11.

[65] Fabrizio Petrini, Darren J. Kerbyson, and Scott Pakin. “The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q”. In: *Proceedings of the 2003 ACM/IEEE conference on Supercomputing (SC’03)*. Phoenix, AZ, USA, 2003.

[66] Charles Reiss et al. “Heterogeneity and dynamicity of clouds at scale: Google trace analysis”. In: *Proceedings of the third ACM symposium on cloud computing*. 2012, pp. 1–13.

[67] Zujie Ren et al. “Workload characterization on a production hadoop cluster: A case study on taobao”. In: *2012 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. 2012, pp. 3–13.

[68] Marcus Ritter et al. “Conquering Noise With Hardware Counters on HPC Systems”. In: *2022 IEEE/ACM Workshop on Programming and Performance Visualization Tools (ProTools)*. 2022, pp. 1–10. DOI: 10.1109/ProTools56701.2022.00007.

[69] Ehsan Saeedizade, Roya Taheri, and Engin Arslan. “I/o burst prediction for hpc clusters using darshan logs”. In: *2023 IEEE 19th International Conference on e-Science (e-Science)*. IEEE. 2023, pp. 1–10.

[70] Efe Sencan et al. “Analyzing GPU Utilization in HPC Workloads: Insights from Large-Scale Systems”. In: *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration*. PEARC ’25. New York, NY, USA: Association for Computing Machinery, 2025. ISBN: 9798400713989. DOI: 10.1145/3708035.3736010. URL: https://doi.org/10.1145/3708035.3736010.

[71] Efe Sencan et al. “Refine: A Robust Approach to Unsupervised Anomaly Detection for Production HPC Systems”. In: *ISC High Performance 2025 Research Paper Proceedings (40th International Conference)*. 2025, pp. 1–12. DOI: 10.23919/ISC.2025.11018307.

[72] Denis Shaikhislamov and Vadim Voevodin. “Smart Clustering of HPC Applications Using Similar Job Detection Methods”. In: *International Conference on Parallel Processing and Applied Mathematics*. Springer. 2022, pp. 209–221.

[73] Siqi Shen, Vincent Van Beek, and Alexandru Iosup. "Statistical characterization of business-critical workloads hosted in cloud datacenters". In: *2015 15th IEEE/ACM international symposium on cluster, cloud and grid computing*. IEEE. 2015, pp. 465–474.

[74] Mohammad Shoeybi et al. *Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism*. Tech. rep. 2020. arXiv: 1909.08053 [cs.CL].

[75] Siddharth Singh and Abhinav Bhatele. "AxoNN: An asynchronous, message-driven parallel framework for extreme-scale deep learning". In: *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*. IPDPS '22. IEEE Computer Society, May 2022.

[76] Siddharth Singh et al. "Democratizing AI: Open-source Scalable LLM Training on GPU-based Supercomputers". In: *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '24. Nov. 2024.

[77] Prasoon Sinha et al. "Not all GPUs are created equal: characterizing variability in large-scale, accelerator-rich systems". In: *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2022, pp. 01–15.

[78] Alina Sîrbu and Ozalp Babaoglu. "Power consumption modeling and prediction in a hybrid CPU-GPU-MIC supercomputer". In: *Euro-Par 2016: Parallel Processing: 22nd International Conference on Parallel and Distributed Computing, Grenoble, France, August 24-26, 2016, Proceedings 22*. Springer. 2016, pp. 117–130.

[79] *Slurm Workload Manager*. 2020. URL: https://slurm.schedmd.com/documentation.html.

[80] Matthew J Sottile and Ronald G Minnich. "Supermon: A high-speed cluster monitoring system". In: *Proceedings. IEEE International Conference on Cluster Computing*. IEEE. 2002, pp. 39–46.

[81] Michela Taufer. "AI4IO: A suite of ai-based tools for io-aware HPC resource management". In: *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE. 2021, pp. 1–1.

[82] A. P. Thompson et al. "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales". In: *Comp. Phys. Comm.* 271 (2022), p. 108171. DOI: 10.1016/j.cpc.2021.108171.

[83] Dan Tsafrir, Yoav Etsion, and Dror G Feitelson. "Backfilling using system-generated predictions rather than user runtime estimates". In: *IEEE Transactions on Parallel and Distributed Systems* 18.6 (2007), pp. 789–803.

[84] Hao Wang et al. "Predicting running time of aerodynamic jobs in HPC system by combining supervised and unsupervised learning method". In: *Advances in Aerodynamics* 3.1 (2021), p. 22.

[85] Cunyang Wei, Keshav Pradeep, and Abhinav Bhatele. "Unmasking Performance Variability in GPU Codes on Production Supercomputers". In: ().

[86] Kaiyue Wu, Jianwen Wei, and James Lin. "SchedP: I/O-aware Job Scheduling in Large-Scale Production HPC Systems". In: *IFIP International Conference on Network and Parallel Computing*. Springer. 2022, pp. 315–326.

[87] Michael R. Wyatt et al. “PRIONN: Predicting Runtime and IO Using Neural Networks”. In: *Proceedings of the 47th International Conference on Parallel Processing*. ICPP ’18. Eugene, OR, USA: Association for Computing Machinery, 2018. ISBN: 9781450365109. DOI: 10.1145/3225058.3225091. URL: https://doi.org/10.1145/3225058.3225091.

[88] Qingqing Xiong et al. “Tangram: Colocating hpc applications with oversubscription”. In: *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE. 2018, pp. 1–7.

[89] Kohei Yoshida et al. “VAHRM: Variation-Aware Resource Management in Heterogeneous Supercomputing Systems”. In: *IEEE Transactions on Parallel & Distributed Systems* 36.08 (Aug. 2025), pp. 1713–1727. URL: https://doi.ieeecomputersociety.org/10.1109/TPDS.2025.3577252.