

# Machine Learning Based Prediction of Containerized Application Failures Using Multivariate Runtime Resource Consumption Patterns: A Review

Km Amrita Upadhyay<sup>1</sup>, Mrs. Arifa Khan<sup>2</sup>

<sup>1</sup>Master of Technology, Computer Science and Engineering, Lucknow Institute of Technology, Lucknow, India

<sup>2</sup>Assistant Professor, Department of Computer Science and Engineering, Lucknow Institute of Technology, Lucknow, India

\*\*\*

**Abstract** - Containerization has become the dominant deployment paradigm for cloud-native and microservice-based software systems, owing to its lightweight isolation, portability, and compatibility with elastic orchestration platforms such as Kubernetes and Docker Swarm. However, the same properties that make containers attractive high density, dynamic scaling, and short lifecycles also make containerized applications susceptible to complex, cascading, and often silent failures that are difficult to diagnose using conventional threshold-based monitoring. Runtime resource consumption metrics such as CPU utilization, memory footprint, disk input/output, and network throughput exhibit multivariate, non-stationary, and inter-dependent behaviour that precedes many classes of application failure. Consequently, a substantial body of research has investigated the use of machine learning (ML) and deep learning (DL) techniques to model these multivariate runtime patterns and to predict failures before they manifest as user-visible outages. This paper presents a structured review of the state of the art in ML-based failure prediction for containerized applications using multivariate runtime resource consumption data. We organize the literature into a taxonomy spanning classical supervised learning, ensemble methods, deep sequence models, graph-based approaches, and unsupervised/semi-supervised anomaly detection. We further examine the datasets, feature engineering strategies, class-imbalance mitigation techniques, and evaluation protocols commonly employed in this domain, and we synthesize a comparative analysis of representative studies. Finally, we identify open research gaps including the scarcity of container-native labelled failure datasets, the lack of standardized multivariate benchmarks, limited interpretability of deep models, and the unresolved challenge of predicting failures under autoscaling and we outline promising directions for future research, including federated and online learning, causal and explainable failure attribution, and multimodal fusion of metrics, logs, and traces.

**Key Words:** containerized applications, failure prediction, machine learning, deep learning, Kubernetes, microservices, multivariate time series, resource consumption, anomaly detection, cloud reliability.

## 1. INTRODUCTION

Cloud-native computing has fundamentally reshaped how software is packaged, deployed, and operated. Containers, popularized by Docker and standardized through the Open Container Initiative, encapsulate an application along with its dependencies into a portable, lightweight unit that can be instantiated in milliseconds and orchestrated at scale by platforms such as Kubernetes [3], [23]. Enterprises increasingly decompose monolithic applications into fine-grained microservices, each deployed as one or more containers that communicate over the network, scale independently, and are subject to automated placement, scheduling, and healing decisions by the orchestration layer [3]. This architectural shift has delivered substantial gains in deployment velocity, resource efficiency, and elasticity; however, it has simultaneously introduced a new class of operational complexity.

Failures in containerized environments rarely originate from a single, easily observable root cause. Instead, they frequently emerge from the interaction of multiple runtime resource dimensions CPU throttling induced by cgroup limits, memory pressure leading to out-of-memory (OOM) kills, network saturation between dependent microservices, or disk I/O contention on shared nodes that jointly and progressively degrade application health before an outright crash, restart, or service-level agreement (SLA) violation occurs [1], [2]. Because containers are short-lived, horizontally autoscaled, and frequently rescheduled across nodes, the number of monitored entities and the shape of the monitored metric set change continuously over time, a property that breaks many of the fixed-input assumptions used by classical statistical monitoring and by early neural failure predictors [2]. Traditional threshold-based alerting, which triggers a notification when a single metric crosses a static boundary, is consequently both noisy (producing excessive false positives during legitimate load spikes) and blind (missing failures that arise from the joint evolution of several metrics, none of which individually breaches its threshold) [2], [17].

Machine learning offers a data-driven alternative: by learning the normal multivariate relationship among runtime resource consumption signals from historical

telemetry, a predictive model can flag deviations that are statistically anomalous even when no single metric exceeds a conventional threshold, and when trained with labelled failure outcomes can forecast the probability of an imminent failure sufficiently in advance for mitigation actions such as pod eviction, traffic rerouting, or preemptive scaling to be triggered [1], [6], [12]. Over the last decade, a large and fragmented body of work has applied classical classifiers (logistic regression, decision trees, random forests, gradient boosting), deep sequence models (recurrent neural networks, LSTM, GRU), and, more recently, graph neural networks and attention-based architectures to this problem, using workload traces from Google, Alibaba, and other hyperscale operators as well as bespoke microservice testbeds such as TrainTicket [1], [4], [6], [8], [9], [11].

The purpose of this review is to consolidate this fragmented literature into a coherent picture specifically focused on the use of multivariate runtime resource consumption patterns as opposed to logs, traces, or purely structural metadata as the primary predictive signal for container and containerized-application failure. The review is intended for postgraduate researchers who require both a conceptual map of the field and a practical understanding of the datasets, features, and evaluation practices that a new study in this area must engage with.

### 1.1. Motivation and Scope

This review restricts its scope to studies in which (i) the target system is containerized, orchestrated, or explicitly cloud-native (including container-based cluster schedulers such as those studied via the Google cluster workload traces), and (ii) the predictive features are derived, wholly or substantially, from multivariate runtime resource consumption measurements CPU, memory, disk, network, and related utilization or request/limit signals rather than exclusively from source code metrics, static configuration, or purely textual log content. Studies that use logs or distributed traces are included where they are fused with resource metrics, since such multimodal approaches represent an important and growing direction, but log-only or trace-only anomaly detection is discussed only for context.

### 1.2. Review Methodology

The literature synthesized in this review was identified through a structured search of major digital libraries (IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and arXiv) using combinations of the keywords "container failure prediction," "microservice anomaly detection," "Kubernetes machine learning," "cloud task failure prediction," "multivariate time series anomaly detection," and "resource utilization prediction." Candidate papers were screened for relevance to containerized or cloud-cluster environments, for the explicit use of multivariate runtime resource features,

and for methodological clarity (dataset description, feature set, and reported evaluation metrics). The review synthesizes findings across three related sub-literatures that are often treated separately but are, for the purposes of failure prediction, closely coupled: (1) task/job failure prediction on large-scale cluster traces such as the Google Cluster Trace, which although originally framed around jobs and tasks rather than containers per se has become a de facto benchmark for the underlying prediction problem [1], [6], [7], [8], [9], [17]; (2) microservice and Kubernetes-native anomaly and failure prediction, which explicitly targets containerized deployments and autoscaling [2], [3], [4]; and (3) the broader multivariate time-series anomaly detection literature, which supplies many of the model architectures subsequently adapted to the container domain [13], [15], [16].

## 2. BACKGROUND: CONTAINERS, ORCHESTRATION, AND FAILURE SEMANTICS

### 2.1. Containers and Orchestration Platforms

A container packages an application with its runtime dependencies while sharing the host operating system kernel, using kernel-level namespaces and control groups (cgroups) to provide process, filesystem, and resource isolation [23]. This is markedly lighter-weight than full virtual-machine virtualization, enabling higher deployment density and faster start-up. Orchestration platforms such as Kubernetes automate the scheduling, placement, replication, and self-healing of containers, organizing them into logical units (pods) that are distributed across a cluster of worker nodes and that can be horizontally autoscaled in response to observed load [2], [3]. Because Kubernetes continuously creates, migrates, and terminates pods, the population of monitored entities and hence the dimensionality of the resource-consumption feature space varies over time, a property with direct consequences for how failure-prediction models must be designed [2].

### 2.2. Failure Semantics in Containerized Systems

Failures in this domain manifest at several levels. At the container level, a process may be killed by the kernel OOM killer when memory limits are exceeded, or a liveness/readiness probe may fail, triggering a restart. At the pod level, repeated container crashes lead to a CrashLoopBackOff state. At the microservice level, cascading failures can occur when one service's resource exhaustion or latency degradation propagates through synchronous call chains to dependent services, eventually manifesting as SLA violations or full outages even though no individual container has crashed [2], [4]. At the cluster or node level, node-wide resource pressure, disk exhaustion, or hardware faults can precipitate the eviction of many pods simultaneously [11]; comparable node- and hardware-level failure phenomena have also been studied in virtualized

high-performance-computing (HPC) clusters, where multivariate resource and health telemetry is similarly used to anticipate imminent node failure [26]. Analogous phenomena have long been studied in the context of cluster job scheduling, where a task or job is marked as "Fail," "Evict," or "Kill" in scheduler traces such as the Google Cluster Trace, and where resource request/usage discrepancies are strongly correlated with the probability of failure [1], [17]. Because these failure modes differ in latency (the length of the degradation window prior to failure), granularity (container versus service versus node), and observability (whether they are captured purely in metrics or also in logs and traces), the choice of predictive features, prediction horizon, and evaluation protocol must be matched to the specific failure semantics under study.

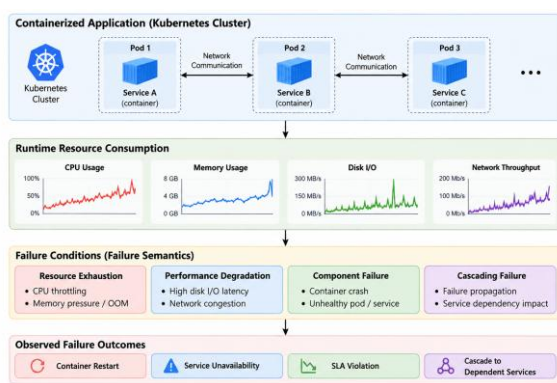


**Figure 1: Failure Semantics in Containerized Systems**

## 2.3. Multivariate Runtime Resource Consumption Signals

The runtime telemetry available to a failure-prediction model typically spans several correlated dimensions: CPU utilization and CPU throttling relative to configured limits; memory usage relative to requests and limits, including cache and resident-set-size decomposition; network ingress/egress throughput and connection counts; disk read/write throughput and I/O wait; and, in orchestration-aware settings, scheduling and scaling metadata such as replica counts, restart counts, and pod age [2], [3], [12]. These signals are inherently multivariate because failures are rarely caused by a single resource dimension in isolation; rather, they emerge from the joint evolution of several metrics for instance, sustained high CPU utilization coinciding with rising memory pressure and increasing queue length is a stronger failure precursor than any one of these signals alone [2]. This inter-dependency is the central methodological reason that univariate, single-metric threshold monitoring is inadequate and that models capable of learning cross-metric and temporal correlations such as recurrent networks, graph neural networks over the dependency structure of microservices, and multivariate statistical models have become the preferred approach in recent literature [4], [13], [15].

## 3. TAXONOMY OF MACHINE LEARNING APPROACHES

The reviewed literature can be organized into four broad methodological families based on the nature of the learning algorithm and the supervision signal employed, summarized in Fig. 1 (conceptually) and elaborated below: (A) classical supervised learning on engineered features, (B) ensemble and boosting methods, (C) deep sequence and representation-learning models, and (D) unsupervised/semi-supervised anomaly detection for settings where labelled failures are scarce.

### 3.1. Classical Supervised Learning

Early and still widely used approaches cast failure prediction as a binary or multi-class classification problem over a feature vector describing a task, job, or container at a given point in time, using algorithms such as logistic regression, decision trees, k-nearest neighbours, naive Bayes, and support vector machines. Jassas and Mahmoud [1] compared Naive Bayes, decision trees, random forests, logistic regression, quadratic discriminant analysis, and k-nearest neighbours for task failure prediction on the Google, Mustang, and Trinity cluster traces, finding that random forest and decision-tree classifiers, combined with recursive feature elimination for feature selection, achieved precision, recall, and F1-scores around 99% on the Google trace, and that unsuccessful tasks correlated strongly with requested resource levels. Padmakumari et al., as summarized in [28], likewise compared k-nearest neighbours, decision trees, support vector machines, and naive Bayes on the Google cluster dataset. Tengku Asmawi et al. [6] provide one of the more comprehensive comparative evaluations, benchmarking logistic regression, decision tree, random forest, gradient boosting, and XGBoost against single-, double-, and triple-layer LSTM networks on the Google Cluster Trace, reporting that XGBoost and random forest achieved the strongest job- and task-level performance respectively, while logistic regression offered the best scalability for large-scale deployment. These studies consistently find that classical tree-based models remain highly competitive with deep architectures on structured, tabular resource-consumption features, particularly when interpretability and computational efficiency at inference time are important operational constraints.

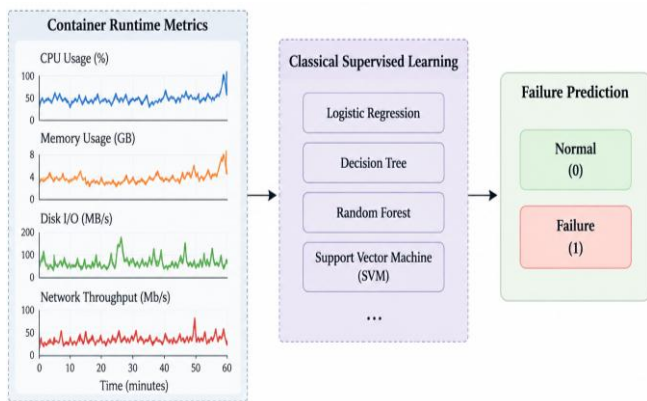


Figure 2: Classical Supervised Learning

### 3.2. Ensemble and Boosting Methods

Ensemble methods that aggregate the predictions of many weak learners, most notably random forests [19] and gradient-boosted tree ensembles, particularly XGBoost [20], have emerged as a consistently strong performer across the reviewed studies. Shetty et al., summarized in [6] and [28], applied XGBoost together with resampling strategies random undersampling, random oversampling, and SMOTE [18] to the Google Cluster Trace and reported precision and recall above 90% when SMOTE was used to counter the pronounced class imbalance between successful and failed tasks. Related work described in [28] similarly reports XGBoost achieving over 94% accuracy for job failure prediction, with disk-space and CPU request features identified as the most influential predictors. Multi-layer ensemble architectures that stack heterogeneous base classifiers decision trees, k-nearest neighbours, XGBoost, AdaBoost, and artificial neural networks followed by a second-stage random-forest classifier for failure-type identification have reported detection accuracies in excess of 99% on more recent (2019) releases of the Google Cluster Trace, with a companion characterization study of the same 2019 release further profiling application failure characteristics and user behaviour patterns across the trace [27], [29]. The general finding across this sub-literature is that ensembles of trees, by aggregating many weak learners over engineered resource-consumption and request features, substantially reduce variance relative to single classifiers and handle the heavy class imbalance inherent to failure data more gracefully than deep sequence models trained without resampling.

### 3.3. Deep Sequence and Representation-Learning Models

Because runtime resource consumption is fundamentally a time series, recurrent architectures most prominently long short-term memory (LSTM) networks [21] and gated recurrent units have been widely applied to capture temporal dependencies that static, snapshot-based

features cannot express. Gao, Wang, and Shen [7] proposed a deep-learning framework for task failure prediction in cloud data centers using recurrent architectures over Google cluster time series, motivated by the observation that support vector machines and hidden semi-Markov models struggle to scale to the volume and dimensionality of cluster traces, while LSTM networks better capture long-term temporal dependencies, albeit with the drawback of not adaptively weighting the relative importance of more distant time steps [7]. Tengku Asmawi et al. [6] similarly evaluate single-, bi-, and tri-layer LSTM architectures alongside classical baselines, finding that deeper LSTM stacks do not uniformly outperform well-tuned tree ensembles on tabular Google Cluster Trace features, underscoring that the benefit of recurrent depth depends strongly on whether the available features retain meaningful temporal structure.

For containerized microservice systems specifically, Denaro et al. [2] introduce Preface, which augments an autoencoder-based neural failure predictor with a preprocessing "Rectifier" step that reduces the variable-size set of key performance indicators (KPIs) produced by autoscaling pods into a constant-size set of descriptive statistics, allowing the predictor to function even as the number of running container replicas changes over time a limitation that fixed-input neural predictors cannot otherwise handle. Preface further ranks the microservices most likely responsible for an impending failure, combining prediction with root-cause localization. Graph-based deep learning has also been applied where the dependency structure among microservices is itself informative: Chen et al. [4] propose TraceGra, an unsupervised encoder-decoder architecture that combines a graph neural network with an LSTM to jointly capture the topological structure of inter-service call graphs and the temporal evolution of container-level performance metrics, reporting strong precision and recall on both an open-source benchmark and a locally collected ARM-cluster dataset, and explicitly noting a strong linear correlation between core-microservice resource usage and end-to-end response time. Such graph-augmented approaches represent an important refinement over models that treat each container's metrics independently, since they allow resource-consumption anomalies to be interpreted in the context of upstream and downstream service dependencies.

### 3.4. Unsupervised and Semi-Supervised Anomaly Detection

Because labelled failure events are comparatively rare relative to the volume of normal operational telemetry, and because exhaustively labelling every failure mode in a production system is impractical, a substantial portion of the literature frames the problem as unsupervised or semi-supervised anomaly detection over multivariate time series, learning a model of normal behaviour and flagging statistically significant deviations as candidate failure

precursors [13], [15], [16]. Generative approaches such as MAD-GAN [15] use a generative adversarial network trained jointly with an LSTM-based generator and discriminator to capture both temporal dependence within each variable and latent spatial correlations across variables, using the discrimination and reconstruction loss as an anomaly score. Autoencoder-based approaches, including the Rectifier-augmented autoencoder in Preface [2], learn a compressed latent representation of normal multivariate resource behaviour and flag high reconstruction error as anomalous. Darban et al. [13] provide a comprehensive taxonomy of deep anomaly detection methods for multivariate time series, organizing the field by learning paradigm (supervised, unsupervised, semi-supervised, self-supervised) and by architecture (recurrent, convolutional, graph-based, transformer-based, and generative), noting that unsupervised approaches dominate the anomaly-detection literature precisely because production failure labels are scarce, while highlighting that evaluation practice in this area remains inconsistent across studies. A practitioner-oriented evaluation by Dhariyal et al. [14] further cautions that many unsupervised multivariate anomaly detectors that perform well on academic benchmarks degrade substantially when deployed against large, heterogeneous, real-world cloud telemetry, underscoring a persistent gap between benchmark performance and production reliability.

## **4. MULTIVARIATE RUNTIME RESOURCE CONSUMPTION FEATURES**

### **4.1. Feature Categories**

Across the reviewed studies, runtime resource consumption features cluster into several recurring categories. Compute-related features include CPU usage rate, CPU request-to-limit ratio, CPU throttling counts, and, in cluster-trace studies, the discrepancy between requested and actually consumed CPU, which several studies identify as one of the strongest individual predictors of task failure [1], [28]. Memory-related features include working-set memory, resident-set size, page-cache usage, memory request-to-limit ratio, and OOM-kill counts. Network features include ingress/egress bytes and packets, connection counts, and, in microservice settings, request latency and error rate, which serve as an application-level proxy for resource-induced degradation [4]. Disk features include read/write throughput and I/O wait time, which are particularly relevant for hardware- and node-level failure studies [11]. Finally, orchestration-level features specific to containerized deployments include restart counts, pod age, replica count (relevant for autoscaling-aware models such as Preface [2]), and scheduling priority, the last of which has been identified as the most influential feature for task-level failure prediction in several Google Cluster Trace studies [28].

### **4.2. Feature Engineering and Windowing**

Because raw telemetry arrives as high-frequency multivariate streams, virtually all reviewed studies apply some form of windowing or aggregation prior to model training: fixed-length sliding windows over which descriptive statistics (mean, variance, minimum, maximum, rate of change) are computed, or, in the case of recurrent and graph-based models, sequences of raw or lightly normalized time steps supplied directly to the network [2], [4], [6]. A distinguishing challenge in the container setting, addressed explicitly by Preface [2], is that the number of monitored entities (pods) contributing to a given window varies as the application autoscales a consequence of horizontal autoscaling policies that dynamically add and remove replicas in response to load, as governed by Kubernetes' native autoscaling controllers [24], [25] which breaks the fixed-dimensionality assumption of standard neural inputs unless an entity-invariant aggregation (such as the descriptive-statistics Rectifier) is applied. Feature selection techniques most commonly recursive feature elimination and importance ranking from tree-based models — are widely used both to improve predictive accuracy and to identify which resource dimensions are operationally most diagnostic, with CPU and memory request/usage features and scheduling priority recurring as top-ranked predictors across independent studies [1], [28].

### **4.3. Class Imbalance and Data Quality**

Failure events are, by nature, rare relative to normal operation, and virtually every supervised study in this review reports significant class imbalance in its dataset, with failed tasks or jobs typically representing a small minority of records. The Synthetic Minority Over-sampling Technique (SMOTE) [18], random oversampling, and random undersampling are the most commonly applied mitigation strategies, and multiple studies report that SMOTE combined with gradient boosting yields the best precision–recall trade-off among the resampling strategies tested [6], [28]. Data quality is a further recurring concern: Kubernetes environments generate noisy, incomplete, or inconsistent telemetry, and missing metrics from a node can lead to false positives or false negatives in downstream detectors [10], a limitation that is frequently acknowledged but less frequently addressed methodologically in the reviewed literature.

## **5. DATASETS AND BENCHMARKS**

The Google Cluster Workload Trace remains the most widely used benchmark in this literature, providing scheduler-level records of job and task submission, resource requests, actual usage, and termination events (including Fail, Evict, and Kill states) across a production-scale cluster [1], [6], [17]. Its 2011 and 2019 releases differ substantially in scale and schema, and comparative studies caution that

results are not always directly transferable between releases [1], [29]. Beyond Google, studies have used the Alibaba cluster trace, the HPC-oriented Mustang and Trinity traces [1], and custom testbeds built around open-source microservice benchmark applications such as TrainTicket, deployed on Kubernetes clusters with injected faults to generate ground-truth failure labels [2], [4]. For the broader multivariate anomaly-detection literature that underpins many container-specific methods, benchmark datasets include large-scale AIOps telemetry collections comprising tens of millions of data points across hundreds to thousands of metrics, purpose-built to support forecasting, classification, and root-cause-analysis tasks in addition to anomaly detection [14]. A consistent theme across the reviewed work is the absence of a single, standardized, container-native benchmark that simultaneously provides (i) multivariate runtime resource telemetry at the container/pod granularity, (ii) ground-truth failure labels spanning multiple failure classes, and (iii) autoscaling dynamics representative of production Kubernetes deployments a gap discussed further in Section VII.

## 6. EVALUATION METRICS AND EXPERIMENTAL PROTOCOLS

Given the class imbalance inherent to failure data, precision, recall, and F1-score are the dominant reported metrics across the reviewed studies, generally preferred over raw accuracy, which can be misleadingly high on imbalanced datasets even for a trivial always-predict-normal classifier [1], [6], [28]. Several studies additionally report specificity and sensitivity, particularly in comparative frameworks evaluating multiple algorithms side by side [29]. For anomaly-detection formulations, precision and recall at the point-adjusted or event level are common, alongside F-measure computed under a chosen anomaly-score threshold, often calibrated using extreme value theory or similar dynamic thresholding schemes rather than a fixed cut-off. Cross-validation, most commonly 5-fold, is widely used for classical ML pipelines [1], while deep-learning studies more often adopt a chronological train/validation/test split to

respect the temporal ordering of the underlying trace and avoid information leakage from future to past. A minority of studies additionally report scalability or training/inference-time comparisons across algorithms, an operationally important but comparatively under-reported dimension given that failure-prediction models must ultimately run within tight latency budgets in production monitoring pipelines [6].

## 7. COMPARATIVE ANALYSIS OF REPRESENTATIVE STUDIES

Table I synthesizes representative studies discussed in Sections III–VI, contrasting the target system, primary methodological family, feature basis, and reported outcome. The comparison highlights three consistent patterns. First, tree-based ensembles (random forest, XGBoost, gradient boosting) applied to well-engineered, resampled tabular features remain highly competitive with, and in several studies superior to, deep sequence models on cluster-trace-style failure prediction, suggesting that the marginal value of representation learning depends heavily on whether the underlying features already encode the relevant temporal and cross-metric structure [1], [6], [28]. Second, studies that explicitly target containerized, autoscaling microservice systems as opposed to batch cluster-scheduler traces consistently find it necessary to introduce architectural modifications (variable-size KPI aggregation, graph-structured dependency modelling) that are unnecessary in the fixed-node cluster-trace setting, reflecting a genuine domain gap between classical cloud-cluster failure prediction and container-native failure prediction [2], [4]. Third, unsupervised and semi-supervised anomaly-detection approaches are disproportionately represented in production-oriented and container-native studies relative to the cluster-trace literature, consistent with the greater difficulty of obtaining exhaustive failure labels in live Kubernetes deployments [13], [14], [15].

**TABLE I: Comparative Summary of Representative Studies**

| Study                       | Target System                          | ML Family                    | Key Runtime Features                 | Reported Outcome                                              |
|-----------------------------|----------------------------------------|------------------------------|--------------------------------------|---------------------------------------------------------------|
| Jassas & Mahmoud [1]        | Google/Mustang/Trinity cluster traces  | RF, DT, LR, NB, QDA, kNN     | CPU/memory request & usage, priority | RF/DT with RFE: ~99% precision/recall/F1 (Google trace)       |
| Denaro et al. (Preface) [2] | Autoscaling microservices (Kubernetes) | Autoencoder + Rectifier (DL) | Variable-size KPI sets across pods   | Effective failure prediction & localization under autoscaling |
| Zhong et al. [3]            | Container orchestration                | Taxonomy of ML methods       | Multi-dimensional performance        | Comprehensive taxonomy; identifies research gaps              |

| Study                        | Target System                           | ML Family                                    | Key Runtime Features                        | Reported Outcome                                         |
|------------------------------|-----------------------------------------|----------------------------------------------|---------------------------------------------|----------------------------------------------------------|
|                              | (survey)                                |                                              | metrics                                     |                                                          |
| Chen et al. (TraceGra) [4]   | Microservices (TrainTicket, Kubernetes) | GNN + LSTM (unsupervised)                    | Trace graph + container performance metrics | Precision 0.97, Recall 0.93                              |
| Tengku Asmawi et al. [6]     | Google Cluster Trace                    | LR, DT, RF, GB, XGBoost, LSTM (1–3 layer)    | CPU/disk requests, task priority            | XGBoost/RF best; LR most scalable                        |
| Gao, Wang & Shen [7]         | Google Cluster Trace                    | RNN/LSTM (DL)                                | Time-series usage features                  | Improved over SVM/HSMM baselines                         |
| Shetty et al. (via [6],[28]) | Google Cluster Trace                    | XGBoost + SMOTE/ROS/RUS                      | Resource request/usage                      | ~92–94.8% precision/recall with SMOTE                    |
| MAD-GAN [15]                 | Cyber-physical / multivariate telemetry | LSTM-GAN (unsupervised)                      | Multi-sensor time series                    | Effective unsupervised anomaly detection                 |
| Darban et al. [13]           | Multivariate time series (survey)       | Taxonomy (RNN, GNN, Transformer, generative) | General MTS                                 | Structured taxonomy of DL anomaly detectors              |
| Dhariyal et al. [14]         | Production cloud telemetry              | Benchmark evaluation (unsupervised)          | 74.5M points, 1311 metrics                  | Reveals gap between benchmark and production performance |

## 8. OPEN CHALLENGES AND RESEARCH GAPS

### 8.1. Absence of Standardized Container-Native Benchmarks

Unlike the cluster-trace literature, which has converged on the Google Cluster Trace as a common benchmark [1], [6], [17], the container-native and microservice failure-prediction literature lacks an equivalent standardized, publicly available, multivariate dataset that captures pod-level resource telemetry, autoscaling dynamics, and ground-truth multi-class failure labels at production scale. Most container-native studies rely on bespoke testbeds (e.g., TrainTicket deployments with injected faults) [2], [4], which supports methodological validation but limits cross-study comparability.

### 8.2. Variable-Dimensionality and Autoscaling

As explicitly identified by Denaro et al. [2], the majority of neural failure predictors assume a fixed-size input vector corresponding to a statically configured set of monitored components, an assumption violated by autoscaling containerized applications whose pod population changes continuously. While Rectifier-style descriptive-statistics

aggregation and graph-based representations [2], [4] partially address this, a general, theoretically grounded treatment of variable-cardinality multivariate inputs for failure prediction remains an open problem.

### 8.3. Interpretability and Actionability

Deep sequence and graph-based models frequently outperform classical baselines only marginally, if at all, while sacrificing interpretability [6], [13]. For a review paper written from an operations perspective, this trade-off matters: an operator responding to a predicted failure needs not only a probability score but an indication of which resource dimension and which service is implicated, a requirement more naturally satisfied by feature-importance-transparent tree ensembles [1], [28] or explicitly localizing approaches such as Preface [2] than by opaque deep architectures.

### 8.4. Evaluation Under Realistic Class Imbalance and Concept Drift

Most reported evaluations rely on offline, static train/test splits over historical traces, whereas production failure prediction must operate under continuous concept drift as workloads, deployment topologies, and failure

modes evolve. The practitioner-oriented evaluation by Dhariyal et al. [14] demonstrates that models performing well on static benchmarks can degrade substantially in live, heterogeneous cloud environments, indicating that current evaluation protocols may systematically overstate real-world readiness.

## 8.5. Multimodal Fusion of Metrics, Logs, and Traces

While this review focuses on runtime resource consumption metrics, several of the strongest container-native results combine metrics with structural (trace/graph) information [4] or, in adjacent literature, with log data [16]. The systematic fusion of metrics, logs, and traces into a single multivariate representation rather than treating each modality with a separate model, as in log-and-trace fusion approaches such as DeepTraLog [5] remains comparatively underexplored specifically for the container failure-prediction problem, despite its demonstrated value in closely related microservice anomaly-detection work.

## 9. FUTURE RESEARCH DIRECTIONS

- Container-native benchmark construction: development of a publicly available, large-scale, multivariate dataset capturing pod-level resource telemetry, autoscaling events, and multi-class ground-truth failure labels across diverse workload types, analogous in stature to the Google Cluster Trace [1], [17] but purpose-built for Kubernetes-native prediction tasks.
- Cardinality-invariant and graph-based architectures: further development of models, following the Rectifier [2] and graph neural network [4] paradigms, that natively accommodate the changing number and topology of monitored containers under autoscaling and dynamic service dependency graphs.
- Explainable and causally-grounded prediction: integration of feature-attribution and causal-inference techniques with deep predictors so that a failure forecast is accompanied by an interpretable, actionable explanation identifying the responsible resource dimension and service, extending the localization capability demonstrated in [2].
- Online and federated learning: adaptation of failure predictors to continuous concept drift through online or continual learning, and exploration of federated learning across organizational boundaries or cluster tenants to build more generalizable models without centralizing sensitive operational telemetry.
- Multimodal fusion architectures: systematic combination of multivariate resource metrics with distributed traces and log embeddings within a single

learned representation, extending trace-metric fusion approaches such as TraceGra [4] and log-trace fusion approaches such as DeepTraLog to explicitly incorporate resource-consumption signals.

- Realistic, drift-aware evaluation protocols: adoption of rolling-window, concept-drift-aware evaluation and production-representative benchmarking, following the cautionary findings of practitioner-oriented studies [14], to close the gap between reported benchmark performance and operational reliability.
- Lightweight, edge-deployable predictors: given the latency-sensitive nature of failure mitigation, investigation of compact model architectures (distilled ensembles, quantized recurrent networks) capable of running within the resource envelope of the monitoring sidecar or node agent itself.
- Attention-based and transformer architectures: adaptation of self-attention mechanisms [22], which have proven effective at modelling long-range dependencies in sequential data, to multivariate runtime resource consumption streams, potentially offering a more parallelizable and longer-context alternative to recurrent architectures for container failure prediction.

## 10. CONCLUSION

This review has synthesized the state of the art in machine-learning-based prediction of containerized application failures using multivariate runtime resource consumption patterns. Across the classical, ensemble, deep-sequence, and unsupervised anomaly-detection families of methods surveyed, a consistent picture emerges: multivariate CPU, memory, network, and disk telemetry particularly when combined with orchestration-level signals such as restart counts and scheduling priority carries substantial predictive signal for impending failure, but the container-native setting introduces distinctive challenges (variable pod cardinality under autoscaling, cascading cross-service degradation, and severe class imbalance) that are not fully addressed by methods transplanted directly from the batch cluster-scheduling literature. Well-tuned tree ensembles remain strong, interpretable baselines, while graph-augmented and autoencoder-based deep architectures offer the clearest gains specifically in dynamic, containerized microservice settings where fixed-input assumptions break down. The field would benefit substantially from standardized, container-native, multivariate benchmarks; from architectures that are natively robust to autoscaling-induced variable dimensionality; from greater attention to interpretability and actionable root-cause localization; and from evaluation protocols that better reflect the concept drift and heterogeneity of production cloud environments. Addressing

these gaps represents a promising and tractable agenda for future postgraduate and applied research at the intersection of machine learning and cloud systems reliability.

## REFERENCES

- [1] M. S. Jassas and Q. H. Mahmoud, "Analysis of Job Failure and Prediction Model for Cloud Computing Using Machine Learning," *Sensors*, vol. 22, no. 5, p. 2035, Mar. 2022, doi: 10.3390/s22052035.
- [2] G. Denaro, N. El Moussa, R. Heydarov, F. Lomio, M. Pezzè, and K. Qiu, "Predicting Failures of Autoscaling Distributed Applications," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 1960–1981, Jul. 2024, doi: 10.1145/3660794.
- [3] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, "Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions," *ACM Comput. Surv.*, vol. 54, no. 10s, Art. 217, Sep. 2022, doi: 10.1145/3510415.
- [4] J. Chen, F. Liu, J. Jiang, G. Zhong, D. Xu, Z. Tan, and S. Shi, "TraceGra: A trace-based anomaly detection for microservice using graph deep learning," *Comput. Commun.*, vol. 204, pp. 109–121, Apr. 2023, doi: 10.1016/j.comcom.2023.03.028.
- [5] C. Zhang et al., "DeepTraLog: Trace-Log Combined Microservice Anomaly Detection through Graph-based Deep Learning," in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng. (ICSE)*, 2022.
- [6] T. N. T. Asmawi, A. B. Ismail, and J. Shen, "Cloud failure prediction based on traditional machine learning and deep learning," *J. Cloud Comput.*, vol. 11, Art. 47, Sep. 2022, doi: 10.1186/s13677-022-00327-0.
- [7] J. Gao, H. Wang, and H. Shen, "Task Failure Prediction in Cloud Data Centers Using Deep Learning," in *Proc. IEEE Int. Conf. Big Data (BigData)*, 2019, pp. 1111–1116.
- [8] M. S. Jassas and Q. H. Mahmoud, "Failure characterization and prediction of scheduling jobs in Google cluster traces," in *Proc. IEEE 10th GCC Conf. Exhib. (GCC)*, 2019, pp. 1–7.
- [9] J. Shetty, R. Sajjan, and G. Shobha, "Task resource usage analysis and failure prediction in cloud," in *Proc. 9th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, 2019, pp. 342–348.
- [10] N. El-Sayed, H. Zhu, and B. Schroeder, "Learning from Failure Across Multiple Clusters: A Trace-Driven Approach to Understanding, Predicting, and Mitigating Job Terminations," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2017, pp. 1333–1344.
- [11] Y. Li, Z. M. Jiang, H. Li, A. E. Hassan, C. He, R. Huang et al., "Predicting Node Failures in an Ultra-large-scale Cloud Computing Platform: An AIops Solution," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 2, pp. 1–24, 2020.
- [12] Z. Chen, J. Hu, G. Min, A. Y. Zomaya, and T. El-Ghazawi, "Towards Accurate Prediction for High-Dimensional and Highly-Variable Cloud Workloads with Deep Learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 923–934, Apr. 2020.
- [13] Z. Z. Darban, G. I. Webb, S. Pan, C. Aggarwal, and M. Salehi, "Deep Learning for Time Series Anomaly Detection: A Survey," *ACM Comput. Surv.*, vol. 57, no. 1, pp. 1–42, 2024.
- [14] B. Dhariyal, H. Ganzeli, K. Maslak, and D. Wijekoon, "Real or Not? A Practitioner's Perspective on Unsupervised Multivariate Time Series Anomaly Detection in Cloud Systems," in *Machine Learning, Optimization, and Data Science, Lecture Notes in Computer Science*, Springer, 2026.
- [15] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng, "MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks," in *Proc. Int. Conf. Artif. Neural Netw. (ICANN)*, 2019, pp. 703–716.
- [16] Meegle Editorial, "Anomaly Detection With Kubernetes," Meegle Technical Reference, 2025. [Online]. Available: <https://www.meegle.com/>
- [17] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," Google Inc., Mountain View, CA, USA, White Paper, 2011, rev. 2014.
- [18] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.
- [19] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
- [20] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining (KDD)*, 2016, pp. 785–794.
- [21] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

- [22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All You Need," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2017, pp. 5998–6008.
- [23] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," Linux J., no. 239, Art. 2, 2014.
- [24] E. Casalicchio, "A study on performance measures for auto-scaling CPU-intensive containerized applications," Cluster Comput., vol. 22, pp. 995–1006, 2019, doi: 10.1007/s10586-019-02997-z.
- [25] Kubernetes Documentation, "Kubernetes," The Kubernetes Authors, 2022. [Online]. Available: <https://kubernetes.io/docs/home/>
- [26] M. Nita and F. Pop, "Failure prediction using machine learning in a virtualised HPC system and application," Cluster Comput., vol. 23, pp. 2043–2061, 2020, doi: 10.1007/s10586-019-02917-1.
- [27] M. Manivannan and D. Manivannan, "A Deep Dive into the Google Cluster Workload Traces: Analyzing the Application Failure Characteristics and User Behaviors," preprint, ResearchGate, 2023.
- [28] M. Gollapalli et al., "Task Failure Prediction Using Machine Learning Techniques in the Google Cluster Trace Cloud Computing Environment," Math. Model. Eng. Probl., vol. 9, no. 2, 2022, doi: 10.18280/mmep.090234.
- [29] D. N. Ng'ang'a, W. K. Cheruiyot, and D. Njagi, "A Machine Learning Framework for Predicting Failures in Cloud Data Centers – A Case of Google Cluster, Azure Clouds and Alibaba Clouds," SSRN Electron. J., 2023, doi: 10.2139/ssrn.4404569.