2025 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C) | 979-8-3315-0215-7/25/$31.00 ©2025 IEEE | DOI: 10.1109/ACSOS-C66519.2025.00066

FedLAD: A Modular and Adaptive Testbed for Federated Log Anomaly Detection

Yihan Liao[1], Jacky Keung[1], Zhenyu Mao[1,*], Jingyu Zhang[1] and Jialong Li[2]
[1]City University of Hong Kong, Hong Kong, China
[2]Waseda University, Tokyo, Japan
yihanliao4-c@my.cityu.edu.hk, jacky.keung@cityu.edu.hk, jzhang2297-c@my.cityu.edu.hk, lijialong@fuji.waseda.jp
*corresponding author: zhenyumao2-c@my.cityu.edu.hk

***Abstract*—Log-based anomaly detection (LAD) is critical for ensuring the reliability of large-scale distributed systems. However, most existing LAD approaches assume centralized training, which is often impractical due to privacy constraints and the decentralized nature of system logs. While federated learning (FL) offers a promising alternative, there is a lack of dedicated testbeds tailored to the needs of LAD in federated settings. To address this, we present FedLAD, a unified platform for training and evaluating LAD models under FL constraints. FedLAD supports plug-and-play integration of diverse LAD models, benchmark datasets, and aggregation strategies, while offering runtime support for validation logging (self-monitoring), parameter tuning (self-configuration), and adaptive strategy control (self-adaptation). By enabling reproducible and scalable experimentation, FedLAD bridges the gap between FL frameworks and LAD requirements, providing a solid foundation for future research. Project code is publicly available at: https://github.com/AA-cityu/FedLAD.**

***Index Terms*—Federated Learning, Log Anomaly Detection, Testbed, Self-adaptive System**

I. Introduction

Modern large-scale software systems, ranging from distributed cloud platforms to edge computing infrastructures, generate massive volumes of system logs that record detailed runtime behaviors [1]. These logs are critical for diagnosing failures, monitoring performance, and ensuring reliability. Given the scale and complexity, manual inspection is infeasible. As a result, **L**og-based **A**nomaly **D**etection (LAD) has become a vital approach for automatically identifying abnormal behaviors and potential faults [2], and it has been successfully applied in diverse domains such as data centers, Internet of Things, and autonomous vehicles [3].

Despite their effectiveness, existing LAD models face several critical limitations. Many assume centralized training with globally accessible logs, which is an impractical assumption due to privacy concerns and organizational data silos. This constraint hinders cross-organizational collaboration, particularly when sensitive operational logs cannot be shared across institutional boundaries [4]. Furthermore, most LAD models are static, struggling to adapt to evolving behaviors or unseen anomalies [1]. These limitations pose serious obstacles to real-world deployment and reproducible research in distributed settings, where logs are diverse, privacy-sensitive, and continuously evolving [5]. To overcome the limitations of centralized LAD, recent research has investigated the use of **F**ederated **L**earning (FL), which enables multiple clients (e.g., data centers, edge nodes) to collaboratively train LAD models without sharing raw log data, thereby preserving data privacy [6]. By combining local training with global model aggregation, FL enables knowledge transfer across heterogeneous sources while maintaining data locality, which is an essential property for LAD scenarios involving geographically distributed logs.

While prior works [7], [8] have proposed FL-based LAD methods, they are typically designed for specific experimental setups with tightly integrated implementations, which limit reproducibility, extensibility, and broader reuse. These efforts often focus on particular LAD models or datasets, making it difficult to experiment with alternative architectures or strategies. By contrast, general-purpose FL frameworks such as Flower [9] provide modular infrastructure for standard domains like image classification. However, they lack native support for structured log data, including log-specific preprocessing or anomaly-centric adaptations. Consequently, there remains a clear gap for a reusable, log-aware FL platform that supports LAD-specific challenges while enabling reproducible and adaptive experimentation. FedLAD fills this gap as the first modular testbed tailored to LAD under FL constraints, integrating pluggable LAD models, log-specific preprocessing, non-independent and identically distributed (IID) simulation, and adaptation driven by detection quality.

Built on a modular architecture, FedLAD decouples key components, including models, datasets, training workflows, and aggregation logic, via standardized interfaces to facilitate extensibility, interoperability, and fair comparison. In addition, FedLAD incorporates a suite of self-managing features, including self-monitoring for logging per-round performance metrics; self-configuration for recommending hyperparameters; and a self-adaptive executor for triggering strategy switching or early stopping. By lowering the barrier to prototyping, automated evaluation, and training adaptation, FedLAD serves as a unified and reliable testbed for accelerating LAD research in FL. Detailed enhancements and contributions include:

- **Novel LAD-specific FL testbed**: FedLAD is the first testbed tailored for LAD under FL, bridging the gap between generic FL frameworks and LAD-specific needs.
- **Modular architecture**: All components, including models, datasets, aggregation strategies, and adaptation policies, are decoupled via standardized interfaces, enabling

979-8-3315-0215-7/25/$31.00 ©2025 IEEE
DOI 10.1109/ACSOS-C66519.2025.00066

flexible deployment into diverse LAD workflows.

- **Experiment extensibility**: FedLAD supports benchmark datasets (BGL [10], HDFS [11], Thunderbird [10]), LAD models (DeepLog [12], LogAnomaly [13], NeuralLog [14]), and aggregation strategies (FedAvg [15], Scaffold [16], FedAdam [17], FedProx [18]) under both IID and non-IID settings [19], while offering unified interfaces for adding new components.
- **Trackable training process via self-monitoring**: Per-round validation metrics are logged to enable continuous performance tracking and data-driven runtime decisions.
- **Automatic training via self-configuration**: An optional auto-configurator suggests initial hyperparameters based on dataset traits, reducing manual tuning effort.
- **Adaptive training via self-adaptation**: A built-in executor monitors runtime feedback and triggers strategy switching or early stopping to improve robustness.

II. Background

A. Log Anomaly Detection

As a foundational technique for preserving software reliability and security, LAD leverages execution logs that encapsulate rich temporal and semantic patterns. These logs serve as a vital source of information for identifying system faults, diagnosing performance bottlenecks, and detecting security violations.

Recent advances in LAD have embraced deep learning to automatically learn complex sequential and contextual representations from raw log data. For example, DeepLog [12] uses LSTMs to model log sequences, LogAnomaly [13] combines semantic and sequential features, and LogBERT [20] leverages pretrained language models for contextual understanding.

Despite growing progress in LAD model development, evaluating these methods in a realistic and reproducible manner remains challenging. Real-world log environments vary widely in format, structure, and semantic patterns, making it difficult to compare models fairly or assess their robustness [1]. These challenges underscore the critical need for a configurable testbed that supports diverse datasets, models, and evaluation setups with consistent monitoring and repeatability.

B. Federated Learning-based Log Anomaly Detection

FL has emerged as a promising paradigm for privacy-preserving machine learning, particularly in domains where data is sensitive and decentralized. In the context of LAD, FL enables multiple clients, such as data centers, edge nodes, or industrial devices, to collaboratively train anomaly detection models without sharing raw log data. This is particularly valuable given the sensitive nature of operational and user logs, and the growing regulatory emphasis on data privacy.

Recent studies have demonstrated the feasibility and potential of applying FL to LAD. For example, FedLog [21] proposes a communication-efficient FL framework tailored for industrial IoT logs, incorporating a masking mechanism to protect sensitive tokens during training. FLOGCNN [22] introduces a lightweight CNN-based model that balances detection performance and model efficiency across distributed clients. These approaches show that FL can successfully support LAD in decentralized environments while maintaining data privacy.

Despite these advancements, several challenges remain. Most existing FL-based LAD solutions are tightly coupled to specific model architectures or datasets, limiting their extensibility. They often prioritize privacy and accuracy metrics, while overlooking key system-level concerns such as runtime adaptivity and client heterogeneity, where clients may differ significantly in log semantics, data volume, or resource availability [**?**]. Moreover, common federated aggregation strategies such as FedAvg, Scaffold, and FedAdam remain underexplored, particularly under non-IID data distributions. Finally, most prior implementations assume static training configurations, reducing the robustness in dynamic environments.

These open challenges call for a unifying testbed that enables flexible experimentation and adapts to dynamic conditions. Therefore, this paper presents FedLAD, a modular, extensible, and self-adaptive testbed that enables scalable and reproducible LAD research within FL.

III. System Design

FedLAD is structured to support configurable, component-wise experimentation across LAD models, datasets, and federated strategies. This section introduces its architectural decomposition and key mechanisms for training orchestration, evaluation logging, and adaptation triggering.

A. FedLAD Architecture

As shown in Fig. 1, FedLAD adopts a modular architecture aligned with self-adaptive system principles. Each module encapsulates a distinct role in the federated LAD lifecycle, while the design separates developer-accessible components from runtime execution and coordination. This enables flexible experimentation with LAD models, strategies, and datasets.

FedLAD is organized into four conceptual regions, **Design-time**, **Managing System**, **Managed (FL) System**, and the **Environment**. Interactions among these layers are governed by well-defined *data flows* (gray) and *control flows* (blue dotted lines), enabling a cohesive testbed for the systematic design and dynamic execution of adaptive FL workflows.

In the design-time region, the user defines both adaptation policies and training configurations via the configurator. The configurator supports manual YAML editing and automated configuration, offering suggestions (e.g., client count) based on dataset characteristics. These settings are then passed to the FedLAD API, which serves as a unified coordination layer between developer-defined inputs and runtime modules. Specifically, adaptation policies are routed to the Managing System, while training configurations are directed to the Environment (i.e., the client-side runtime). As a testbed, FedLAD enables the rapid deployment of adaptation policies and flexible reconfiguration of federated environments, supporting seamless integration of new models, aggregation rules, or adaptation strategies. This design facilitates realistic and controllable experimentation across diverse FL scenarios.

The Managing System implements the core Monitor–Analyzer–Planner–Executor (MAPE) feedback loop that drives self-adaptation. The Monitor collects training states, such as loss and F1-score, from the Managed System. In FedLAD's implementation, the Analyzer and Planner functionalities are integrated into the Executor module. This unified Executor detects performance trends (e.g., validation plateaus), evaluates adaptation conditions (e.g., early stop or strategy switch), and enforces control decisions back to the training system. By consolidating adaptation logic into a single module, FedLAD simplifies runtime coordination while maintaining explainability and extensibility, enabling rapid prototyping of adaptive strategies without modifying training components.

The Managed System includes all components necessary for training. The data splitter simulates realistic data heterogeneity across clients, supporting both IID and non-IID partitioning strategies. The aggregator receives model updates from clients and performs aggregation using FedAvg, FedProx, Scaffold, or FedAdam. After aggregation, the Global Model is broadcast back to clients. The Logger captures round-level metrics such as loss and F1-score, and records adaptation events triggered by the Executor. It streams this information to the Monitor as the entry point of the MAPE loop, which processes the data through analysis and planning stages before the Executor triggers adaptive actions.

The Environment represents the client-side runtime, where simulated clients operate on partitioned datasets and perform local training independently. Each client maintains its local model, trains on private data, and uploads updates to the aggregator. Clients also respond to adaptation signals (i.e., early stop and strategy change instructions) broadcast from the Executor. This separation allows the Environment to be flexibly reconfigured to mimic real-world federated scenarios.

In summary, FedLAD's architecture follows self-adaptive system principles: the Managing System (MAPE loop) observes and steers the Managed System (server-side coordination) and interacts with the Environment (clients) via a defined Coordination Interface (FedLAD API). Beyond runtime adaptation, FedLAD supports design-time configurability through a developer-facing configurator, allowing researchers to flexibly define training parameters, client settings, and adaptation logic. This design enables fast prototyping and controlled experimentation across diverse client environments. All data and control flows are explicitly annotated to ensure traceability and reproducibility. Implementation details are in Section IV.

B. System Workflow

FedLAD adopts a synchronous FL workflow that proceeds in communication rounds, with each round comprising a complete cycle of local training, aggregation, adaptation, and synchronization. This procedural flow, depicted in Fig.2, illustrates how the Managed System coordinates with the Environment across rounds.

The process begins with ① where the configurator parses experiment settings from a YAML configuration file, including

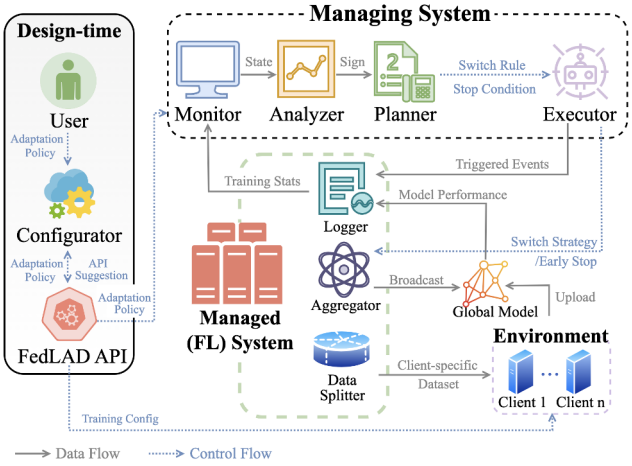


Fig. 1: Overview of the architecture of FedLAD.

dataset, model, number of clients, and aggregation strategy. These configurations are dispatched to both server and client components via the FedLAD API. In ②, each simulated client loads its assigned data partition and initializes its corresponding LAD model. Clients then perform local training for a configurable number of epochs using standard optimizers, independently updating their models based on local log data. After training, in ③, clients upload their updated model parameters to the server. These updates capture each client's individualized view of system behavior, which is then used to inform global model aggregation and further adaptation logic.

Upon receiving updates, during ④ the server performs global aggregation using the selected strategy (e.g., FedAvg, FedProx, Scaffold). During this phase, the Logger captures key performance metrics for each round, such as global validation loss and F1-score. In ⑤, these metrics are passed to the managing system, which monitors training dynamics and determines whether adaptation should be triggered. If predefined criteria are met, such as stagnation in performance over T rounds or a significant drop in accuracy, the Executor may switch aggregation strategies or invoke early stopping to improve training efficiency. Finally, in ⑥, the aggregated global model is broadcast back to all clients, who update their local models accordingly. This completes a full training round, after which the workflow advances to the next iteration.

To summarize, the complete training cycle empowers FedLAD to emulate realistic federated LAD scenarios with dynamic adaptation, offering a flexible and reproducible platform for experimentation across diverse configurations.

IV. Implementation

A. Deployment Environment

We tested FedLAD on a server equipped with 4× NVIDIA RTX 3090 GPUs (24GB each), running Ubuntu 20.04 and CUDA 11.4. The framework supports Python 3.9 and 3.10. For reproducibility, we recommend a Linux environment with at least 8GB RAM and an NVIDIA GPU (CUDA 11+) for

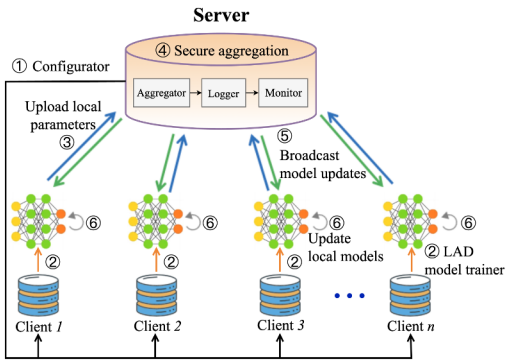


Fig. 2: Federated learning workflow in FedLAD.

accelerated training. FedLAD supports both single-machine and multi-machine execution through process-based client simulation, with optional GPU acceleration for local training. All components are container-ready, facilitating efficient deployment and end-to-end experimentation.

B. Implementation Details

1) Configuration Management: FedLAD relies on YAML-based configuration files to specify experiment parameters, such as client count, dataset, model, training epochs, learning rate, aggregation method, and adaptation policy. These files are parsed at startup by the configurator, which distributes the settings across system components. FedLAD also includes an optional auto-configurator that inspects dataset characteristics to suggest initial hyperparameters, reducing manual tuning.

2) Dataset and Model Handling: FedLAD provides built-in support for benchmark log datasets, including BGL, HDFS, and Thunderbird. Raw log files are preprocessed into sequences of event templates using a sliding window mechanism. For datasets with external anomaly annotations (e.g., HDFS), label alignment is handled automatically. Each client receives a distinct data partition based on the configured IID or non-IID strategy. LAD models are encapsulated as plug-and-play components following a unified interface. This interface defines standard functions for forward propagation, loss computation, and parameter updates, enabling consistent training and aggregation across models.

3) Federated Training Engine: The training engine coordinates server–client communication and implements the full FL protocol. Clients train models locally and return state dictionaries, which are aggregated on the server side. The engine supports per-client optimizer configurations and simulates heterogeneous local resource conditions. All training operations are encapsulated for reusability and extensibility, allowing users to experiment with custom aggregation schemes. It leverages PyTorch's modular training loop to manage local model execution and gradient handling. Server-side aggregation is implemented as a strategy-agnostic class, enabling dynamic hot-swapping of optimization methods during runtime.

4) Aggregation Strategy Implementation: FedLAD encapsulates aggregation strategies as pluggable components that adhere to a unified interface. Each strategy is implemented as an independent Python class defining core aggregation methods. During training, the server dynamically loads the selected strategy based on configuration files or adaptation signals from the Executor. Strategy selection can be performed statically before training begins or dynamically at runtime, allowing FedLAD to adapt to evolving training conditions. To support extensibility, new strategies can be integrated by subclassing the base aggregator interface and registering them through the configuration system. This decoupled design enables researchers to prototype custom aggregation methods within a consistent framework, while FedLAD internally tracks runtime metadata, such as client weights and gradient norms, to support advanced strategies requiring this information.

5) Executor and Adaptation Logic: FedLAD's Executor enables runtime adaptivity through a monitor–trigger–actuate control loop. After each aggregation round, the Logger records validation metrics (e.g., loss, F1-score), which are passed to the Executor for evaluation. The Executor checks whether predefined conditions are met, such as performance stagnation over T rounds or F1-score degradation beyond a threshold δ, and triggers appropriate adaptation actions. These actions include switching aggregation strategies (e.g., from FedAvg to Scaffold) or invoking early stopping. Internally, the Executor operates as a finite state machine that tracks training progress and transitions, ensuring traceable and explainable behavior.

6) Logging and Reproducibility: FedLAD ensures reproducibility by enforcing fixed random seeds across all components and logging comprehensive experiment metadata. Recorded logs include per-round metrics, model checkpoints, adaptation events, and client-specific training statistics. Results are saved in structured formats (CSV, JSON, PNG) and exported automatically for visualization and analysis.

Together, these components make FedLAD a modular, extensible, and reproducible testbed for FL-based LAD, with extension instructions for models, datasets, and strategies provided in the artifact guide.

V. Experiments

A. Experiment Workflow

To demonstrate how developers can use FedLAD to evaluate their own LAD models or federated training strategies, this section presents a full experiment workflow composed of two stages: prepare and deploy, and execute, as illustrated in Fig. 3. The overall lifecycle enables pluggable customization and adaptive evaluation in FL.

Preparation and Deployment. The user starts by registering their LAD model or aggregation strategy into the corresponding registry. FedLAD provides standardized interfaces for custom extensions, allowing researchers to evaluate new algorithmic designs without modifying the underlying infrastructure. After implementation, users may define manually configuration parameters or invoke the built-in assistant script, which automatically recommends dataset-specific settings. These settings are passed to the Configurator and finalized into a YAML file. The system then initializes the full pipeline,

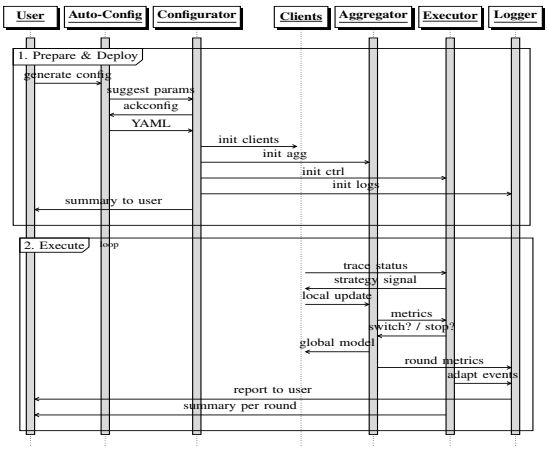


Fig. 3: Two-stage lifecycle in FedLAD.

including client simulators, the Aggregator, the Executor for adaptive logic, and the logger for tracking performance. A summary is returned for user validation. The deployment phase completes with all clients assigned data partitions (IID or non-IID) and a training environment for execution.

Execution. In the execution stage, each round proceeds by simulating local training at each client. Clients report runtime status to the Executor, which evaluates current progress and may issue adaptive instructions such as switching strategies or triggering early stopping. Local model updates are sent to the Aggregator, which performs federated aggregation (e.g., FedAvg or a user-defined strategy) and broadcasts the global model back to clients. Round-wise metrics, including validation loss and F1-score, are logged in real-time. Throughout the process, the Executor continuously evaluates whether adaptation criteria are met, such as plateaued performance, and triggers corresponding events. Logs of all metrics, events, and decisions are reported to the user. After training concludes, FedLAD exports all relevant artifacts, including model checkpoints, metric logs, and adaptation history to support analysis.

This end-to-end workflow demonstrates how researchers can use FedLAD to easily test, compare, and analyze their customized algorithms in realistic federated LAD scenarios, benefiting from the testbed's modular design, automation capabilities, and adaptive control mechanisms.

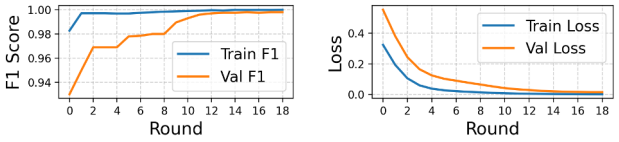
B. Experiment Results

To illustrate the evaluations of FedLAD, this subsection reports example results from running an LAD model on a benchmark dataset under a federated setup with 50 simulated clients. After specifying the model and training configurations, researchers can use FedLAD's built-in logging and visualization modules to obtain detailed runtime insights.

Detection Performance. Fig. 4 shows the detection performance curve of LogAnomaly under a non-IID data distribution, illustrating how performance varies across training rounds in the absence of adaptation. The training monitor captures key performance indicators such as F1-score and

```
round,train_loss,val_loss,train_f1,val_f1
0,0.32485184302082226,0.5542810282707215,0.982761170323619,0.9302361356239637
1,0.1937191289608381,0.3820453407764435,0.9970874696831775,0.9499116441402634
2,0.10578006971884107,0.2442496123313904,0.9970944536668668,0.9689835923203374
3,0.059443719062486 71,0.16484102243185042,0.9970966589354525,0.9689835923203374
4,0.03784243322357 5125,0.12476855105161667,0.9968798032128078,0.9689835923203374
5,0.027295263650784 85,0.10296125420928001,0.9968935984738957,0.9779956871546823
```

(a) Per-Round Metrics Tracked by the Training Monitor.



(b) F1-score over rounds (c) Loss over rounds

Fig. 4: Detection performance in FedLAD. Parameter settings: *dataset* = Thunderbird, *distribution* = non-IID, *model* = LogAnomaly.

```
=== Final Evaluation Report ===

True Label Distribution:
Counter({1: 2000, 0: 2000})

Predicted Label Distribution:
Counter({0: 2011, 1: 1989})

Final Evaluation Results:
Accuracy     : 0.9972
Precision    : 1.0000
Recall       : 0.9945
F1 Score     : 0.9972
ROC-AUC      : 1.0000

Detailed Report:
              precision    recall  f1-score   support

           0     0.9945    1.0000    0.9973      2000
           1     1.0000    0.9945    0.9972      2000

    accuracy                         0.9972      4000
   macro avg     0.9973    0.9972    0.9972      4000
weighted avg     0.9973    0.9972    0.9972      4000

Confusion Matrix:
[[2000    0]
 [  11 1989]]

=== Training Summary ===
early_stop_round: 21
strategy_switch_count: 1
wall_clock_time: 629.82
```

Fig. 5: Example of the final evaluation report with adaptive control enabled. Parameter settings: *dataset* = Thunderbird, *distribution* = IID, *model* = NeuralLog.

loss at each round. These results are automatically logged in CSV format for further analysis. Developers can easily visualize convergence behavior across different configurations to compare training stability and model generalization. In this example, the model rapidly achieves an F1-score above 0.98 within two rounds and converges toward 0.997, demonstrating strong detection performance with minimal training cost.

To further demonstrate FedLAD's versatility, we compare centralized and FL training using NeuralLog on the Thunderbird dataset. Existing FL-based LAD implementations are often tightly integrated with specific models or datasets, lacking standardized support for evaluating alternative architectures or training setups such as centralized LAD. The centralized setup processes all logs on a single client, while the FL distributes data across ten clients. Both achieve comparable accuracy (0.9977 vs. 0.9992), with the FL variant slightly outperforming in recall and F1-score. Training time remains similar (80.2s vs. 90.1s), and the communication overhead is moderate, which is

about 27.5MB per client across 11 rounds. These results show that FedLAD supports FL workflows with minimal overhead, validating its extensibility and practical efficiency.

FedLAD's adaptive Executor empowers researchers to evaluate runtime decision-making under dynamic conditions. Fig. 5 illustrates the execution trace for NeuralLog under an IID setting, where the adaptive mechanism enabled both early stopping and strategy switching. Specifically, the Executor halted training at round 21 upon detecting a validation, reducing unnecessary computation. Additionally, it triggered a strategy switch in response to performance drift, demonstrating how FedLAD supports real-time control adaptations without entangling model logic. Besides, total training time was approximately 630 seconds, including model updates, aggregation, and adaptation overheads. The high F1-score of 0.9972 and rapid convergence underscore FedLAD's utility in supporting fast FedLAD evaluation.

To evaluate the benefits of these adaptive features, we compared them with a fixed-strategy baseline using the same model and dataset but without early stopping or strategy switching. The adaptive setup achieved a higher F1-score (0.9972 vs. 0.9946), over 25% less training time, fewer communication rounds, and better convergence. These results highlight the practical advantage of FedLAD's self-adaptation over fixed-policy FL training, suggesting that adaptive mechanisms improve efficiency and generalization by mitigating overfitting.

VI. Threat to Validity

While FedLAD provides a flexible and modular testbed for benchmarking federated LAD models, it also has certain limitations. First, FedLAD currently simulates clients on a single machine, which may not fully reflect network communication overheads in real-world FL deployments. Secondly, although we support three public datasets, expanding to industrial-scale logs may require custom parsers and encoders. Finally, the adaptation controller currently focuses on early stopping and aggregation switching, where future work can explore more advanced control strategies such as reinforcement learning.

VII. Conclusion and Future Works

This paper presented FedLAD, a modular and self-adaptive testbed for benchmarking LAD in federated environments. FedLAD supports realistic and reproducible experimentation by enabling pluggable LAD models, benchmark log datasets, configurable IID/non-IID data partitioning, and multiple aggregation strategies. Its modular architecture separates configuration, coordination, and execution concerns, supporting extensibility, scalability, and reproducibility. FedLAD further integrates self-managing features, including self-monitoring for logging per-round metrics, self-configuration for automatic hyperparameter tuning, and self-adaptation for enabling runtime decisions such as strategy switching and early stopping. By lowering the barrier to prototyping, evaluation, and extension, FedLAD provides a unified platform for advancing privacy-preserving, scalable, and adaptive LAD research under federated constraints. Future work will explore deployment across real-world distributed agents, integration of advanced privacy defenses, and support for upstream log preprocessing tasks, including automated log parsing.

References

[1] M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger, "Deep learning for anomaly detection in log data: A survey," *Machine Learning with Applications*, vol. 12, p. 100470, 2023.

[2] M. Catillo, A. Pecchia, and U. Villano, "Autolog: Anomaly detection by deep autoencoding of system logs," *Expert Systems with Applications*, vol. 191, p. 116263, 2022.

[3] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" in *Proceedings of the 44th international conference on software engineering*, 2022, pp. 1356–1367.

[4] V. Mothukuri, P. Khare, R. M. Parizi *et al.*, "Federated-learning-based anomaly detection for iot security attacks," *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 2545–2554, 2021.

[5] Y. Xie, H. Zhang, and M. A. Babar, "Loggd: Detecting anomalies from system logs with graph neural networks," in *2022 IEEE 22nd International conference on software quality, reliability and security (QRS)*. IEEE, 2022, pp. 299–310.

[6] C. Zhang, Y. Xie, H. Bai *et al.*, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[7] C. Hu, M. Zhang, N. Li, J. Li *et al.*, "Adapting aggregation rule for robust federated learning under dynamic attacks," in *2025 IEEE/ACM 20th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2025, pp. 171–177.

[8] D. Domini, G. Aguzzi *et al.*, "Proximity-based self-federated learning," in *2024 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 2024, pp. 139–144.

[9] D. J. Beutel, T. Topal, A. Mathur *et al.*, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

[10] A. Oliner *et al.*, "What supercomputers say: A study of five system logs," in *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*. IEEE, 2007, pp. 575–584.

[11] W. Xu, L. Huang, A. Fox *et al.*, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 117–132.

[12] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[13] W. Meng, Y. Liu, Y. Zhu *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI*, vol. 19, no. 7, 2019, pp. 4739–4745.

[14] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 492–504.

[15] B. McMahan, E. Moore, D. Ramage *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[16] S. P. Karimireddy, S. Kale, M. Mohri *et al.*, "Scaffold: Stochastic controlled averaging for federated learning," in *International conference on machine learning*. PMLR, 2020, pp. 5132–5143.

[17] S. Reddi, Z. Charles, M. Zaheer *et al.*, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.

[18] T. Li, A. K. Sahu, M. Zaheer *et al.*, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

[19] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[20] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 international joint conference on neural networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[21] B. Li, S. Ma, R. Deng, K.-K. R. Choo, and J. Yang, "Federated anomaly detection on system logs for the internet of things: A customizable and communication-efficient approach," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1705–1716, 2022.

[22] Y. Guo, Y. Wu, Y. Zhu *et al.*, "Anomaly detection using distributed log data: A lightweight federated learning approach," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.