

Project Report

Time-Series Deep Learning for GPU Failure Prediction

Karim Elezabawy

MatrNr: 24175459

Supervisor: Michael Bidollahkhani

Georg-August-Universität Göttingen
Institute of Computer Science

September 29, 2026

Abstract

Graphics processing units (GPUs) have become the main source of compute in modern high-performance computing (HPC) systems, and with this role they have also become their weakest link. Because a single failing GPU can abort a job running on thousands of others, anticipating failures before they occur is an attractive alternative to reactive and fixed-schedule maintenance. This report studies data-driven GPU failure prediction on the public telemetry of the Summit supercomputer, a 4,626-node system with six NVIDIA V100 GPUs per node, using the curated ClusterWise release that covers five month-long segments between 2020 and 2022 at a 10-second resolution. The dataset is first characterised: 260,039 logged GPU errors spanning 13 NVIDIA error types, of which fewer than 0.6% are hardware related, strongly varying monthly failure volumes, and heterogeneous failure signatures ranging from abrupt power drops to patterns indistinguishable from normal operation. A preprocessing pipeline then converts the raw per-node streams into supervised instances through look-ahead windowing with a 30-minute observation and a 60-minute prediction window, negative subsampling, and model-specific tensor shaping. Four supervised models, a gradient-boosted decision tree ensemble, a multi-layer perceptron, a one-dimensional convolutional neural network and a long short-term memory network, are trained and evaluated with Precision@K, following the protocol of Liu et al. The convolutional network and the tree ensemble perform best, reaching a Precision@K of about 0.61 and 0.59 at a positive rate of 11%, and a parallel ensemble of both improves precision further to 0.62. An extended evaluation restricted to hardware failures, with per-GPU models evaluated across months at the natural failure rate, draws a more sober picture: node-level models barely beat chance, per-GPU models gain mainly from failure history rather than telemetry, and only interconnect errors are predicted well, with 44% of them receiving an alarm about a day in advance. The report closes with directions towards richer signals and cross-dataset evaluation.

Declaration on the use of ChatGPT and comparable tools in the context of examinations

In this work I have used ChatGPT or another AI as follows:

- ☐ Not at all
- ☒ During brainstorming
- ☒ When creating the outline
- ☐ To write individual passages, altogether to the extent of **XX**% of the entire text
- ☒ For the development of software source texts
- ☐ For optimizing or restructuring software source texts
- ☒ For proofreading or optimizing
- ☐ Further, namely:-

I hereby declare that I have stated all uses completely.

Missing or incorrect information will be considered as an attempt to cheat.

Contents

List of Tables	v
List of Figures	v
List of Abbreviations	vi
1 Introduction	1
2 Background	2
2.1 Types of Maintenance	2
2.2 GPU Failures and XID Errors	2
2.3 Related Work	3
3 The Dataset	3
3.1 Public Datasets for GPU Failure Prediction	3
3.2 The Summit Supercomputer	3
3.3 Data Collection	3
3.4 Time Span and Availability	4
3.5 Features	4
3.6 The Failure Landscape	4
4 Data Analytics	5
4.1 What a Failure Looks Like	5
4.2 Failure Co-Occurrence	6
4.3 Monthly Variability	6
5 Methodology	7
5.1 Handling Missing Values	8
5.2 From Raw Streams to Supervised Instances	8
5.3 Handling Class Imbalance	9
5.4 Scaling and Model-Ready Tensors	9
5.5 Models	9
5.6 Evaluation Metric: Precision@K	10
5.7 Parallel Ensemble	10
6 Results	10
6.1 Model Comparison	10
6.2 Ensembling GBDT and 1D-CNN	11
6.3 Comparison with Liu et al.	12
6.4 Hardware Failures Across Months	12
7 Discussion	13
7.1 Key Observations	13
7.2 Limitations	14
8 Conclusion	14

9	Future Work	14
	References	15
A	NVIDIA XID Errors	A1

List of Tables

1	Features of each record.	5
2	Input representation of the windows for each model (L time steps, $F = 28$ features).	9
3	Leave-one-month-out results for hardware failures with a 24-hour horizon, pooled over all five months. The last column gives the share of failures that receive an alarm when 1% of all scored hours raise one, with the share expected from random alarms in parentheses.	13
4	The 13 NVIDIA XID error codes that occur in the Summit dataset, with their meaning according to the NVIDIA documentation [NVI] and their number of occurrences across the five months.	A1

List of Figures

1	The three maintenance strategies: reactive, preventive, and predictive. . . .	2
2	Layout of a Summit compute node: two Power9 CPUs, each connected to three NVIDIA V100 GPUs, and two power supplies.	4
3	The 13 GPU error types logged across the five months, split into application-associated and driver- or hardware-related errors (260,039 errors in total). . .	5
4	GPU core temperature, memory temperature and power around six example failures of different XID types, standardised per window. The red bands mark the timestamps labelled as failure.	6
5	GPU failure co-occurrence on Summit.	7
6	Failure log entries and affected hosts per month.	7
7	Overview of the preprocessing and modelling pipeline.	8
8	Look-ahead windowing and labelling. A failure in the prediction window yields a positive instance (B); a failure in the observation window discards the instance (C).	8
9	Parallel ensemble after Liu et al. [Liu+23]: a window is flagged only if both models independently rank it within their top- K	10
10	Precision@K of the four models on the two test days at a test ratio of 8:1 (random ranking: 0.11).	11
11	Precision@K of the parallel ensemble compared with its two members. . . .	11

List of Abbreviations

HPC High-Performance Computing

AI Artificial Intelligence

ML Machine Learning

GPU Graphics Processing Unit

CPU Central Processing Unit

OLCF Oak Ridge Leadership Computing Facility

BMC Baseboard Management Controller

HBM High Bandwidth Memory

XID NVIDIA Error Identifier

DBE Double-Bit Error

GBDT Gradient-Boosted Decision Trees

MLP Multi-Layer Perceptron

CNN Convolutional Neural Network

LSTM Long Short-Term Memory

GWDG Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

1 Introduction

High-Performance Computing (HPC) systems underpin scientific and industrial workloads ranging from climate modelling to the training of large Artificial Intelligence (AI) models, and their availability directly determines the throughput of the research they serve. Yet these systems fail regularly: surveys report that half of all HPC sites experience failures every month, at a cost of \$100,000 to \$1,000,000 per day of downtime [Hyp20], and a nine-year study at Los Alamos National Laboratory documented up to 700 failures per year on a single cluster [SG06]. In today’s accelerated systems, the component that fails most often is the Graphics Processing Unit (GPU). When Meta trained Llama 3 (405B) on 16,384 H100 GPUs, the job was interrupted by 419 unexpected failures in 54 days, one failure every three hours; 58.7% of these interruptions were GPU-related, with 270 GPU and GPU-memory failures against only two Central Processing Unit (CPU) failures [Dub+24]. Because such jobs are tightly coupled, one faulty GPU can abort a computation running on thousands of others.

Three maintenance strategies can respond to this problem. **Reactive maintenance** repairs components after they break and therefore pays the full cost of every failure. **Preventive maintenance** services components on a fixed schedule regardless of their condition, which wastes resources and still misses spontaneous failures. **Predictive maintenance** instead uses continuously collected operational data and Machine Learning (ML) models to anticipate failures and to intervene just in time, for example by draining a node before the scheduler places a large job on it. This report explores the third strategy for GPUs and asks a single question: *can GPU failures be predicted from node telemetry before they happen?*

We address this question on the public telemetry of the Summit supercomputer at the Oak Ridge Leadership Computing Facility (OLCF) [Shi+21; Lu+26], one of the few publicly available datasets that combines fine-grained per-GPU power and temperature measurements with timestamped GPU error logs. Following the problem formulation of Liu et al. [Liu+23], we treat failure prediction as the binary classification of time-series windows and evaluate models with Precision@K, which reflects the limited number of alerts that operators can act upon.

Contributions

- A characterisation of the ClusterWise release of the Summit dataset: its collection process, features, failure landscape across 13 GPU error types, failure co-occurrence, monthly variability, and per-type failure signatures.
- A complete preprocessing pipeline that turns raw per-node telemetry streams into supervised instances through look-ahead windowing, class-imbalance handling and model-specific tensor shaping.
- A comparison of four supervised models, Gradient-Boosted Decision Trees (GBDT), Multi-Layer Perceptron (MLP), 1D-Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM), evaluated with Precision@K, and a parallel ensemble following Liu et al. [Liu+23].
- An extended evaluation restricted to hardware failures, with per-GPU and per-failure-family models evaluated across months at the natural failure rate.

The remainder is structured as follows. Section 2 covers background concepts and related work. Section 3 introduces the dataset and Section 4 analyses it. Section 5 describes the preprocessing pipeline and the models, Section 6 presents the results, including an extended evaluation on hardware failures, and Section 7 discusses the findings. Section 8 concludes and Section 9 outlines future work.

2 Background

2.1 Types of Maintenance

The three maintenance strategies introduced in Section 1 differ in when an intervention happens and what information triggers it. Figure 1 illustrates them on a common timeline. Reactive maintenance waits until the breakdown occurs, preventive maintenance intervenes at fixed intervals whether or not it is necessary, and predictive maintenance raises a warning from monitoring data shortly before a failure is expected, so that the intervention happens just in time. Predictive maintenance is the only strategy that both avoids unplanned downtime and avoids unnecessary service, but it depends on the availability of informative monitoring data and on models that can extract early warning signs from it.

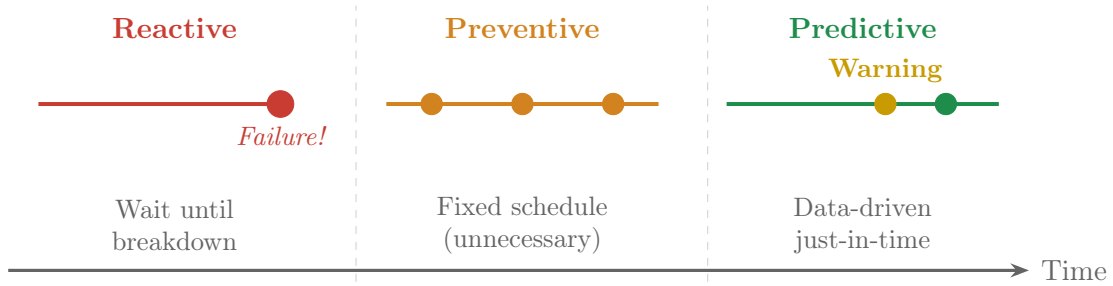


Figure 1: The three maintenance strategies: reactive, preventive, and predictive.

2.2 GPU Failures and XID Errors

GPU failures have three broad origins. **Hardware faults** include bit flips in the on-package High Bandwidth Memory (HBM) caused by radiation, charge leakage or wear, as well as devices that fall off the PCIe bus and errors on the NVLink interconnect. **Environmental stress** covers overheating, cooling problems and power fluctuations. **Software and driver errors** range from driver crashes to applications that misbehave, for example by accessing invalid memory. NVIDIA drivers report all of these events as NVIDIA Error Identifier (XID) errors: numbered codes that the GPU firmware writes to the kernel log at the moment an error occurs, such as XID 48 for a Double-Bit Error (DBE) in GPU memory or XID 79 for a GPU that has fallen off the bus; Appendix A explains all codes in the dataset. Shin et al. [Shi+21] split these codes into errors that can be associated with user applications and errors that are driver- or hardware-related, a distinction that matters for maintenance: only the latter call for an intervention on the hardware.

2.3 Related Work

Two works form the basis of this report. **Shin et al.** [Shi+21] provide the first system-wide analysis of power, energy and thermal dynamics of Summit, based on one year of per-node telemetry sampled at 1 Hz from all 4,626 nodes, together with job scheduler logs, facility data and GPU error logs. Their reliability study shows that the vast majority of GPU errors are associated with user applications, that a few nodes account for a disproportionate share of hardware errors, and that high temperature, unlike on Summit’s predecessor Titan [Ost+20], is not a significant direct cause of failures. They analyse failures but do not predict them.

Liu et al. [Liu+23] study GPU failure prediction on a four-month production dataset of about 350 million entries from ByteDance’s deep learning clusters. They define the task used in this report: a time-series instance consists of an observation window of l steps, and its label is positive if the GPU fails within the following prediction window of length p . They find that classic models (GBDT, MLP, LSTM and 1D-CNN) are both inaccurate and unstable over time, with the 1D-CNN reaching the best Precision@K of 46.3%. They improve this with a *parallel* ensemble, which flags only instances that all member models rank in their top- K , and a *cascade* ensemble, as well as with a sliding training scheme that periodically retrains the model on recent data, reaching a precision of 85.4%.

3 The Dataset

3.1 Public Datasets for GPU Failure Prediction

Few public datasets combine GPU telemetry with failure labels. The Titan GPU failure data [Ost+20] records failure timestamps and spatial topology for a large GPU fleet but no dense telemetry. Telemetry from the Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG) was analysed by Bidollahkhani et al. [BNK26]: they found out that detectors based on GPU telemetry alone gave practically no advance warning for this dataset. We therefore use **ClusterWise** [Lu+26], a curated benchmark release of the Summit telemetry, which offers per-component time series together with failure labels.

3.2 The Summit Supercomputer

Summit is a 200 PF pre-exascale system at the OLCF in the United States and was ranked first on the Top500 list in 2018 [Shi+21]. It consists of 4,626 IBM AC922 nodes arranged in 257 water-cooled cabinets of 18 nodes each. Every node holds two IBM Power9 CPUs and six NVIDIA V100 GPUs with 16 GB of HBM, 27,756 GPUs in total, and draws its power through two power supplies. Figure 2 shows the node layout: each CPU is connected to three GPUs, so that GPUs 0–2 belong to the first socket and GPUs 3–5 to the second, and the hostname of a node encodes its cabinet and its position in the rack.

3.3 Data Collection

The dataset aligns three data streams. **Telemetry** is pushed by every node at 1 Hz: the Baseboard Management Controllers (BMCs) read the power and temperature of each component out of band, that is, without running any software on the compute node

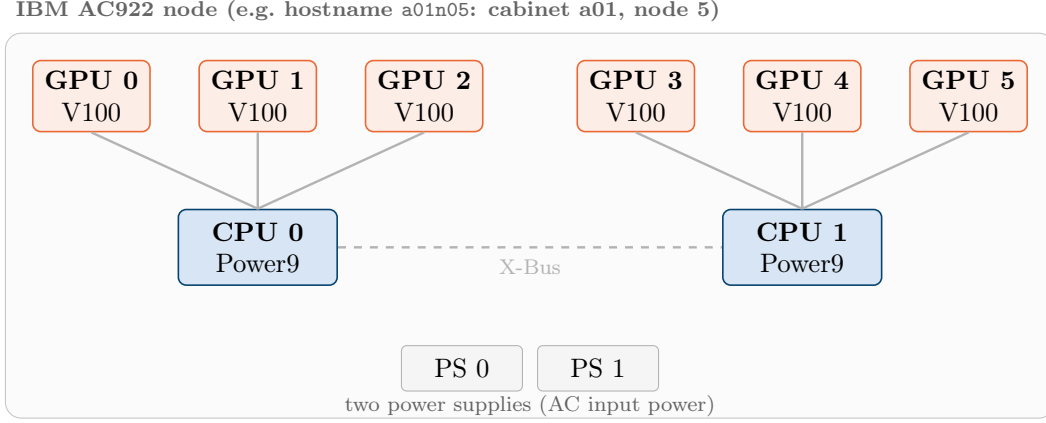


Figure 2: Layout of a Summit compute node: two Power9 CPUs, each connected to three NVIDIA V100 GPUs, and two power supplies.

itself, and forward roughly 460,000 metrics per second across the cluster to a central collection point with an average delay of about four seconds [Shi+21]. **Job scheduler logs** record which job ran on which nodes and when. **GPU error logs** contain the XID codes recorded by the GPU firmware at the moment of failure, together with the affected node and GPU. Combining the three streams yields telemetry that is annotated with job context and failure events.

3.4 Time Span and Availability

The raw internal telemetry covers January 2020 to May 2022 at 1 Hz and occupies about 16 TB. Only part of it is public: the OLCF release [Shi+22] contains five month-long segments (January and August 2020, February and August 2021, and January 2022) aggregated to 10-second and 1-minute resolution, and ClusterWise curates the same five segments into a 221 GB benchmark at 10-second resolution. All analytics, preprocessing and modelling in this work were run on the HPC systems of the GWDG.

3.5 Features

Each record describes one node at one timestamp through 28 telemetry features and additional context, listed in Table 1. Twelve features describe the core and memory temperature of each GPU, six its power consumption, eight the power and minimum, mean and maximum core temperature of each CPU, and two the AC input power of both power supplies. The records further carry the job allocation, the node location encoded in the hostname, and a failure label together with its XID code.

3.6 The Failure Landscape

Across the five available months, Summit logged 260,039 GPU errors spanning 13 XID types. Figure 3 shows their distribution on a logarithmic scale, split into application-associated and driver- or hardware-related errors. Three application-associated types, graphics engine exceptions, memory page faults and stopped processing, account for 258,586 errors. The ten driver- and hardware-related types together account for only

Table 1: Features of each record.

Group	Features
GPU thermal (12)	Core and memory (HBM) temperature per GPU
GPU power (6)	DC power consumption per GPU
CPU (8)	Power and minimum/mean/maximum core temperature per CPU
Node power (2)	AC input power of both power supplies
Job context	Allocation ID, job start and end time, node state
Location	Hostname encoding cabinet and position in the rack
Failure label	Normal (0) or failure (1), plus the XID error code

1,453 errors, less than 0.6% of the total, with page retirement failures and events, pre-emptive cleanups, DBEs and NVLink errors as the most frequent. Hardware failures, which are the actual target of maintenance, are therefore rare events hidden in a large volume of application errors.

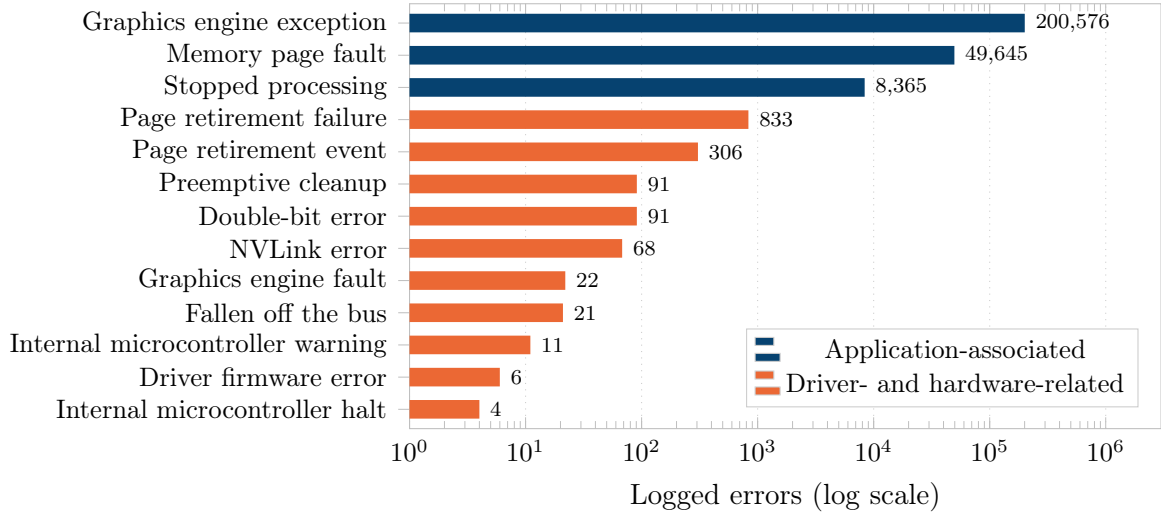


Figure 3: The 13 GPU error types logged across the five months, split into application-associated and driver- or hardware-related errors (260,039 errors in total).

4 Data Analytics

4.1 What a Failure Looks Like

Figure 4 shows the GPU core temperature, memory temperature and power around six example failures of different types, each standardised within its window, with the labelled failure period highlighted in red. The signatures differ strongly between types. Some leave an unmistakable trace: before a GPU stops processing (XID 43) all three signals collapse, and a GPU that falls off the bus (XID 79) shows abrupt spikes followed by flat, idle readings. Others, such as the NVLink error (XID 74), occur in the middle of a regular periodic workload and look almost identical to normal operation. Memory errors such as DBEs (XID 48) appear within noisy, rapidly fluctuating signals. This heterogeneity

rules out simple thresholds on individual sensors and motivates learned models that can combine weak patterns across features and time.

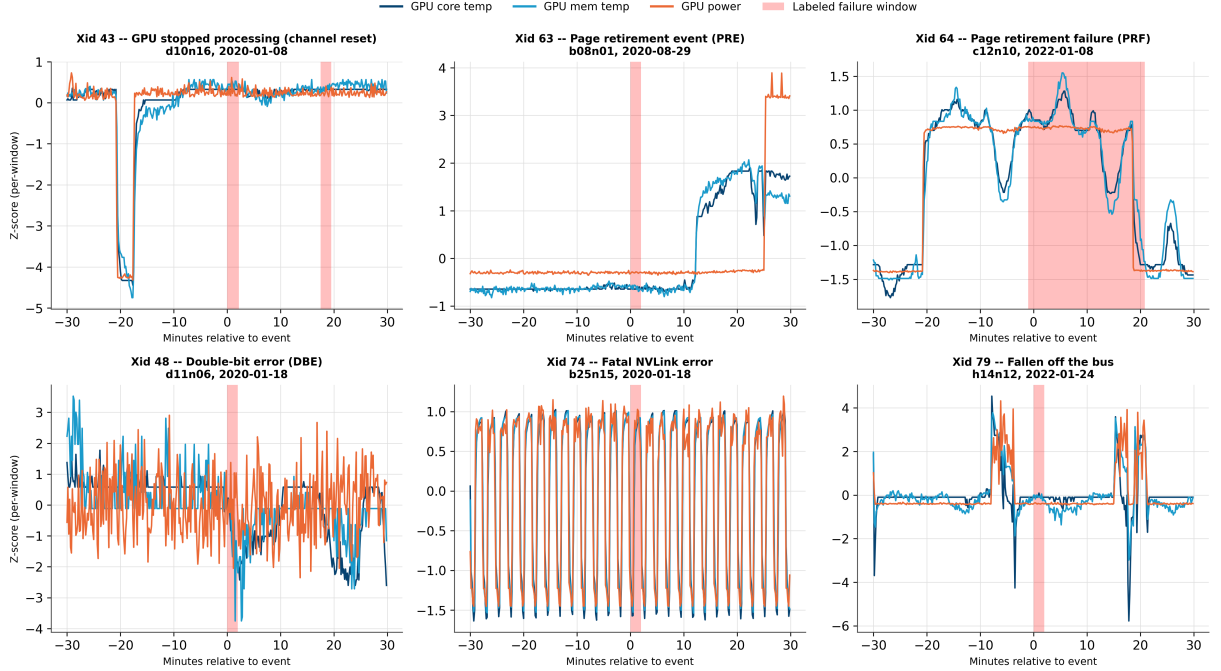


Figure 4: GPU core temperature, memory temperature and power around six example failures of different XID types, standardised per window. The red bands mark the timestamps labelled as failure.

4.2 Failure Co-Occurrence

To understand whether failure types appear together, we counted the failures of each type per node and computed the Pearson correlation between these per-node counts for every pair of types, keeping only coefficients that are significant at 0.05 after a Bonferroni correction. Figure 5 shows the resulting matrix. Most pairs are independent, with near-zero correlation. A few, however, are almost inseparable: driver error handling exceptions and internal microcontroller warnings (0.96), and DBEs and preemptive cleanups (0.98), which also correlate with page retirement events (0.66 and 0.65). Among application errors, stopped processing co-occurs with graphics engine exceptions (0.68). Such pairs suggest that a soft error can act as an early indicator of a more severe one, for instance a microcontroller warning preceding a fatal driver error.

4.3 Monthly Variability

Failures are present on every available day, but their volume is far from stable. Figure 6 shows the number of failure log entries per month together with the number of affected hosts. February 2021 stands out with 179,259 entries across 2,849 hosts, roughly six to thirteen times the volume of any other month, while the number of affected hosts in that month is not higher than elsewhere. Such bursts, driven by a limited number of workloads or nodes, mean that the class balance, and potentially the failure patterns themselves,

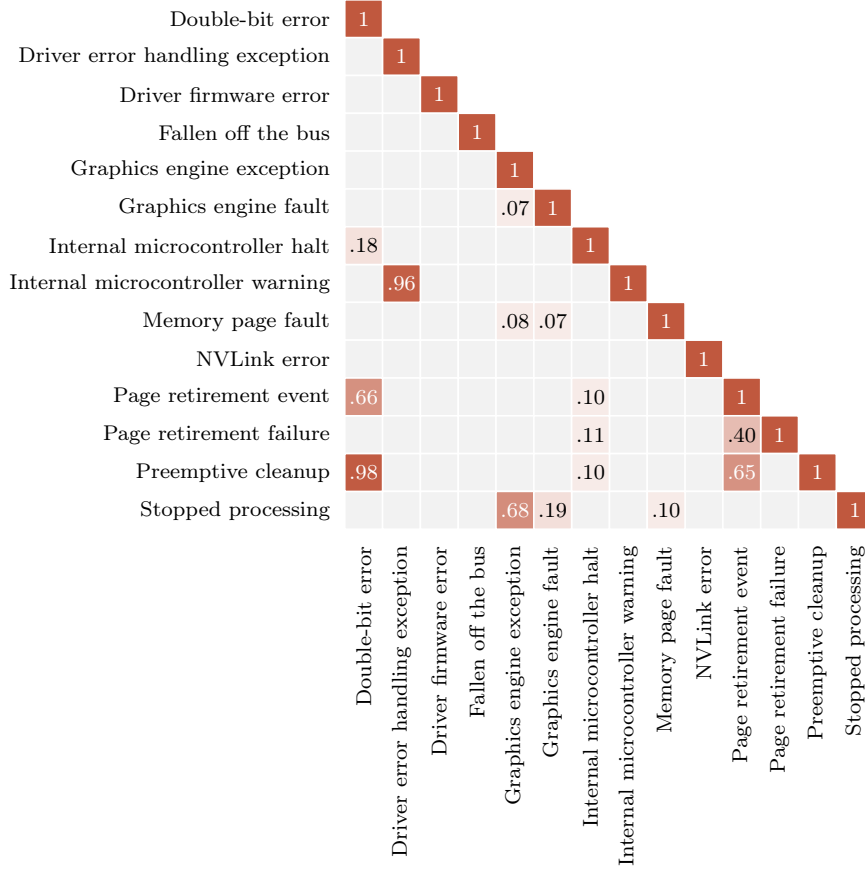


Figure 5: GPU failure co-occurrence on Summit.

shift from month to month, a phenomenon Liu et al. [Liu+23] describe as failure pattern drift.

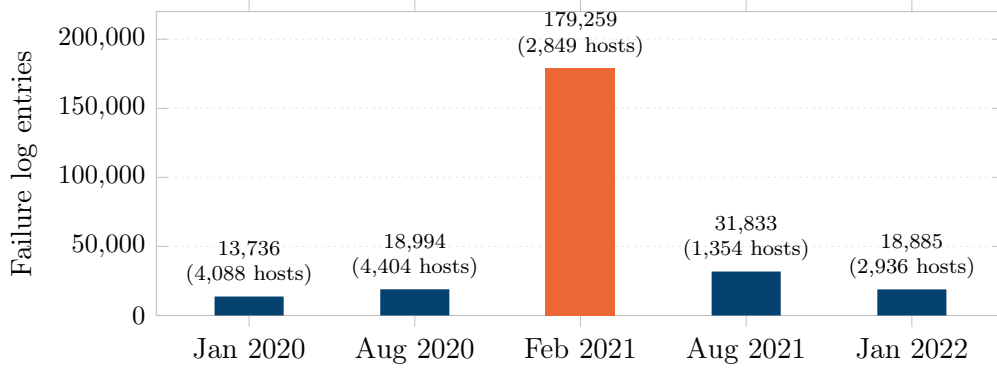


Figure 6: Failure log entries and affected hosts per month.

5 Methodology

Figure 7 gives an overview of the pipeline. The raw per-node streams are cleaned, segmented into observation and prediction windows, rebalanced, scaled and reshaped into

one view per model. The four models then score every test window, and the ranked scores are evaluated with Precision@K. The following subsections describe each step.

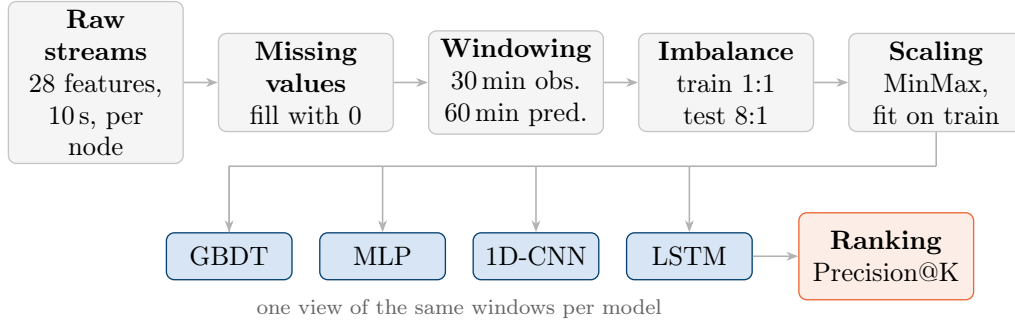


Figure 7: Overview of the preprocessing and modelling pipeline.

5.1 Handling Missing Values

Sensor readings are missing when a GPU is idle or offline and does not report a value. The rate is small but not zero, about 0.5% of the rows for most features. We fill missing values with zero, which is simple, matches the convention of the dataset’s original publication, and lets the models recognise non-reporting devices as a distinct state.

5.2 From Raw Streams to Supervised Instances

Each host’s stream is converted into time-series instances by sliding a pair of windows over it: a **30-minute observation window**, which is the model input, followed by a **60-minute prediction window**, which determines the label. The pair advances with a stride of 60 seconds. Figure 8 illustrates the three possible cases. An observation window must be failure-free, and the instance is labelled positive if and only if at least five rows of its prediction window are labelled as failure (window B), and negative otherwise (window A). If a failure falls into the observation window itself (window C), the instance is discarded rather than relabelled, since the model should predict failures before they begin and not recognise ongoing ones.

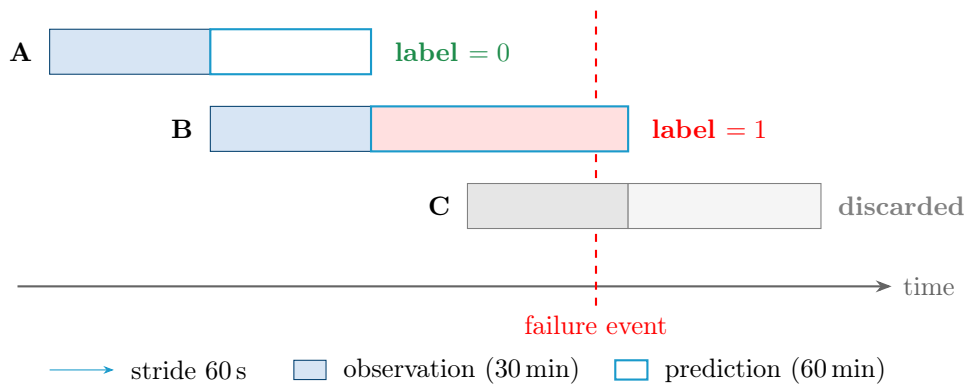


Figure 8: Look-ahead windowing and labelling. A failure in the prediction window yields a positive instance (B); a failure in the observation window discards the instance (C).

5.3 Handling Class Imbalance

At the window level, the classes are severely imbalanced, with roughly one positive for every 2,200 negatives. We keep every positive window and randomly subsample the negatives to a target ratio instead of using all of them. The training set is balanced to a negative-to-positive ratio of 1:1, which gives the models a balanced learning signal, while the test set keeps a ratio of 8:1, which preserves part of the skew at test time and corresponds to a positive rate of 11.1%.

5.4 Scaling and Model-Ready Tensors

All features are scaled to $[0, 1]$ with a MinMax scaler that is fitted on the training split only and then applied to the test split, which avoids leaking statistics of the test data into the model. Each window of L time steps and $F = 28$ features is then reshaped into the input format expected by its model, as listed in Table 2. The same windowed instances thus feed every model, and only their view differs.

Table 2: Input representation of the windows for each model (L time steps, $F = 28$ features).

Model	Input shape	Description
GBDT	$4F = 112$	mean, standard deviation, minimum and maximum per feature
MLP	$L \cdot F$	flattened window
1D-CNN	(F, L)	channel-first, features as channels
LSTM	(L, F)	time-first sequence

5.5 Models

We train the four model families that Liu et al. [Liu+23] use as baselines. They represent increasingly expressive views of the window, from order-free summary statistics to explicit temporal modelling.

GBDT. A gradient-boosted decision tree ensemble, implemented with the histogram-based gradient boosting classifier of scikit-learn, operates on the 112 aggregated window features. It uses up to 800 trees with a learning rate of 0.05, at most 63 leaves per tree, and L_2 regularisation. Trees handle heterogeneous feature scales and interactions well, but the aggregation discards the order of the observations.

MLP. A multi-layer perceptron receives the flattened raw window of $L \cdot F$ values. Two hidden layers with 256 and 64 units, each followed by a ReLU activation and dropout of 0.2, feed a single output unit. The network is trained with a binary cross-entropy loss on logits, with the positive class re-weighted according to the batch composition.

1D-CNN. A one-dimensional convolutional network treats the 28 features as channels and convolves over time. Three convolutional blocks with 64, 64 and 128 channels and kernel sizes 5, 5 and 3, each followed by batch normalisation and ReLU, are followed by one max-pooling layer, an adaptive average pooling layer and a fully connected head. The convolutions learn local, short-range temporal patterns such as sudden drops or spikes.

LSTM. A single-layer LSTM with a hidden size of 64 reads the window step by step. Its final hidden state is passed through Linear(64 $\rightarrow$ 32), ReLU, dropout of 0.2 and Linear(32 $\rightarrow$ 1). The recurrent state is meant to capture long-range dependencies across the full 30-minute window.

5.6 Evaluation Metric: Precision@K

In operation, a failure predictor does not issue a binary verdict per window but a ranking: operators can only act on a finite number of alerts, for example by draining the most suspicious nodes. Precision@K models this alert budget directly. All test windows are ranked by their predicted score, the top K are flagged, and the metric is the share of true failures among them:

$$\text{Precision@}K = \frac{|\{\text{true failures among the top-}K \text{ ranked windows}\}|}{K}. \quad (1)$$

We report K as a fraction of all test windows, at 5%, 10% and 20%. With the test ratio of 8:1, a random ranking achieves a Precision@K of $1/9 \approx 0.11$ at every K , which serves as the reference point for all results.

5.7 Parallel Ensemble

Liu et al. [Liu+23] observe that models with different architectures make different errors but tend to agree on obvious failures. Their parallel ensemble, shown in Figure 9, feeds each test window to several models, lets each model select its own top- K windows, and flags only the windows in the intersection of these sets. The ensemble trades recall for precision, which matches the goal of an alerting system in which false alarms are costly. We combine the two strongest individual models, the GBDT and the 1D-CNN.

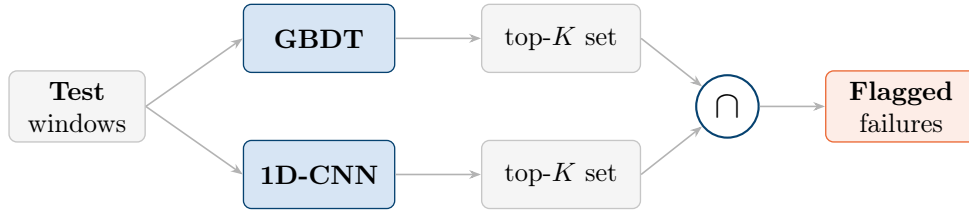


Figure 9: Parallel ensemble after Liu et al. [Liu+23]: a window is flagged only if both models independently rank it within their top- K .

6 Results

All four models are trained on six pooled days and tested on two held-out days under the same preprocessing, sampling and scaling.

6.1 Model Comparison

Figure 10 shows the Precision@K of the four models at the three alert budgets. All models clearly exceed the random reference of 0.11. The 1D-CNN is the strongest model

at every budget, with a Precision@K of 0.61 at $K = 5\%$ and $K = 10\%$, followed by the GBDT with 0.58 to 0.60. The two models also behave differently as the budget grows: the precision of the 1D-CNN is almost flat, which means its highest-ranked windows are not more reliable than those further down, whereas the GBDT gains slightly with larger K . The MLP and the LSTM lag behind with 0.50 to 0.57, and the LSTM is the weakest model at the tightest budget of 5%, where precision matters most. Both improve towards $K = 20\%$, but never reach the two leading models.

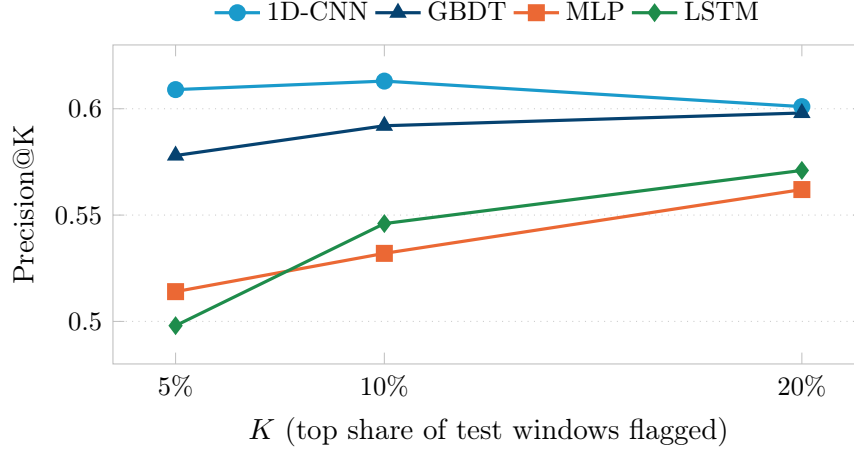


Figure 10: Precision@K of the four models on the two test days at a test ratio of 8:1 (random ranking: 0.11).

6.2 Ensembling GBDT and 1D-CNN

Figure 11 compares the parallel ensemble of the GBDT and the 1D-CNN with its two members. The ensemble improves on the better member at every budget, reaching a Precision@K of 0.615, 0.623 and 0.608 at 5%, 10% and 20%, respectively. The gain over the 1D-CNN alone is small, between 0.6 and 1.0 percentage points, but consistent: windows that both a tree ensemble on summary statistics and a convolutional network on the raw sequence rank highly are more likely to precede a failure than windows selected by either model alone.

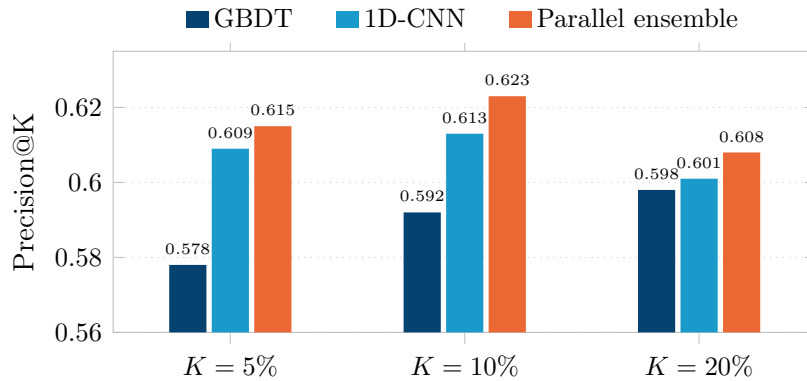


Figure 11: Precision@K of the parallel ensemble compared with its two members.

6.3 Comparison with Liu et al.

Liu et al. [Liu+23] report a Precision@2% of 46.3% for their best single model, the 1D-CNN, 58.2% for the parallel ensemble, and up to 85.4% with sliding training. The two studies share the task formulation, the model families, the test ratio of 8:1 and the metric, but differ in the system and workload (deep learning at ByteDance), the window lengths (a 3-hour observation and a one-day prediction window), the training period (15 days) and the value of K , so the numbers cannot be compared on equal terms. Two qualitative findings nevertheless carry over: in both studies the 1D-CNN is the strongest individual model and the LSTM one of the weakest, and in both the parallel ensemble improves over its best member, although the gain on Summit is much smaller than the 11.9 percentage points reported by Liu et al.

6.4 Hardware Failures Across Months

The setup above pools all failure types, tests on two days and evaluates at an artificial ratio of 8:1. To test how far its results carry over to the maintenance use case, a second set of experiments removes these simplifications.

Setup. Only the ten driver- and hardware-related XID types count as failures. Following Liu et al. [Liu+23], the task is to predict from a 3-hour observation window whether a failure occurs within the next 24 hours. To make the full dataset tractable, the 10-second telemetry is aggregated into 10-minute steps holding the mean, standard deviation, minimum and maximum of each signal. A failure episode is the first failure on a node or GPU after at least 24 hours without one, and windows within 24 hours after a failure are excluded. The models are evaluated with leave-one-month-out cross-validation: they are trained on four months and tested on the fifth, and the predictions of all five folds are pooled. Every node or GPU is scored every hour at the natural failure rate of about one positive per 2,500 node hours. Besides Liu’s Precision@2% on a test set resampled to 8:1, we report the area under the ROC curve (ROC-AUC) and the share of failure episodes that receive at least one alarm in their 24-hour lead-up when the top 1% of all scored hours raise an alarm, compared with the share that the same number of random alarms would catch.

In addition to a node-level GBDT, we train a per-GPU GBDT in which each GPU is an instance, described by its own temperature and power statistics, its deviation from the other five GPUs of the node, node-level context, its slot, and its own failure history (the time since its last failure and its number of failures in the past 30 days). Since a 24-hour horizon produces about 140 nearly identical positive windows per failure, training keeps at most 12 windows per failure episode, which prevents the models from memorising individual failures. Separate per-GPU models are trained for all hardware failures, for memory errors (XID 45, 48, 63, 64) and for interconnect errors (XID 74, 79). Table 3 summarises the results.

These results change the picture in three ways. First, the node-level models barely beat chance across months: their ROC-AUC is 0.50 to 0.53, and at an alarm budget of 1% they catch fewer failures than random alarms would, even though their Precision@2% of 0.39 to 0.49 is far above the random 0.11. The resampled protocol therefore overstates the operational value of a model. Second, per-GPU modelling helps, raising the ROC-AUC to 0.59 and the share of alarmed failures to 23.8%, but the gain disappears without the failure history: on telemetry alone, every model falls to or below chance. The predictive

Table 3: Leave-one-month-out results for hardware failures with a 24-hour horizon, pooled over all five months. The last column gives the share of failures that receive an alarm when 1% of all scored hours raise one, with the share expected from random alarms in parentheses.

Model	Target	Failures	ROC-AUC	P@2%	Alarmed (random)
Node GBDT, all windows	all hardware	285	0.50	0.39	13.7% (19.2%)
Node GBDT, capped	all hardware	285	0.53	0.49	15.4% (19.2%)
GPU GBDT	all hardware	307	0.59	0.72	23.8% (19.1%)
GPU GBDT, no history	all hardware	307	0.53	0.30	13.7% (19.1%)
GPU GBDT	memory	220	0.51	0.49	13.6% (19.2%)
GPU GBDT, no history	memory	220	0.50	0.24	10.9% (19.2%)
GPU GBDT	interconnect	66	0.74	0.99	43.9% (18.0%)

signal is thus mostly that a GPU has failed before, not a precursor in its telemetry. Third, the failure families behave differently. Memory errors cannot be predicted, whereas interconnect errors can: the per-GPU model reaches a ROC-AUC of 0.74 and raises an alarm for 43.9% of them, compared with 18.0% for random alarms, with a median lead time of 23 hours. This result rests on only 66 failure episodes, and given the role of failure history it likely reflects GPUs with recurring link problems.

7 Discussion

7.1 Key Observations

- **Local temporal patterns matter most.** In the setup of the presentation, the 1D-CNN outperforms the LSTM, consistent with the abrupt failure signatures in Section 4.1, while the parallel ensemble adds at most one point.
- **The evaluation protocol decides the verdict.** A Precision@K of about 0.6 at a ratio of 8:1 over two days turns into near-chance performance for hardware failures when models are evaluated across months at the natural failure rate (Section 6.4). Resampled test sets and short test periods are convenient but can hide that a model is not useful in operation.
- **Failure history dominates telemetry.** Where hardware failures are predictable at all, the signal comes from a GPU’s past failures, echoing the repeat offenders found by Shin et al. [Shi+21].
- **Failure types are heterogeneous.** Signatures range from abrupt flatlines to patterns hidden in normal operation, and predictability differs between failure families. Bidollahkhani et al. [BNK26] observe the same split between drift-dominated and detachment-class instabilities and find that value-centric detection on numeric telemetry alone misses the latter.

7.2 Limitations

The main experiments pool all 13 XID types, which are dominated by application errors, and use a short training and test period and an artificial test ratio; Section 6.4 shows how strongly this affects the conclusions. The extended evaluation has limitations of its own. Only five non-contiguous months are available, the interconnect result rests on 66 failure episodes, and the aggregation to 10-minute steps may remove short precursors that the 1 Hz telemetry would retain. Working at 1 Hz was not feasible, however: the full telemetry occupies about 16 TB, and even the 10-second release amounts to about 500 GB. Finally, overlapping windows derived from the same failure are highly correlated, which reduces the effective number of independent failure events in every split.

8 Conclusion

This report investigated whether GPU failures in an HPC system can be predicted from node telemetry before they occur, using the public ClusterWise release of the Summit supercomputer. The dataset analysis showed that GPU errors are dominated by application-associated types, that hardware failures account for less than 0.6% of all logged errors, that failure volumes vary strongly between months, and that failure signatures range from unmistakable to indistinguishable from normal operation.

In our setup, which pools all failure types and tests on two days at a ratio of 8:1, all four models clearly outperformed a random ranking; the 1D-CNN reached a Precision@K of about 0.61 and a parallel ensemble with the GBDT 0.62. The extended evaluation, restricted to hardware failures and run across months at the natural failure rate, draws a more sober picture: node-level models barely beat chance, and per-GPU models improve mainly through failure history rather than telemetry. The one clear success is the prediction of interconnect errors, where 44% of failures receive an alarm about a day in advance. Predicting GPU hardware failures from coarse numeric telemetry therefore remains hard, and how a model is evaluated matters as much as which model is chosen.

9 Future Work

Richer signals. Since coarse telemetry carries little predictive signal, future models should use the full 1 Hz resolution, job context such as the running application, and observability signals beyond numeric telemetry, such as missing or stalled sensor readings, together with feature attributions that reveal what drives the predictions for each failure family.

Robustness and generalisation. The sliding training scheme of Liu et al. [Liu+23] should be evaluated across months to cope with failure pattern drift, and the pipeline should be applied to other HPC systems, in particular the GPU clusters of the GWDG, to test whether the findings hold beyond Summit.

References

- [BNK26] Michael Bidollahkhani, Freja Nordsiek, and Julian M. Kunkel. *When GPUs Fail Quietly: Observability-Aware Early Warning Beyond Numeric Telemetry*. 2026. arXiv: 2603.28781 [cs.DC]. URL: <https://arxiv.org/abs/2603.28781>.
- [Dub+24] Abhimanyu Dubey et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI].
- [Hyp20] Hyperion Research. *HPC Market Update and Trends*. Tech. rep. Hyperion Research, 2020.
- [Liu+23] Heting Liu et al. “Predicting GPU Failures With High Precision Under Deep Learning Workloads”. In: *Proceedings of the 16th ACM International Conference on Systems and Storage (SYSTOR '23)*. Haifa, Israel, 2023, pp. 124–135. DOI: 10.1145/3579370.3594777.
- [Lu+26] Yiyang Lu et al. *ClusterWise: HPC Timeseries Dataset*. 2026. DOI: 10.57967/hf/9384.
- [NVI] NVIDIA Corporation. *XID Errors*. NVIDIA. URL: <https://docs.nvidia.com/deploy/xid-errors/index.html> (visited on 09/26/2026).
- [Ost+20] George Ostrouchov et al. “GPU Lifetimes on Titan Supercomputer: Survival Analysis and Reliability”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20)*. 2020.
- [SG06] Bianca Schroeder and Garth A. Gibson. “A Large-Scale Study of Failures in High-Performance Computing Systems”. In: *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*. Philadelphia, PA, USA, 2006, pp. 249–258.
- [Shi+21] Woong Shin et al. “Revealing Power, Energy and Thermal Dynamics of a 200PF Pre-Exascale Supercomputer”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*. 2021. DOI: 10.1145/3458817.3476188.
- [Shi+22] Woong Shin et al. *Long Term Per-Component Power and Thermal Measurements of the OLCF Summit System*. 2022. DOI: 10.13139/OLCF/1861393.

A NVIDIA XID Errors

NVIDIA drivers report GPU errors as XID messages in the kernel log, for example `NVRM: Xid (PCI:0035:05:00): 31, pid=2629696, ...`, where the number after the PCI address is the error code [NVI]. An XID records that an event occurred and where, but not necessarily its root cause: the same code can be triggered by a faulty application, the driver or the hardware. Table 4 explains the 13 codes that occur in the five months of the Summit dataset, grouped into application-associated and driver- or hardware-related errors, together with the number of occurrences in the dataset (Section 3.6).

Table 4: The 13 NVIDIA XID error codes that occur in the Summit dataset, with their meaning according to the NVIDIA documentation [NVI] and their number of occurrences across the five months.

XID	Name	Meaning and typical cause	Count
<i>Application-associated</i>			
13	Graphics engine exception	A kernel running on the GPU raised an exception, typically an out-of-bounds memory access or an illegal instruction in user code; rarely a driver or hardware problem.	200,576
31	Memory page fault	The GPU memory management unit caught an access to an invalid or unmapped address, most often an illegal memory access by the application.	49,645
43	Stopped processing	A GPU channel was reset and stopped executing work, usually because the application running on it faulted.	8,365
<i>Driver- and hardware-related</i>			
64	Page retirement failure	The driver failed to record the retirement of a GPU memory page after ECC errors, so the faulty page may be used again; points to failing GPU memory.	833
63	Page retirement event	A GPU memory page was retired after repeated or uncorrectable ECC errors, and the retirement was recorded; takes effect after the next GPU reset.	306
45	Preemptive cleanup	The driver cleaned up a GPU context after a previous error or because the process was terminated; usually follows another, more specific error.	91
48	Double-bit error	An uncorrectable (double-bit) ECC error in GPU memory; the affected application is terminated and the GPU needs a reset.	91
74	NVLink error	An error on the NVLink interconnect between GPUs or between GPU and CPU, caused by a faulty link, connector or GPU.	68
44	Graphics engine fault	A fault in the graphics engine during a context switch, typically caused by driver or hardware issues.	22
79	Fallen off the bus	The GPU is no longer reachable over the PCIe bus; caused by hardware, power or thermal problems, and requires a node reboot.	21
61	Internal microcontroller warning	The on-GPU microcontroller hit a breakpoint or reported a warning, indicating a firmware or hardware anomaly.	11
38	Driver firmware error	The driver detected an error in the GPU firmware.	6
62	Internal microcontroller halt	The on-GPU microcontroller halted, a more severe form of XID 61 that requires a GPU reset.	4