From Noisy Telemetry to Actionable Warnings: GPU Failure Prediction in Industrial Clusters

Yongqian Sun[†], Run Zhu[†], Wenwei Gu[†*], Mengyao Li[†], Shenglin Zhang[†], Guanjin Wang[†]
Yang Zhang[‡], Xin Wu[‡], Linlin Han[‡], Feng Wang[‡], Xiaozhou Liu[‡], Yu Zhang[‡]
[†]*Nankai University* [‡]*ByteDance*

***Abstract*—GPU clusters are critical infrastructure for AI services, but accurate and actionable GPU failure prediction remains a problem in production settings. We study ticket-linked telemetry from a ByteDance GPU cluster and identify three obstacles: workload-confounded telemetry, heterogeneous fault precursors, and the gap between window-level predictions and actionable alerts. These findings motivate *Falcon*, a fault-specific warning framework combining missingness-aware temporal and peer-relative features, fault-specific learner selection, and an event policy based on thresholding, persistence, and cooldown. On the test set, *Falcon* achieves the highest F1 among four baselines and reaches 70.6% F1 on the best-performing fault type. Detected cases provide median lead times of 17.34–35.57 hours. We further report a production deployment, where *Falcon* is calibrated toward high-precision alerts to reflect false-positive costs. Together, these results show that fault-specific modeling improves early warning from noisy production GPU telemetry.**

***Index Terms*—GPU cluster, failure prediction, event-level evaluation, fault-specific modeling**

I. Introduction

As modern software systems are increasingly driven by AI services, large language models, and agent workflows, their reliability increasingly depends on this AI infrastructure. GPU faults therefore surface as software-level failures, including service interruptions, service-level objective (SLO) violations, failed training jobs, increased inference latency, and broken workflow execution. At this scale, a device problem rarely stays local. One unhealthy card can stall synchronization, trigger job recovery, occupy expensive capacity, or delay diagnosis across several infrastructure layers. Large GPU clusters now carry workloads whose useful progress depends on thousands of tightly coupled accelerators, network links, power and cooling systems, drivers, and distributed runtimes [1]–[3]. GPU failure prediction is therefore a foundational problem for AI-native software reliability: risky devices must be identified early enough to inspect, drain, or replace them before the next ticket interrupts a workload. Existing systems cover adjacent parts of this reliability problem. Recent GPU degradation forecasting studies show that multivariate telemetry can contain pre-failure signals and use sliding-window health forecasting over GPU-cluster metrics [4]–[6], but they primarily produce window-level health or failure scores rather than stable device-level warning events tied to fault families, tickets, and intervention horizons. Diagnosis and mitigation systems such as Astral, Holmes, Minder, GREYHOUND, and Oobleck operate around runtime symptoms, faulty-machine localization, fail-slow mitigation, or resilient training [2], [7]–[10]. They motivate a complementary operations question: before a maintenance ticket, can ordinary monitoring telemetry become an early, precise, and actionable fault-specific warning?

The first obstacle is that production telemetry is not a direct fault signal. Temperature, power, utilization, clocks, and NVLink or PCIe traffic are indirect health indicators and are strongly confounded by workload. Optional counters may disappear for days, samples are not perfectly aligned, and healthy GPUs move among operating modes as jobs change. The model must distinguish workload-driven healthy variation from pre-fault health changes under missing and misaligned telemetry.

Fault mechanisms are also heterogeneous. Remapping failures may be preceded by remapping-state changes, ECC increments, or severe XID events; uncorrected-memory errors expose bursts of ECC or XID evidence; thermal faults show sustained temperature, throttling, or power-brake behavior; and power anomalies surface through abnormal power draw or braking states. Because signal density, label volume, and lead time differ by fault, one architecture or global threshold is unlikely to serve all fault families.

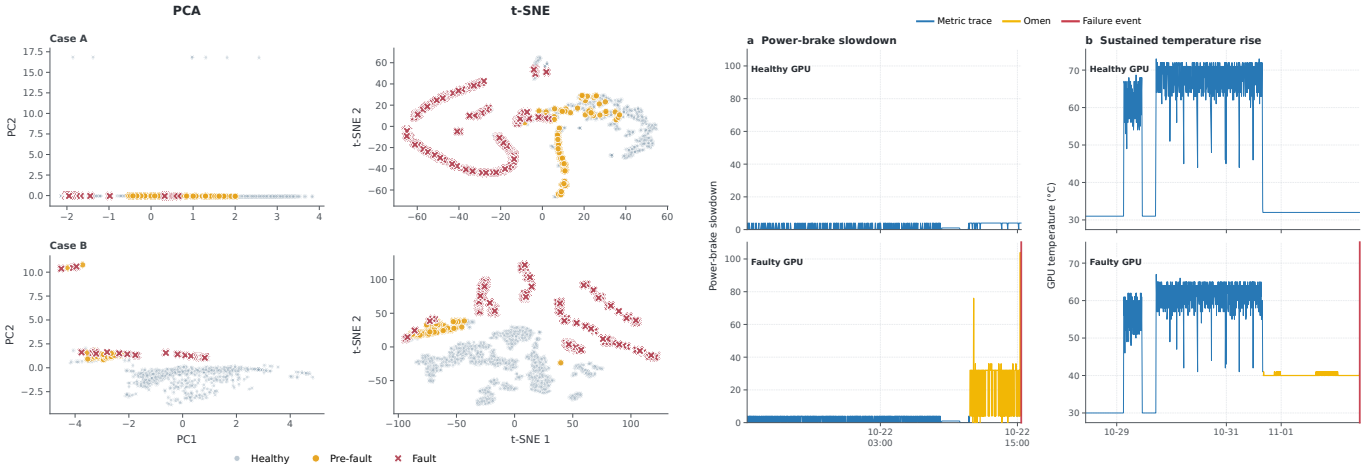
Finally, prediction must become alerts. Operators do not act on every window-level probability; they need stable warning events associated with a device, a fault family, and an intervention horizon. Event formation is therefore part of the prediction problem: isolated spikes should be filtered, repeated warnings suppressed, and precision, recall, and lead time balanced under an operator-attention budget. This also explains why the ByteDance remapping deployment uses a high-precision operating point rather than maximizing a window score alone.

We propose *Falcon*, short for Fault-specific Alerting for Large-scale GPU Clusters from Operations, as an analysis-driven warning framework for this setting. *Falcon* addresses workload-confounded telemetry with missingness-aware temporal and peer-relative features, addresses fault heterogeneity by selecting learners separately for each fault, and addresses alert operationalization through thresholding, N-of-K persistence, and cooldown.

This paper makes four contributions:

- We formulate production GPU failure prediction as a fault-specific, event-level warning problem and analyze

*Wenwei Gu is the corresponding author.



(a) Projection evidence

(b) Trace evidence

Fig. 1. Empirical evidence for GPU fault predictability. PCA/t-SNE projections show that pre-fault states are not cleanly linearly separable from healthy states, while representative traces expose temporal shifts in power-brake slowdown and temperature.

ticket-linked ByteDance telemetry to separate statistical separability from operational predictability.

- We present *Falcon*, which combines missingness-aware temporal features, peer-relative evidence, fault-specific learners, and event formation with thresholding, persistence, and cooldown.
- On four held-out fault types, *Falcon* achieves the best event-level F1 among the implemented baselines while providing 17.34–35.57 hours of median lead time.
- We report a production remapping-failure deployment at ByteDance with 0.800 converged-alert precision under a 2-day confirmation protocol.

II. Empirical Study

We study anonymized telemetry and maintenance tickets from a production ByteDance GPU cluster covering more than 1,000 GPUs. Each ticket records the affected GPU, fault type, occurrence time, and, when available, closing time. The telemetry contains 67 raw fields, including 13 metadata fields and 54 monitoring fields such as temperature, power, clocks, utilization, PCIe/NVLink throughput, power-brake states, XID events, ECC counters, and row-remapping states. The ticket corpus is broader than the prediction tasks: hardware faults, facility or environmental faults, and software-stack faults account for 41.05%, 12.98%, and 45.96% of recorded tickets, respectively.

We do not treat every ticket type as a prediction target. Some classes are not GPU-scoped, some have too few labels, and others lack continuous pre-fault telemetry. We select four targets that are GPU-specific, have sufficient ticket-linked pre-fault telemetry, and expose plausible precursors for maintenance action: GPU Power Unknown Error, GPU Remapping Failure, GPU Temperature High, and GPU Uncorrected Error Over ByteDance Limit. Other ticket types remain in the taxonomy but are excluded from model evaluation because they would require a different incident-classification setup.

We screen the selected targets for pre-fault evidence using KL divergence, two-sample KS tests, low-dimensional projections, and representative traces. The checks give three findings. First, some faults expose observable pre-fault signals. At the 8-hour window, GPU memory temperature has mean and median KL values of 1.705 and 0.880, and GPU temperature has 1.785 and 0.813; neither metric has any job with KL below 10^{-1}. For remapping, power draw, memory temperature, GPU temperature, NVLink/PCIe throughput, and memory utilization also repeatedly pass the KS screen. Second, workload variation prevents clean linear separability: PCA overlaps substantially, while t-SNE and traces reveal local shifts such as power-brake slowdown and sustained temperature rise (Fig. 1). Third, precursor families differ by fault: temperature and power targets depend on dense thermal, power, and throttling channels, whereas remapping and uncorrected-memory targets rely more on sparse XID, ECC, and remapping-state evidence.

These findings motivate three design requirements. The encoder must preserve temporal change and peer context rather than rely only on raw values. Risk learning must be fault-specific instead of sharing one model and threshold across tickets. Finally, separability does not guarantee deployability: window-level risk must be converted into stable alert events with thresholding, persistence, and cooldown before warning quality is evaluated.

III. Methodology

A. Overview and Problem Formulation

Fig. 2 summarizes *Falcon*. The framework maps directly to the three empirical challenges in Section II. First, missingness-aware temporal and peer-relative encoding addresses indirect, incomplete, and workload-confounded telemetry. Second, fault-specific learner selection addresses heterogeneous precursor families and warning horizons. Third, alert-event formation addresses the operational gap between a window-level score and an operator-visible warning. *Falcon* therefore separates

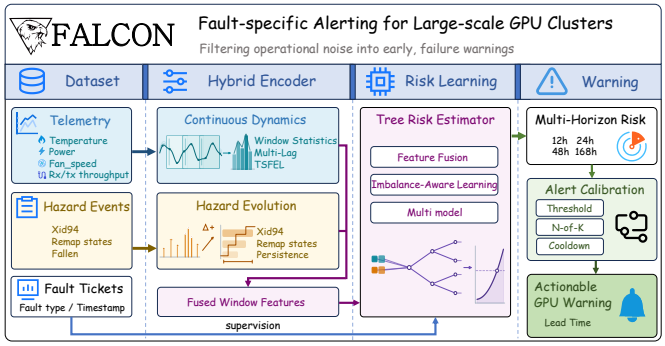


Fig. 2. Fault-specific GPU warning framework. Dense telemetry and sparse hazard events are encoded into fused window features; a selected tree-based learner estimates risk; threshold, N-of-K persistence, and cooldown convert window-level scores into actionable warning events.

four decisions: define the target fault, encode recent telemetry and hazard evolution, learn a fault-specific risk score, and calibrate that score into a warning event.

For a fault type f, let $\mathbf{X}_{g,t-W:t}$ denote the telemetry of GPU g in a window of length W ending at time t. The model estimates

$$p_f(g,t) = P\left(T_{g,f} \in (t, t+H] \mid \mathbf{X}_{g,t-W:t}\right), \qquad (1)$$

where $T_{g,f}$ is the next ticket time for fault f and H is the prediction horizon. The score $p_f(g,t)$ is only a window-level risk estimate. A separate event policy decides whether successive scores become a valid warning within the intervention horizon.

B. Temporal and Peer-Relative Feature Construction

The feature encoder addresses incomplete and workload-confounded telemetry. Production GPU streams are sampled at one-minute intervals but remain partially missing and strongly affected by job phases. We align each GPU stream to the one-minute grid. Missing values in continuous telemetry are filled by linear interpolation before window aggregation. To make this imputation visible to the model, we also retain availability indicators, valid-sample counts, and missing ratios for each window. Temperature, power, clocks, utilization, and PCIe/NVLink throughput are then summarized by lagged statistics, changes, slopes, and persistence summaries. Count-like hazard fields, such as ECC and XID signals, are encoded through non-negative increments and event counts rather than smoothed averages.

Peer-relative features provide a local workload reference. For a target GPU, peers are the GPUs on the same IP that are running the same task at the same time; in our cluster, this group contains up to eight GPUs. When peer measurements are available, the encoder computes peer means, dispersions, z-scores, differences from peer means, ranks, and isolated-error indicators. These features ask whether one GPU is abnormal relative to its same-host, same-task context, which helps separate single-card degradation from collective workload shifts.

TABLE I
RANKED PRECURSOR INDICATORS USED FOR TEMPORAL AND PEER-RELATIVE ENCODING.

Rank	Indicator	Rank	Indicator
1	Power draw	6	Memory utilization
2	GPU memory temperature	7	PCIe RX throughput
3	GPU temperature	8	PCIe TX throughput
4	NVLink RX throughput	9	Power-brake slowdown
5	NVLink TX throughput	10	SM clock

Table I lists the ranked indicators used to construct temporal and peer-relative features.

Sparse hazard streams are encoded as hardware-evolution signals. XID events, ECC increments, remapping-state changes, power-brake slowdown, event intervals, and consecutive hazardous states describe risk patterns that window averages can dilute. The deployed remapping configuration can additionally use TSFEL descriptors for dense signals [11]. Offline and deployed encoders are configuration-specific, but both produce fixed window-level representations for risk learning.

C. Fault-Specific Risk Learning

Fault-specific learning addresses heterogeneous mechanisms and warning horizons. We train one risk model per fault type because thermal faults, remapping failures, power anomalies, and uncorrected-memory errors expose different signals and tolerate different alert timing. Candidate learners include XGBoost [12], LightGBM [13], CatBoost [14], Random Forest [15], and Isolation Forest [16]. These models fit heterogeneous tabular features containing imputed values, missingness indicators, sparse labels, and non-linear feature interactions.

For each target fault, validation selects the learner and event-policy parameters by event-level F1 over candidate learners, thresholds, N-of-K rules, and cooldown values. The best configuration is frozen before held-out evaluation. Because positive ticket windows are rare, *Falcon* avoids accuracy and fixed 0.5 thresholds, using precision-recall-oriented selection for imbalanced prediction [17]. The production remapping deployment uses a precision-oriented variant, as described in Section V.

D. Event-Level Warning Policy

The event layer converts scores into maintenance-facing alerts. Given threshold τ_f, a warning candidate is emitted only when at least N_f of the latest K_f scores exceed τ_f; after emission, further warnings for the same GPU and fault are suppressed for cooldown C_f. Thresholding controls sensitivity, persistence filters isolated spikes, and cooldown prevents repeated alerts from one sustained condition. The policy is fault-specific because useful horizons and alert burden differ by fault.

Because this layer changes alert count and timing, it is validated with the learner rather than attached afterward. The pipeline loads the frozen learner, threshold, persistence rule, and cooldown to generate held-out warning events. A faulty

case contributes one true positive if at least one warning occurs within the valid intervention horizon; alerts outside that horizon or on healthy cases are false positives. We therefore report case-level precision, recall, F1, and lead time after event formation, following time-series and online failure-prediction evaluation concerns [18], [19].

IV. Experiment

A. Setup

We evaluate *Falcon* on anonymized production telemetry and tickets from a ByteDance GPU cluster. The taxonomy uses the full corpus with more than 1,000 GPUs, while held-out experiments use a traceable subset of 313 GPUs sampled at one-minute intervals with telemetry and ticket artifacts. We study four ticket types: GPU Remapping Failure, GPU Power Unknown Error, GPU Temperature High, and GPU Uncorrected Error Over ByteDance Limit.

Data are split chronologically into training, validation, and test intervals with a 6:2:2 ratio. This matches deployment, where past behavior predicts future failures, and avoids leakage from non-temporal splits in which the same GPU or workload regime can expose future operating patterns during training.

The experiment uses two time scales. The window label uses a six-hour prediction horizon: a window is positive if the target ticket occurs within the next six hours, which gave the best validation performance among tested horizons. Window scores are then converted into events by threshold, N-of-K persistence, and cooldown. Event-level evaluation uses a 48-hour intervention horizon selected by ByteDance operators for inspection, migration, isolation, or maintenance scheduling. Thus, six hours defines the learned risk target, whereas 48 hours defines when an alert remains actionable; it does not relax the prediction label. A faulty case is a true positive if any warning appears within 48 hours before its ticket. Missed faulty cases are false negatives; warnings on healthy cases are false positives. Lead time is measured from the first valid warning to the ticket and is reported only for detected cases. Offline experiments used Ubuntu/Python with standard gradient-boosting and scikit-learn libraries.

B. Baselines

We compare *Falcon* with four baselines. Liu et al. [5] represents GPU-specific failure prediction. Directly transferring this predictor to our setting is challenging because our features mainly describe short-term GPU telemetry dynamics, while the original study also uses broader contextual information such as GPU type, driver version, machine-room or rack location, and device aging. Without such context, the baseline has limited ability to explain abrupt or environment-related failures.

The LSTM baseline represents recurrent time-series anomaly detection [20]. LSTMs are suited to continuous temporal trends, but GPU fault precursors can be sparse, abrupt, or intermittent, such as sudden counter increments, transient ECC/remap-state changes, and short pre-fault windows with no clear trend. As a result, LSTM can cover some true failures and often achieves high recall, but it struggles to form precise decision boundaries under unstable short-window signals.

The Transformer baseline is trained from scratch on the same task [21]; few and heterogeneous positive cases may limit the benefit of training sequence models from scratch.

The ChatTime 7B Embedding baseline uses ChatTime 7B as a general embedding extractor with a lightweight classifier [22]. It may capture generic numerical variation, while fault-specific structure may be needed to preserve physical semantics, cross-metric relations, and temporal context in GPU telemetry.

TABLE II
Event-level comparison on the held-out test set.

Fault	Method	F1	Precision	Recall	Median lead (h)
Remapping Failure	***Falcon***	**0.500**	**0.611**	0.423	22.00
	Transformer	0.308	0.308	0.308	17.01
	ChatTime	0.229	0.182	0.308	**27.50**
	Liu et al.	0.071	0.067	0.077	1.06
	LSTM	0.495	0.343	**0.885**	2.75
Power Unknown Error	***Falcon***	**0.333**	**0.219**	0.700	**35.57**
	Transformer	0.267	0.200	0.400	23.77
	ChatTime	0.159	0.094	0.500	23.03
	Liu et al.	0.074	0.059	0.100	2.43
	LSTM	0.247	0.143	**0.900**	2.85
Temperature High	***Falcon***	**0.706**	**0.632**	**0.800**	**23.02**
	Transformer	0.323	0.312	0.333	22.03
	ChatTime	0.435	0.625	0.333	14.00
	Liu et al.	0.074	0.083	0.067	1.42
	LSTM	0.588	0.526	0.667	2.63
Uncorrected Error Limit	***Falcon***	**0.444**	**0.667**	0.333	17.34
	Transformer	0.111	0.067	0.333	14.50
	ChatTime	0.090	0.047	**1.000**	**17.90**
	Liu et al.	0.005	0.003	0.167	0.67
	LSTM	0.154	0.083	**1.000**	2.65

C. RQ1: Effectiveness Across Fault Types

Table II reports the main event-level comparison. For compactness, Uncorrected Error Limit abbreviates GPU Uncorrected Error Over ByteDance Limit. *Falcon* achieves the highest F1 among the four baselines on all four faults. Temperature High is the most predictable target, reaching 0.632 precision, 0.800 recall, and 0.706 F1. Remapping Failure is also stable, with 0.611 precision and 0.500 F1. Power Unknown Error remains difficult because *Falcon* improves recall but still has low precision. Uncorrected Error reaches high precision.

The best *Falcon* learner also differs by fault: CatBoost is selected for Remapping Failure, Random Forest for Power Unknown Error, and LightGBM for Temperature High and Uncorrected Error. This pattern is consistent with the fault-specific design: the four ticket types expose different precursor patterns, sample sizes, and precision–recall trade-offs.

D. RQ2: Ablation Study

Table III tests the two design choices most visible to operators: fault-specific learning and event-policy formation. "All" denotes one model for all faults. Removing N-of-K persistence often increases sensitivity but also raises false positives. For Remapping Failure, recall rises from 0.423 to 0.692, but F1 drops from 0.500 to 0.409. Removing cooldown

TABLE III
ABLATION RESULTS ON THE HELD-OUT TEST SET.

Fault	Variant	F1	Precision	Recall	Median lead (h)
Remapping Failure	Full	**0.500**	**0.611**	0.423	22.00
	w/o persistence	0.409	0.290	0.692	21.02
	w/o cooldown	0.278	0.500	0.192	20.03
	All	0.339	0.214	**0.808**	**25.00**
Power Unknown Error	Full	**0.333**	0.219	**0.700**	**35.57**
	w/o persistence	0.267	0.200	0.400	11.89
	w/o cooldown	0.182	**1.000**	0.100	4.93
	All	**0.333**	0.219	**0.700**	**35.57**
Temperature High	Full	**0.706**	**0.632**	0.800	23.02
	w/o persistence	0.552	0.571	0.533	9.02
	w/o cooldown	0.167	0.143	0.200	9.70
	All	0.500	0.351	**0.867**	**24.38**
Uncorrected Error Limit	Full	**0.444**	**0.667**	0.333	**17.34**
	w/o persistence	0.250	0.200	0.333	14.30
	w/o cooldown	0.286	0.250	0.333	5.00
	All	0.273	0.187	**0.500**	12.13

degrades all four faults, confirming that alert deduplication is part of the predictive system rather than only a post-processing convenience. A shared model often increases recall, but its precision drops, showing that one learner does not fit all fault mechanisms.

E. RQ3: Lead-Time Analysis

Falcon provides 17.34–35.57 hours of median lead time among detected cases, within the 48-hour actionable horizon for inspection, isolation, or maintenance scheduling. Lead time must be interpreted with alert quality. ChatTime gives longer median lead time for Remapping Failure but only 0.182 precision; on Uncorrected Error it detects all faulty cases but raises 121 false-positive cases, reducing precision to 0.047. Earlier warning is useful only under a comparable precision or alert budget. Overall, *Falcon* improves the precision–recall trade-off while preserving actionable lead time.

V. DEPLOYMENT

In production, the system uses a precision-oriented strategy tailored to operational constraints. Unlike the offline study, deployment must account for per-alert costs: workload migration, scheduling changes, diagnostics, stress validation, and manual inspection. The objective therefore shifts toward high alert precision because each false positive triggers costly operational work. Accordingly, deployment calibration uses a precision-oriented objective rather than the offline F1 objective, and the model output is treated as an operational trigger rather than a purely statistical score.

The deployed pipeline couples periodic retraining with online inference. It aggregates temperatures, power draw, throttling indicators, XID counts, uncorrectable ECC errors, and remapped-row metrics, then derives TSFEL-based [11] temporal descriptors and rule-based failure features over sliding windows. Retraining uses a rolling 100-day window, with 80 days for training and 20 days for validation. The threshold is selected on the precision-recall curve to maximize precision under a minimum-recall constraint, and a model is promoted only if validation precision satisfies the deployment criterion. At inference time, the system uses the selected feature subset and persisted threshold to preserve training-serving consistency.

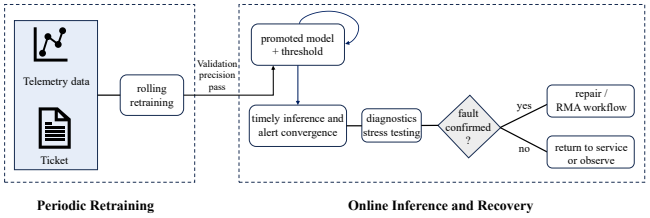


Fig. 3. *Falcon* production deployment workflow.

Online inference runs every 20 minutes. Each run retrieves the latest model artifacts and applies the same aggregation, imputation, sliding-window feature construction, and probabilistic scoring to new telemetry. To suppress repeated alerts during sustained degradation, convergence keeps only the earliest high-confidence alert for each IP within 48 hours. This converts continuous risk scores into a tractable alert stream aligned with ticketing, migration, and diagnostic workflows.

A high-confidence alert is not treated as definitive evidence of hardware failure. It triggers proactive intervention: workloads are migrated away from the suspected host, and the alert is checked through diagnostics, stress testing, and hardware-state inspection. Confirmed faults enter repair or return merchandise authorization (RMA); unconfirmed hosts return to service or remain under observation. This couples model-based prediction, risk isolation, and human-in-the-loop validation.

The deployment ran for one month on a production GPU cluster covering more than 10,000 GPUs. After convergence, the system generated 50 alerts, of which 40 were confirmed by engineers. Because the complete denominator of production remapping failures is confidential, we report precision but not recall; the result therefore characterizes alert confidence rather than overall failure coverage. These results show that the precision-oriented strategy can produce a tractable high-confidence alert stream and connect offline prediction to practical intervention through migration, diagnosis, and RMA decisions.

VI. RELATED WORK

A. GPU Cluster Reliability and Diagnosis

GPU-cluster studies mainly address reliability characterization, runtime diagnosis, and mitigation. Astral describes monitoring, diagnosis, and operational mechanisms for large-scale LLM training infrastructure [2]; *Revisiting Reliability in Large-Scale Machine Learning Research Clusters* quantifies job failures, faults, and resource usage in multi-tenant machine learning clusters [3]; and earlier HPC work characterizes GPU errors and operational implications [23]. Runtime systems

such as Holmes, GREYHOUND, Minder, and Oobleck localize irregular training behavior, detect fail-slow behavior, identify faulty machines, or make distributed training resilient after symptoms appear [7]–[10]. *Falcon* addresses a different operating point: infrastructure-level telemetry and case-level warning before maintenance tickets occur.

B. GPU Failure Prediction and Predictive Maintenance

Forecasting Machine Degradation of GPU Clusters is the most closely related GPU-specific predictive-maintenance study [4]. It uses multivariate telemetry to forecast GPU-cluster degradation, showing that production telemetry can contain pre-failure information. *Predicting GPU Failures With High Precision Under Deep Learning Workloads* studies production GPU failure prediction and uses ensemble and sliding-training techniques to improve precision and stability [5]. Machine-learning models for GPU error prediction in large-scale HPC systems provide another related predictive setting [6]. Outside GPU clusters, Xu et al. predict disk errors in cloud systems to improve service availability [24]; Falcon shares this operational motivation but targets heterogeneous GPU fault families and converts scores into fault-specific, case-level warning events. *Falcon* differs by defining fault-specific ticket targets and evaluating GPU-level warning events with persistence, cooldown, precision, recall, F1, and lead time.

VII. Conclusion

Actionable GPU warning requires more than high risk scores. Production telemetry is incomplete, workload-entangled, fault-specific, and costly to operationalize when false alarms consume engineer attention. Falcon addresses this setting by qualifying predictable faults, building temporal and peer-relative evidence, selecting learners per fault, and validating threshold, persistence, and cooldown as one event policy. Across four held-out fault types, Falcon selects distinct learners and operating profiles while improving event-level F1 over the implemented baselines. The one-month deployment shows that the workflow can produce a high-confidence alert stream in a cluster with more than 10,000 GPUs.

References

[1] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong *et al.*, "Megascale: Scaling large language model training to more than 10,000 gpus," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 745–760.

[2] Q. Meng, H. Zheng, Z. Zhang, C. Lao, C. Huang, B. Li, Z. Zhu, H. Lu, W. Dang, Z. Lin *et al.*, "Astral: A datacenter infrastructure for large language model training at scale," in *Proceedings of the ACM SIGCOMM 2025 Conference*, 2025, pp. 609–625.

[3] A. Kokolis, M. Kuchnik, J. Hoffman, A. Kumar, P. Malani, F. Ma, Z. DeVito, S. Sengupta, K. Saladi, and C.-J. Wu, "Revisiting reliability in large-scale machine learning research clusters," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2025, pp. 1259–1274.

[4] S. Cai, S. Nie, Z. Chen, N. Gulalkari, G. Vanica, C. Jain, and S. Sankaran, "Forecasting machine degradation of gpu clusters," in *Machine Learning for Systems 2025*, 2025.

[5] H. Liu, Z. Li, C. Tan, R. Yang, G. Cao, Z. Liu, and C. Guo, "Predicting gpu failures with high precision under deep learning workloads," in *Proceedings of the 16th ACM International Conference on Systems and Storage*, 2023, pp. 124–135.

[6] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. Tiwari, "Machine learning models for gpu error prediction in a large scale hpc system," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 95–106.

[7] Z. Yao, P. Hu, C. Miao, X. Jia, Z. Liang, Y. Xu, C. He, H. Lu, M. Chen, X. Li *et al.*, "Holmes: Localizing irregularities in {LLM} training with mega-scale {GPU} clusters," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, 2025, pp. 523–540.

[8] Y. Deng, X. Shi, Z. Jiang, X. Zhang, L. Zhang, Z. Zhang, B. Li, Z. Song, H. Zhu, G. Liu *et al.*, "Minder: Faulty machine detection for large-scale distributed model training," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, 2025, pp. 505–521.

[9] T. Wu, W. Wang, Y. Yu, S. Yang, W. Wu, Q. Duan, G. Yang, J. Wang, L. Qu, and L. Zhang, "{GREYHOUND}: Hunting {Fail-Slows} in {Hybrid-Parallel} training at scale," in *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, 2025, pp. 731–747.

[10] I. Jang, Z. Yang, Z. Zhang, X. Jin, and M. Chowdhury, "Oobleck: Resilient distributed training of large models using pipeline templates," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 382–395.

[11] M. Barandas, D. Folgado, L. Fernandes, S. Santos, M. Abreu, P. Bota, H. Liu, T. Schultz, and H. Gamboa, "Tsfel: Time series feature extraction library," *SoftwareX*, vol. 11, p. 100456, 2020.

[12] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[13] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[14] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," *Advances in neural information processing systems*, vol. 31, 2018.

[15] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[16] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[17] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets," *PloS one*, vol. 10, no. 3, p. e0118432, 2015.

[18] N. Tatbul, T. J. Lee, S. Zdonik, M. Alam, and J. Gottschlich, "Precision and recall for time series," *Advances in neural information processing systems*, vol. 31, 2018.

[19] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Computing Surveys (CSUR)*, vol. 42, no. 3, pp. 1–42, 2010.

[20] P. Malhotra, L. Vig, G. Shroff, P. Agarwal *et al.*, "Long short term memory networks for anomaly detection in time series," in *Proceedings*, vol. 89, no. 9, 2015, p. 94.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[22] C. Wang, Q. Qi, J. Wang, H. Sun, Z. Zhuang, J. Wu, L. Zhang, and J. Liao, "Chattime: A unified multimodal time series foundation model bridging numerical and textual data," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 12, 2025, pp. 12 694–12 702.

[23] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux *et al.*, "Understanding gpu errors on large-scale hpc systems and the implications for system design and operation," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 331–342.

[24] Y. Xu, K. Sui, R. Yao, H. Zhang, Q. Lin, Y. Dang, P. Li, K. Jiang, W. Zhang, J.-G. Lou, M. Chintalapati, and D. Zhang, "Improving service availability of cloud systems by predicting disk error," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, 2018, pp. 481–494.