

# *Anomaly Detection in Software System Logs Using Machine Learning*

**Jirong Han**

*College of Arts and Sciences, University of Washington, Seattle, USA*  
*hjr23679@uw.edu*

**Abstract.** Modern software systems generate large volumes of logs that record events, states, errors, and user or component activities. Manual inspection is increasingly unable to support timely fault diagnosis, performance monitoring, and security analysis. This review examines how machine learning is used to detect anomalies in software system logs, with emphasis on the full pipeline from raw log messages to operational alerts. It first explains software logs, event templates, anomaly types, labels, and commonly used evaluation measures. It then compares log parsing, window construction, count-based features, semantic embeddings, classical machine learning, deep sequence models, self-supervised learning, and parser-free approaches. Recent journal studies show that model accuracy depends not only on architecture but also on parsing distinguishability, feature representation, window design, label quality, dataset imbalance, and cross-system differences. Traditional models can remain competitive when features and tuning are appropriate, while transformer, contrastive, and end-to-end methods improve semantic modeling and adaptation to unseen events. However, data drift, limited labels, inconsistent benchmarks, false alarms, interpretability, privacy, and deployment cost remain important barriers. Future work should prioritize adaptive, transferable, explainable, and privacy-aware systems that combine accurate detection with practical maintenance needs.

**Keywords:** log anomaly detection, software logs, machine learning, log parsing, system monitoring

## **1. Introduction**

Web services, cloud platforms, databases, distributed applications, and Internet of Things systems continuously produce logs while they run. A log message may record a timestamp, a component name, an operation, a severity level, an error code, or a short description written by a developer. These records are among the most direct observations of software behavior after deployment. They support troubleshooting, reliability analysis, performance monitoring, security auditing, and incident response. The practical difficulty is scale. A busy service may generate millions of messages in a short period, and meaningful abnormal events may be hidden among repetitive normal activity. Manual keyword searches and fixed rules are still useful for known failures, but they cannot easily keep pace with changing software, unfamiliar faults, or long event sequences. Machine learning-based log anomaly detection addresses this problem by learning patterns associated with normal or

abnormal behavior and then identifying observations that do not follow the learned patterns. A recent survey shows that the field now covers preprocessing, representation learning, sequence modeling, threshold selection, and operational evaluation rather than a single classification step [1].

The central challenge is that log data are not naturally ready for machine learning. Headers are often structured, but the message body is semi-structured text that mixes stable words with changing values such as addresses, file paths, identifiers, durations, and counts. Most pipelines therefore parse raw messages into templates, group events into sessions or time windows, and convert the resulting units into numerical features. These preprocessing choices determine what information is preserved for the detector and can therefore affect downstream anomaly-detection performance. An empirical study comparing six representation techniques, seven learning models, and four public datasets found that the representation and aggregation method can substantially affect downstream performance [2]. Parsing quality also requires careful interpretation. Khan et al. found no strong general correlation between commonly used parsing-accuracy measures and anomaly-detection accuracy; instead, the ability of parsed templates to keep normal and abnormal behaviors distinguishable was more important [3]. These findings show that an accurate-looking intermediate component does not necessarily produce an accurate end-to-end detector.

Research has moved from keyword rules, statistical thresholds, principal component analysis (PCA), and shallow classifiers toward models that learn event semantics and order. LayerLog represents words, log events, and log sequences in a hierarchy so that local meaning and sequence context can be modeled together [4]. ContextLog avoids a separate parsing stage and preserves raw-message information while strengthening contextual representation [5]. LogEncoder applies contrastive representation learning so that normal and anomalous sequences become more separable in the learned space [6]. A semi-supervised bidirectional temporal convolution network models contextual dependencies in both directions while reducing reliance on fully labeled data [7]. At the same time, the growth of deep learning does not make classical methods irrelevant. An evaluation of an improved PCA pipeline reported that a lightweight, semantically informed traditional method could approach the effectiveness of more complex methods while using fewer resources and remaining stable with limited training data [8]. The appropriate method therefore depends on the data, supervision, latency, and maintenance setting rather than on model complexity alone.

This paper reviews anomaly detection in software system logs from a data-science perspective. Its goals are to explain the basic concepts of log data and anomaly types, describe how raw messages are parsed and represented, compare classical and deep learning methods, summarize representative journal studies from the past three years, and identify limitations that affect practical use. The discussion connects software operations with text processing, event sequences, feature representations, classification, and anomaly-detection metrics. Because this is a literature review, it does not reproduce a model or claim a new experimental result. Instead, it focuses on how methodological choices influence reported findings and on what is still needed for reliable, efficient, and trustworthy deployment.

## 2. Basic concepts of software logs and anomaly detection

A software log is a sequence of records produced by programs, operating systems, middleware, servers, or network devices during execution. A typical record contains a structured header and a less structured message. The header may include a timestamp, host, process or thread identifier, component, and severity level such as INFO, WARN, or ERROR. The message describes an event, but often combines fixed text with variable parameters. For example, two messages that report a failed connection may contain the same words but different Internet Protocol addresses and port

numbers. A log parser replaces or groups variable fields and assigns messages to event templates. A template is therefore an abstraction of a family of similar messages, while the original parameters may still contain important evidence. Parsing reduces the size of the event vocabulary and makes counting or sequence analysis possible, but excessive abstraction can remove values that reveal an anomaly. The distinction between stable tokens and parameters is thus a modeling decision, not a purely mechanical cleaning step [2, 3].

An anomaly is an observation or pattern that differs from the expected behavior of the system. A point or single-event anomaly occurs when one message is unusual by itself, such as an unknown error template or an impossible parameter value. A frequency anomaly occurs when a familiar event appears too often or too rarely within a given period. A sequential anomaly occurs when normal events appear in an unexpected order, an expected event is missing, or a transition is unlikely. A contextual anomaly may be normal in one context but abnormal for a specific component, host, time, or workload. Finally, a collective anomaly is visible only when several records are considered together. These categories overlap. A repeated login failure, for example, can be normal as a single event but abnormal as a high-frequency sequence. This is why window construction and contextual representation strongly affect what a detector can recognize [1, 4].

Web access logs, application logs, server logs, and distributed-system logs differ in both content and purpose. Web logs emphasize requests, status codes, resources, response sizes, and client information. Application logs contain developer-defined events that may change when source code changes. Server and operating-system logs record resource, process, hardware, and service states. Distributed systems add cross-node timing and interleaving, which makes it difficult to reconstruct one request or failure path. Public research often uses the Hadoop Distributed File System, Blue Gene/L, Thunderbird, Spirit, or Hadoop log datasets because they provide established benchmarks, but their event distributions, labels, and units of analysis differ. Hadoop Distributed File System logs are commonly grouped by block identifiers, whereas supercomputer logs are often divided by fixed windows. A performance number from one dataset is therefore not automatically comparable with a number from another.

Learning settings are usually described as supervised, semi-supervised, or unsupervised. Supervised methods use labeled normal and anomalous examples. Semi-supervised methods combine labeled and unlabeled data, while one-class methods are often trained primarily on normal examples and detect deviations from learned normal behavior. Terminology varies across log-anomaly studies, so the supervision setting should be stated explicitly, especially when a method is trained only on normal data. Real datasets are highly imbalanced because failures are rare, and labels may describe an entire session rather than the exact responsible event. Training, validation, and test sets should also respect time or system boundaries; otherwise, similar messages can leak across splits and make results too optimistic. These basic definitions are necessary for interpreting model comparisons.

### 3. Data processing and machine learning methods for log anomaly detection

A typical pipeline begins with collection and normalization. Duplicated records may be removed when they are produced by the collection process, and timestamps, severity fields, and component names are converted to consistent formats. Raw messages may then be parsed into templates using tree-based, clustering, or similarity-based parsers. The records are grouped into fixed time windows, sliding windows, sessions, or identifier-based sequences. A fixed window is simple and supports online processing, but it may split one failure across two windows or combine unrelated requests. A sliding window preserves more local context but creates overlapping samples and increases

computation. Session windows can better represent a complete transaction when a reliable identifier exists. The output can be an event-count vector, a transition or sequence, a term-frequency vector, or an embedding learned from the text. Empirical evidence indicates that representation, parsing, and aggregation should be evaluated together rather than selected independently [2, 3].

Classical machine learning methods remain useful because they are relatively fast, interpretable, and available in mature software libraries. Clustering methods group similar event-count or text vectors and may flag samples that lie far from dense clusters or cluster centers as anomalous. PCA models the dominant variation in normal count vectors; a large reconstruction residual can indicate behavior outside the normal subspace. Isolation Forest uses random partitions and tends to isolate rare observations with fewer splits. One-Class Support Vector Machines learn a boundary around normal examples, while logistic regression, decision trees, and random forests can be trained when labeled anomalies are available. Their main limitations are dependence on hand-designed aggregation, difficulty capturing long order relationships, and sensitivity to scale, class weights, thresholds, and hyperparameters. Nevertheless, improved PCA results show that semantic features and careful tuning can make a simple model competitive [8]. This supports selecting classical and deep models according to dataset characteristics and operational constraints, rather than assuming one family is always superior.

Deep learning methods reduce some manual feature engineering by learning representations from sequences or text. Recurrent neural networks and long short-term memory networks model relationships among ordered events and can flag events that are unlikely given the recent event history. Convolutional networks detect local patterns in event or character sequences and can be efficient in parallel computation. Attention and transformer models capture relationships across a wider context and support pre-training on large unlabeled log collections. Hierarchical models encode information at several levels, such as tokens, messages, and windows [4], while contextual raw-log models can retain information removed by parsing [5]. Contrastive learning constructs an embedding space in which positive pairs, often different views of related samples, are pulled closer while negative pairs are pushed apart [6]. Semi-supervised bidirectional temporal convolution can learn context from both directions while reducing the need for complete labels [7]. These models can capture semantics and order more directly than event counts, but they require more training resources and may be difficult to interpret or update.

Recent pipelines also question whether a separate parser is always necessary. OneLog uses a character-based end-to-end neural model that reduces dependence on separate parsing and vectorization stages while preserving digits and punctuation that template pipelines may discard [9]. LogFiT fine-tunes a pre-trained language model on normal logs with self-supervised objectives and does not require a fixed vocabulary of event templates [10]. Parser-free methods can reduce dependence on template parsing and may avoid cascading errors caused by parser mistakes, although their performance on unseen messages still requires empirical validation. However, they process longer raw inputs, may require more computation, and can learn accidental correlations from identifiers or formatting. A practical compromise is to retain raw and parsed views, using the template for stable event structure and selected parameters for values that carry operational meaning.

Evaluation usually reports precision, recall, and F1-score. Precision asks what proportion of alerts are truly anomalous, while recall asks what proportion of known anomalies are detected. F1-score balances the two, but it does not show all operational trade-offs. False positive rate is important because even a small percentage can create thousands of alerts in a high-volume system. Detection delay matters for online monitoring, and training time, prediction time, memory use, and retraining cost affect deployment. Results must also state whether metrics are computed per event, window,

session, or incident. Thresholds should be selected on validation data instead of the test set, and time-aware splits are preferable when the intended use is future detection. Without these controls, an apparently high F1-score may reflect data preparation rather than robust anomaly recognition.

#### 4. Recent studies of machine learning-based log anomaly detection

Recent journal studies illustrate a shift from isolated model proposals toward representation, transfer, and end-to-end evaluation. LayerLog uses a word-log-sequence hierarchy and emphasizes that a log is neither only plain text nor only an event identifier [4]. ContextLog instead processes logs without a separate parser, preserving character and non-character information while strengthening contextual representation [5]. LogEncoder uses contrastive learning to improve separability in the representation space [6]. These studies share the goal of extracting more meaning than event-count vectors, but their mechanisms differ. Hierarchy explicitly organizes information levels, parser-free modeling retains raw evidence, and contrastive learning shapes distances among samples. Their effectiveness therefore depends on tokenization, the available corpus, the definition of positive and negative pairs, and how windows preserve the fault context.

Learning under limited supervision is another major research direction. Yin et al. propose a semi-supervised detector based on bidirectional temporal convolution, allowing contextual information to be captured from both directions in an event sequence [7]. This direction is important because abnormal logs are rare and expert annotation is expensive. Yet semi-supervised learning introduces a risk: incorrect pseudo-labels or incorrect assumptions about normal samples can be reinforced, especially when anomalies are diverse or the normal distribution changes. Evaluation should therefore report label selection, class balance, and sensitivity to the labeled fraction, not only the final score.

The continued value of traditional machine learning is demonstrated by the improved PCA study. By combining PCA with a lightweight semantic representation and carefully designed settings, Yang et al. found that a simpler unsupervised approach could be competitive with advanced supervised or semi-supervised deep models while remaining efficient and stable when training data were limited [8]. This result does not imply that PCA is universally superior. It means that conclusions about model families are sensitive to feature engineering, tuning, data volume, and evaluation design. It also supports a practical baseline rule: complex models should be compared against well-tuned classical methods rather than weak default configurations.

Parser-free and end-to-end studies respond to unstable templates and unseen messages. OneLog applies a character-based hierarchical convolutional network directly to raw messages or sequences and merges the parser, vectorizer, and classifier. Its experiments across message- and sequence-based datasets suggest that an end-to-end model can learn useful representations directly from raw logs and can benefit from multi-project training [9]. LogFiT takes a language-model approach, fine-tuning a pre-trained transformer through self-supervised learning on normal log data [10]. The two methods preserve raw text but make different trade-offs. Character models have small vocabularies and retain punctuation and numerical form, whereas transformer language models offer rich contextual semantics but typically require more memory and careful tokenization. Both reduce dependence on manually selected templates, although neither removes the need for normalization, windowing, and threshold selection.

Cross-system generalization has also become more central. LogETA adds time-interval information and optimizes inter-class boundaries for transfer to another system, addressing the fact that identical event orders may have different timing meanings [11]. LogGT represents log behavior with heterogeneous graph features and uses transfer learning to improve cross-system anomaly

detection [12]. These methods recognize that a model trained on one service should not be expected to work unchanged on another. They also raise difficult questions about what is transferable: words, event relationships, timing, labels, or decision boundaries. A method may show strong transfer between related public datasets but still fail when production components, logging practices, or fault distributions are different.

Broader empirical work provides a useful counterweight to individual model results. Yu et al. compared classical and deep machine learning techniques across five public log anomaly detection datasets. Their study found that simple classical methods can outperform deep learning methods in both accuracy and time efficiency under the evaluated settings [13]. This conclusion helps explain why papers can report conflicting results even when they use familiar benchmarks. It also suggests that future evaluations should publish complete preprocessing choices and resource costs, not only the highest F1-score.

Two 2025 studies extend the discussion toward realistic adaptation. Ladle targets a setting with many log types produced by different components. It embeds overlapping log segments with a sentence transformer, compares them with type-specific reference data, and recenters anomaly scores so that variation levels are comparable across log types. New reference records can be added to adapt to data drift without retraining the language model [14]. LogSentry combines Bidirectional Encoder Representations from Transformers with contrastive learning and a retrieval-augmented k-nearest-neighbor component, joining a learned classifier with similarity to stored examples [15]. Both approaches separate general language representation from a local reference or retrieval mechanism. This design may be valuable in changing systems because local evidence can be updated more easily than a large neural model. However, Ladle's evaluation used injected anomalies when sufficient real failures were unavailable, and retrieval-based methods depend on the quality and privacy of stored examples.

Taken together, recent studies show progress in semantic representation, reduced dependence on labeled data, parser-free processing, cross-system learning, and adaptation. They do not establish a single universally superior architecture. Public datasets differ in anomaly ratio, message stability, label granularity, and sequence construction, while industrial logs add confidential fields, component heterogeneity, software updates, and limited fault labels. The strongest conclusion is therefore methodological: an anomaly detector is a pipeline, and its result reflects the interaction among data source, parsing or tokenization, windows, representation, model, threshold, and metric.

## 5. Limitations and prospects

The first limitation is instability in log format and meaning. Developers modify logging statements, components are upgraded, identifiers change, and new services appear. A parser trained on an earlier version may split one event into several templates or merge events that should remain distinct. Even when a parser scores highly against a template benchmark, it may remove precisely the distinction required by the anomaly detector [3]. Raw-text models avoid template identifiers but are not immune to drift because their token distributions and normal contexts also change. Future systems need monitoring for representation drift, controlled updates, and rollback when an update causes a true fault to be treated as normal. Incremental reference stores, as used by Ladle, offer one practical direction because they can incorporate a new normal state without full neural retraining [14].

A second limitation is the scarcity and quality of labels. Many logs are labeled only at the session or machine level, so the exact event responsible for a failure is unknown. Labels may be produced from tickets, alerts, or injected faults and can contain delays or mistakes. Severe class imbalance encourages a detector to favor normal behavior. Semi-supervised and self-supervised learning

reduce annotation needs [7, 10], but they also rely on assumptions about normal data and augmentations. If the normal training set contains hidden failures, the model may learn to accept them. If pseudo-labels are wrong, the model may amplify the error. Future research should report uncertainty, use active learning to prioritize informative cases for expert labeling, and distinguish unknown anomalies from simple low-confidence predictions.

Window design and evaluation create a third set of problems. A window that is too short loses context; one that is too long delays alerts and mixes separate processes. Fixed, sliding, session, and identifier-based windows can produce different sample counts and different levels of class imbalance from the same raw file. Random train-test splits may place nearly identical neighboring windows in both sets. Metrics can also hide operational costs. F1-score gives equal weight to precision and recall, but an operations team may care more about avoiding missed critical failures or limiting daily false alerts. The empirical comparison by Yu et al. demonstrates how dataset characteristics, preprocessing choices, and computational cost can affect conclusions [13]. Future studies should use time-based and cross-version validation, report event- and incident-level metrics, measure delay, and publish the threshold-selection process.

Generalization across systems remains limited. Event identifiers have no shared meaning across projects, and natural-language similarity does not guarantee operational similarity. Transfer methods such as LogETA and LogGT model timing, relationships, or boundaries, but they still require a credible connection between source and target data. A model trained on a supercomputer may not recognize anomalies in a web application whose failures are expressed through response codes and traces. Cross-system research should separate transfer without target labels, transfer with a small target sample, and full target retraining. Multi-source monitoring could also combine logs with metrics, traces, topology, and change records, reducing the pressure on text alone to explain the system state.

Computational cost and interpretability affect whether a model can move from a research prototype to an operational production environment. Large transformer models can improve representation quality but may increase training cost, inference latency, energy use, and dependence on specialized hardware.

A high anomaly score without a reason does not tell an engineer which component failed or which event transition was important. Lightweight baselines and the improved PCA result show that efficiency should be treated as a performance dimension [8]. Character-based OneLog suggests that end-to-end representations can preserve information useful for parsing and anomaly detection, but these internal representations still require separate explanation and validation [9]. Future detectors should return supporting events, influential tokens or parameters, similar historical incidents, and calibrated confidence. Retrieval-augmented methods can support this goal by presenting comparable stored cases, provided that retrieval does not simply repeat incorrect or outdated labels [15].

Privacy and security are often underemphasized. Logs may contain user names, addresses, request paths, query content, tokens, device identifiers, or internal network details. Centralizing logs and retaining examples for retrieval can increase the damage of a data leak. Aggressive masking protects privacy but may remove the parameter pattern that signals an attack. Future work should examine privacy-preserving tokenization, access control, retention limits, encrypted processing, and federated or distributed learning. Models must also be protected from log injection and poisoning, because an attacker may create repeated malicious messages that are gradually learned as normal. These issues connect anomaly detection with governance and secure software operations, not only predictive accuracy.

A promising future direction is an adaptive hybrid pipeline. Self-supervised representation learning can use large collections of unlabeled normal data, while a lightweight statistical or retrieval layer can be updated quickly for a particular component. Parsed templates can describe stable structure, and selected raw parameters can preserve numerical and textual evidence. Online drift monitoring can decide when the reference set, threshold, or model should be updated. Cross-system pre-training can provide a starting representation, but validation on target-system data should precede deployment. Finally, explanations and human feedback should be built into the detection loop. Progress should be measured by timely and actionable incident support, not only by a benchmark F1-score.

## 6. Conclusion

This paper reviewed machine learning-based anomaly detection in software system logs. Raw logs contain structured headers, semi-structured messages, changing parameters, and event relationships that must be converted into useful model inputs. The review compared parsing, window construction, count features, semantic embeddings, classical machine learning, sequence models, transformers, contrastive learning, semi-supervised learning, and parser-free methods. Recent studies demonstrate advances in semantic representation, reduced dependence on labels, cross-system transfer, and adaptation to multiple log types. They also show that complex deep models are not automatically superior to well-designed classical baselines. Parsing distinguishability, feature choice, window length, class balance, labels, thresholds, and evaluation splits can be as important as model architecture. Taken together, the reviewed evidence indicates that reliable log anomaly detection depends on the entire processing pipeline rather than on the detector alone. Parsing and tokenization determine which details remain available, window construction defines the context presented to the model, and threshold selection affects the balance between missed anomalies and false alarms. Classical methods remain useful when computational efficiency, limited labels, and straightforward interpretation are priorities, whereas deep, contrastive, self-supervised, and parser-free methods offer greater capacity to learn semantic and sequential patterns. Cross-system transfer and retrieval-based adaptation provide additional options for changing environments, but their effectiveness depends on differences in logging practices, data drift, and the quality of target or reference data. Practical evaluation should therefore consider time-aware or cross-version splits, label quality, detection delay, resource cost, and operational interpretability in addition to conventional accuracy metrics. Privacy and security also require attention because logs may contain sensitive identifiers and may themselves be affected by poisoning or malicious injection. Future systems should combine self-supervised learning, online updates, cross-system adaptation, retrieval, and interpretable evidence while protecting sensitive log data. Understanding these methodological choices helps connect software operational data with data-science methods and supports more reliable alerts and maintenance decisions.

## References

- [1] Landauer, M., Onder, S., Skopik, F., & Wurzenberger, M. (2023). Deep learning for anomaly detection in log data: A survey. *Machine Learning with Applications*, 12, 100470. <https://doi.org/10.1016/j.mlwa.2023.100470>
- [2] Wu, X., Li, H., & Khomh, F. (2023). On the effectiveness of log representation for log-based anomaly detection. *Empirical Software Engineering*, 28(6), Article 137. <https://doi.org/10.1007/s10664-023-10364-1>
- [3] Khan, Z. A., Shin, D., Bianculli, D., & Briand, L. C. (2024). Impact of log parsing on deep learning-based anomaly detection. *Empirical Software Engineering*, 29(6), Article 139. <https://doi.org/10.1007/s10664-024-10533-w>

- [4] Zhang, C., Wang, X., Zhang, H., Zhang, J., Zhang, H., Liu, C., & Han, P. (2023). LayerLog: Log sequence anomaly detection based on hierarchical semantics. *Applied Soft Computing*, 132, 109860. <https://doi.org/10.1016/j.asoc.2022.109860>
- [5] Xiao, R., Li, W., Lu, J., & Jin, S. (2024). ContexLog: Non-parsing log anomaly detection with all information preservation and enhanced contextual representation. *IEEE Transactions on Network and Service Management*, 21(4), 4750-4762. <https://doi.org/10.1109/TNSM.2024.3400283>
- [6] Qi, J., Luan, Z., Huang, S., Fung, C., Yang, H., Li, H., Zhu, D., & Qian, D. (2023). LogEncoder: Log-based contrastive representation learning for anomaly detection. *IEEE Transactions on Network and Service Management*, 20(2), 1378-1391. <https://doi.org/10.1109/TNSM.2023.3239522>
- [7] Yin, Z., Kong, X., & Yin, C. (2024). Semi-supervised log anomaly detection based on bidirectional temporal convolution network. *Computers & Security*, 140, 103808. <https://doi.org/10.1016/j.cose.2024.103808>
- [8] Yang, L., Chen, J., Gao, S., Gong, Z., Zhang, H., Kang, Y., & Li, H. (2024). Try with simpler - An evaluation of improved principal component analysis in log-based anomaly detection. *ACM Transactions on Software Engineering and Methodology*, 33(5), Article 115, 1-27. <https://doi.org/10.1145/3644386>
- [9] Hashemi, S., & Mäntylä, M. (2024). OneLog: Towards end-to-end software log anomaly detection. *Automated Software Engineering*, 31(2), Article 37. <https://doi.org/10.1007/s10515-024-00428-x>
- [10] Almodovar, C., Sabrina, F., Karimi, S., & Azad, S. (2024). LogFiT: Log anomaly detection using fine-tuned language models. *IEEE Transactions on Network and Service Management*, 21(2), 1715-1723. <https://doi.org/10.1109/TNSM.2024.3358730>
- [11] Gong, K., Luo, S., Pan, L., Zhang, L., Zhang, Y., & Yu, H. (2024). LogETA: Time-aware cross-system log-based anomaly detection with inter-class boundary optimization. *Future Generation Computer Systems*, 157, 16-28. <https://doi.org/10.1016/j.future.2024.03.028>
- [12] Wang, P., Zhang, X., Cao, Z., Xu, W., & Li, W. (2024). LogGT: Cross-system log anomaly detection via heterogeneous graph feature and transfer learning. *Expert Systems with Applications*, 251, 124082. <https://doi.org/10.1016/j.eswa.2024.124082>
- [13] Yu, B., Yao, J., Fu, Q., Zhong, Z., Xie, H., Wu, Y., Ma, Y., & He, P. (2024). Deep learning or classical machine learning? An empirical study on log-based anomaly detection. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)* (pp. 403-415). Association for Computing Machinery. <https://doi.org/10.1145/3597503.3623308>
- [14] Mylläri, J., Aalto, T., & Nurminen, J. K. (2025). Ladle: A method for unsupervised anomaly detection across log types. *Automated Software Engineering*, 32(2), Article 34. <https://doi.org/10.1007/s10515-025-00504-w>
- [15] Li, W., Wu, Y., Huang, W., Zhou, F., Ou, W., Wang, H., & Deng, L. (2025). System log anomaly detection based on contrastive learning and retrieval augmented. *Scientific Reports*, 15, Article 38370. <https://doi.org/10.1038/s41598-025-22208-7>