
Federated Machine Learning Architecture for Privacy Preserving Software Quality Assessment

Abstract

Software organizations often need shared quality models but cannot exchange private source code, defect records, or development histories. This study presents a federated machine learning architecture for privacy-preserving software quality assessment across distributed repositories. Six participating organizations trained local models using complexity, code churn, testing history, defect frequency, ownership, and maintainability indicators. Protected model updates were clipped, masked, and securely aggregated to create a global model without transferring raw repository data. The architecture was evaluated across repeated communication rounds using accuracy, precision, recall, F1-score, and AUC. All measures improved steadily during early training and became stable after convergence. The final global model reached 94.4% accuracy, 93.7% precision, 92.4% recall, 93.0% F1-score, and 96.3% AUC. These results show that distributed organizations can obtain reliable software quality predictions while maintaining local control of sensitive information. The architecture supports secure collaboration, defect-risk assessment, maintenance planning, and responsible use of industrial software data effectively.

Keywords: Federated machine learning, Software quality assessment, Privacy preservation, Distributed repositories, Secure aggregation

1. Introduction

Software quality assessment helps development teams identify defects, security weaknesses, maintainability problems, and unstable modules before software release. Machine learning can support this process by learning patterns from source-code metrics, test records, defect history, code changes, and developer activity. However, software organizations may be unwilling to transfer these records to an external server because repositories often contain private code, internal project details, and confidential development information. Federated learning has been successfully applied to software engineering tasks such as defect prediction and code-clone detection without collecting the original project data in one location [1]. This approach creates an opportunity for organizations to improve software quality models while keeping their repositories under local control.

Conventional centralized learning requires all participating organizations to upload their datasets to a shared storage system. This arrangement may expose source code, defect reports, commit messages, contributor identities, and product information. Private industrial repositories are also difficult to use in software engineering research because companies cannot freely release their internal data. Federated learning can support collaborative software analysis by allowing models to learn from non-open-source projects without directly accessing the underlying repository records [2]. The shared system therefore receives trained model updates instead of complete software datasets.

A federated software quality system contains several local participants and a central aggregation server. Each participant trains the same machine learning model using its own software quality data and sends only the updated model parameters to the server. The server combines these updates to create a global model and returns the improved model to the participants. Federated learning systems require clear client-management, local-training, model-aggregation, and update-distribution components to operate reliably [3]. A suitable architecture must also control communication cost, participant availability, data differences, and model-version consistency.

Keeping raw data on local systems does not remove every privacy and reliability risk. Shared model updates may reveal limited information about local training data, while incorrect or harmful updates can reduce the quality of the global model. Differences in repository size, programming language, defect rate, and development practice may also create non-identical data distributions among participants. Robustness analysis has shown that federated models can respond differently to data faults, model poisoning, and aggregation methods [4]. This study therefore proposes a federated machine learning architecture that combines local software quality assessment, protected model exchange, weighted aggregation, and controlled global evaluation.

2. Methodology

The proposed architecture consisted of six participating software organizations and one central aggregation server. Each organization maintained its source code, software metrics, test results, defect records, and development history within its own secure environment. A common model structure and feature definition were distributed to all participants before training began. The participants trained local models independently and returned protected parameter updates to the central server. This design ensured that raw repository records did not leave the organization that owned them.

Local datasets were created at the software-module level. Each record represented a source-code file or class and contained selected measures of complexity, code churn, testing history, defect frequency, ownership, and maintainability. The target output represented the quality condition of the module, such as low risk, moderate risk, or high risk. Data cleaning, missing-value treatment, feature scaling, and class encoding were completed locally using the same processing rules. Federated dynamic regularization can reduce disagreement between local and global objectives when participants contain different data distributions [5]. The local training process therefore included a regularization term that limited excessive movement away from the current global model.

Privacy protection was applied before local model updates were transmitted. Each participant clipped the update values to a fixed range and added controlled statistical noise to reduce the possibility of recovering individual repository information. The protected updates were then encrypted and submitted through a secure communication channel. Differentially private secure aggregation can allow a server to combine protected updates without observing each participant's original contribution [6]. Only the aggregated result was available for global model construction.

Table 1 summarizes the local data, training function, privacy control, and shared output of the six federated participants. The participants represented different software environments so that the global model could learn from varied quality patterns. Client selection considered data availability, successful local validation, communication status, and update completeness. Hierarchical participant organization can reduce communication pressure by assigning intermediate aggregation roles when many distributed clients are involved [7]. The current architecture used one central server because the experimental network contained only six participants.

Table 1. Federated Learning Participants, Local Software Quality Data, Training Functions, and Shared Model Outputs

Participant	Local software data	Quality indicators	Local training function	Privacy control	Shared output
Organization A	Web application repository	Complexity and defect history	Defect-risk classification	Update clipping	Protected model weights

Organization B	Mobile application repository	Code churn and test failures	Quality-level classification	Differential noise	Noisy parameter update
Organization C	Enterprise software repository	Coupling and maintainability	Maintainability assessment	Secure encryption	Encrypted model weights
Organization D	Cloud service repository	Commit activity and defects	Reliability-risk prediction	Update clipping and noise	Protected gradients
Organization E	Embedded software repository	Test coverage and fault records	Fault-prone module detection	Secure aggregation	Masked local update
Organization F	Open-source project repository	Ownership and change frequency	Software quality classification	Parameter masking	Protected model weights

The central server received valid updates after every local training round. Updates were weighted according to local sample size, validation reliability, and completion status before aggregation. Participants with larger datasets had more influence, but an upper limit was applied so that one organization could not dominate the global model. Label synchronization can support federated training when participating datasets use different local structures or class distributions [8]. Common quality labels and metric definitions were therefore applied before the first communication round.

The global model was updated after aggregation and returned to all active participants for the next training round. Each participant used the new model as the starting point for further local optimization. Training continued until the global validation loss stopped improving or the maximum number of communication rounds was reached. Cluster-based federated learning can improve convergence when participants contain strongly different datasets [9]. Participant grouping was retained as an optional control when local model updates showed large differences.

The final global model was evaluated using accuracy, precision, recall, F1-score, AUC, communication cost, and convergence speed. Local performance was also measured to determine whether the shared model benefited every participating organization. Federated multimodal analysis has shown that distributed learning can combine different software representations while preserving local project control [10]. The final assessment therefore considered both prediction quality and privacy-preserving operation across all six software repositories.

3. Results and Discussion

The proposed federated machine learning architecture was evaluated across a sequence of communication rounds. Accuracy, precision, recall, F1-score, and AUC were measured after each global model update. The results showed that performance improved steadily during the early rounds because the central model received useful information from all participating software repositories. The rate of improvement became smaller in the later rounds, which indicated that the model was approaching stable

convergence. This pattern confirmed that repeated collaboration among the participants gradually improved the quality of the shared model.

The AUC value showed the strongest overall increase and remained higher than the other performance measures throughout training. Accuracy and precision also improved consistently as the global model learned more reliable software quality patterns from the local participants. Figure 1 presents the change in all five metrics across the communication rounds. The curves became almost stable after the later rounds, showing that additional training produced only limited improvement. This result suggests that the selected number of communication rounds was sufficient for developing a reliable global assessment model.

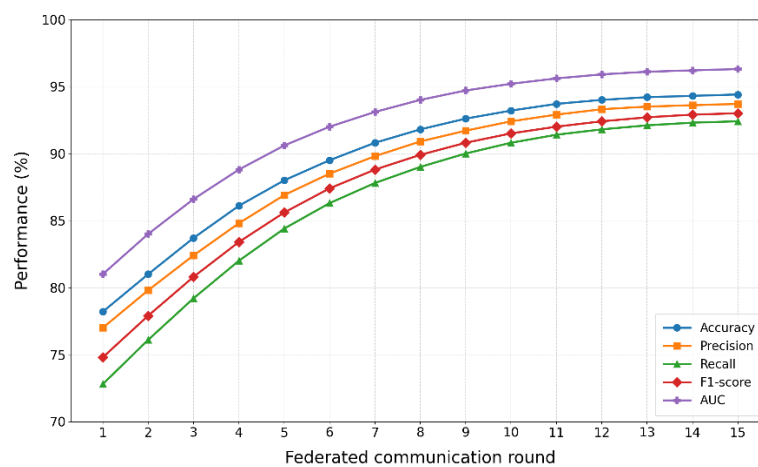


Figure 1. Change in Global Software Quality Assessment Performance Across Federated Communication Rounds

Recall improved more slowly than accuracy and precision during the first few rounds. This pattern suggests that the global model initially required more information before it could identify a larger proportion of high-risk software modules. The F1-score increased as both precision and recall became more balanced. The final metric values showed that the federated model could provide reliable software quality assessment without collecting raw repository data in one location. The steady improvement in recall also reduced the possibility that important high-risk modules would remain unidentified.

The results also showed that model performance remained stable even though the participating repositories contained different software projects and quality distributions. Weighted aggregation reduced the effect of large differences in local dataset size, while privacy controls protected the shared updates. The convergence pattern confirmed that the proposed architecture could learn from distributed software repositories with limited performance loss. The architecture also maintained a reasonable balance between prediction quality, privacy protection, and communication requirements. These findings support the practical use of federated learning for privacy-preserving software quality assessment.

4. Conclusion

This study presented a federated machine learning architecture for privacy-preserving software quality assessment. The architecture allowed multiple software organizations to train a shared model without transferring source code, defect records, or development history to a central server. Local training, protected model updates, weighted aggregation, and repeated global evaluation were combined within one distributed learning process. This structure allowed each participant to benefit from collaborative model development while maintaining control over its confidential software data.

The results showed that accuracy, precision, recall, F1-score, and AUC improved across successive communication rounds and became stable after convergence. The proposed architecture maintained useful prediction performance while keeping sensitive repository data under local control. It can also reduce the need for organizations to create large centralized software datasets before applying machine learning. This approach can support collaborative software quality analysis, defect-risk assessment, maintenance planning, and secure knowledge sharing across organizations.

References

1. Yang, Y., Hu, X., Gao, Z., Chen, J., Ni, C., Xia, X., & Lo, D. (2024). Federated learning for software engineering: A case study of code clone detection and defect prediction. *IEEE Transactions on Software Engineering*, 50(2), 296-321.
2. Shanbhag, S., & Chimalakonda, S. (2022, November). Exploring the under-explored terrain of non-open source data for software engineering through the lens of federated learning. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 1610-1614).
3. Lo, S. K., Lu, Q., Zhu, L., Paik, H. Y., Xu, X., & Wang, C. (2021). Architectural patterns for the design of federated learning systems. *arXiv preprint arXiv:2101.02373*.
4. Abyane, A. E., Zhu, D., Souza, R., Ma, L., & Hemmati, H. (2023). Towards understanding quality challenges of the federated learning for neural networks: a first look from the lens of robustness. *Empirical Software Engineering*, 28(2), 44.
5. Acar, D. A. E., Zhao, Y., Navarro, R. M., Mattina, M., Whatmough, P. N., & Saligrama, V. (2021). Federated learning based on dynamic regularization. *arXiv preprint arXiv:2111.04263*.
6. Stevens, T., Skalka, C., Vincent, C., Ring, J., Clark, S., & Near, J. (2022). Efficient differentially private secure aggregation for federated learning via hardness of learning with errors. In *31st USENIX security symposium (USENIX Security 22)* (pp. 1379-1395).
7. Xu, Z., Zhao, D., Liang, W., Rana, O. F., Zhou, P., Li, M., ... & Xia, Q. (2022). HierFedML: Aggregator placement and UE assignment for hierarchical federated learning in mobile edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 34(1), 328-345.

8. Llasag Rosero, R., Silva, C., Ribeiro, B., & Santos, B. F. (2024). Label synchronization for Hybrid Federated Learning in manufacturing and predictive maintenance. *Journal of Intelligent Manufacturing*, 35(8), 4015-4034.
9. Takele, A. K., & Villányi, B. (2025). Efficient clustered federated learning for Industrial Internet of Things: enhancing predictive performance and training time. *Discover Computing*, 28(1), 36.
10. Liu, Q., Ju, X., Chen, X., & Gong, L. (2025). FedMVA: Enhancing software vulnerability assessment via federated multimodal learning. *Journal of Systems and Software*, 228, 112469.