Enhancing Java Grading Through Large Language Models

Lei Yao
University of Wisconsin-Milwaukee
Milwaukee, Wisconsin, USA
leiyao@uwm.edu

Hiba Alsghaier
University of Wisconsin-Milwaukee
Milwaukee, Wisconsin, USA
alsghai2@uwm.edu

Tian Zhao*
University of Wisconsin-Milwaukee
Milwaukee, Wisconsin, USA
tzhao@uwm.edu

Abstract

The rapid evolution of Large Language Model (LLM) has significantly influenced AI applications in education, particularly in the instruction of computer programming. To address data privacy concerns that hinder LLM adoption, we present a privacy-preserving assessment framework that leverages the Federated Learning (FL) paradigm. By applying algorithms such as FedAvg, FedProx, and FedAdam to the instruction tuning of the Qwen3 base model using a dataset of over 1,500 Java tasks, we evaluated the feasibility of decentralized training for sensitive data. Experimental results on a held-out test set demonstrate that the federated model effectively learns to identify and explain syntax, runtime, and logical errors, validating a secure and scalable path for AI tutoring systems.

CCS Concepts

• **Computing methodologies → Machine learning**; • **Applied computing → Education**.

Keywords

Large Language Model, Machine Learning, Computer Education, Java, Code Understanding, Automated Grading

ACM Reference Format:
Lei Yao, Hiba Alsghaier, and Tian Zhao. 2026. Enhancing Java Grading Through Large Language Models. In *The 41st ACM/SIGAPP Symposium on Applied Computing (SAC '26), March 23–27, 2026, Thessaloniki, Greece.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3748522.3780023

1 Introduction

Large Language Model (LLM) offers promising support for higher education [10], but raises concerns regarding accuracy, bias, data privacy, and academic integrity. This research addresses some of these challenges by developing a privacy-preserving LLM tool for assessing Java programming assignments, using Federated Learning (FL) and synthetic surrogate data to avoid exposure of raw student submissions. Leveraging recent advances in LLM, the framework enhances contextual code analysis, error classification, and explanation generation, contributing to emerging hybrid human-AI teaching models that combine efficiency with essential pedagogical oversight.

LLM holds promise for automated grading of programming assignments. However, standard centralized training introduces data privacy risks, as it requires aggregating student code into a central server, potentially violating educational data policies. In parallel, the assessment methodology itself faces inherent limitations. Current automated grading systems [1] are entirely based on instructor-written test cases, which impose significant authoring overhead. These tests are time-consuming to create and frequently fail to cover important edge cases, boundary conditions, or input variations. As a result, incorrect or hard-coded student solutions often pass undetected, especially when random tests are limited or when specific data types and special conditions are not thoroughly exercised.

To address these limitations, we propose a privacy-preserving framework that leverages established FL paradigms to perform instruction-tuning on the Qwen3 model, specifically for automated Java code assessment. Our approach empowers the open-source model to generate precise, natural language feedback that complies with rigorous assessment standards, ensuring data privacy while maintaining alignment with instructional rubrics. The code and data of this work are available at https://github.com/uwm-se/PE.

2 Literature Review

GenAI in Education. Researchers have noted that existing datasets for LLM-based computing education are often inadequate for rigorous evaluation. Prather *et al.* [7] showed that most publicly available datasets were not created for LLM code generation research and frequently lack critical pedagogical context, such as starter code, data files, and reliable test suites. The survey further emphasizes the limited diversity of available datasets, which predominantly contain Python-focused problems and are thus insufficient for assessing models across languages and task types. Building on this foundation, Prather *et al.* [8] investigated how instructors themselves engage with GenAI tools, often using them to develop supportive learning materials.

FL for Education. A personalized FL framework has been used to improve student modeling within underrepresented groups [2]. The framework predicts learning outcomes based on student activities, with subgroups defined by demographic variables. Evaluation across three online course datasets shows that the method consistently, and in some cases significantly, outperforms baseline algorithms in maximizing prediction accuracy for each subgroup.

Educational data mining techniques have also been applied to assess students' academic performance [3]. A federated learning model was introduced to forecast learning outcomes across multiple educational institutions, with a strong focus on preserving data privacy. Feature refinement, extensive model testing, and comprehensive performance evaluation were conducted to ensure reliability. Results show that Federated Learning and Support Vector Machines achieved the best performance on both training and testing datasets. The use of advanced methodologies and comparative analyses provides meaningful insights into student performance prediction, with the potential to inform improvements in educational practice.

Hierarchical Personalized Federated Learning (HPFL) is a client-server framework that supports privacy-aware learning across heterogeneous users [11]. The framework organizes data into a hierarchical structure, enabling flexible segmentation in accordance with differing privacy requirements. Each client trains a user model with components tailored to the hierarchical structure of its local data, and applies a personalized update strategy to capture statistical variations among users. In parallel, the server performs component-level aggregation, allowing diverse user models to be combined flexibly – an essential capability when models differ due to privacy constraints or data characteristics.

LLM for Education. LLMs have been integrated into Integrated Development Environments (IDEs) to support real-time code understanding. The GILT system, a VS Code plugin using GPT-3.5-turbo, enables developers to request context-aware explanations without explicit prompting [6]. Empirical evaluation revealed significant improvements in task completion rates and user satisfaction compared to traditional search-based approaches. JavaLLM [12] exemplifies the potential of domain-tuned LLMs for programming education. By fine-tuning on Java-centric datasets, including code snippets, documentation, and instructional material, the model achieved superior results in Java code generation and question answering, providing tailored support aligned with programming curricula.

3 Dataset and Preprocessing

3.1 Data Source

We build our dataset from a publicly available Java benchmark on HuggingFace, `Dataset678/humaneval-java`. Each item in this dataset provides a Java programming task description and reference solution code. We first extract the natural language task statement from the original `prompt` field and treat it as the assignment specification for students.

To simulate realistic student submissions, we use the Sonnet 4.5 model to generate between 10 and 15 candidate Java solutions per task. The generated code is prompted to include a mixture of correct implementations and solutions containing different types of mistakes (e.g. syntax, runtime, and logical errors), thereby approximating the diversity observed in real classroom settings.

3.2 Synthetic Data Generation

We generate structured feedback for each synthetic student solution, identifying and explaining all errors present in the code. Feedback generation is guided by a rubric-style system prompt that specifies the expected output and restricts errors to predefined categories. This ensures that feedback is uniform, interpretable, and aligned with the evaluation criteria. An example data record is shown below:

```
{"system_prompt": "Analyze the student's Java code according
                   to the given task requirements.",
 "user_prompt":
  "Java code requirement: Input to this function is a string...
    student code: import java.util.*; class Solution {...}",
 "feedback":
  "1) Logical Error: Does not handle overlaps correctly.
  2) Logical Error: Returns segments count, not occurrence count"}
```

To ensure label quality, we manually inspected all records to verify that the feedback accurately matches the student code and that each reported error is correctly categorized and explained. The final dataset contains 1,556 labeled examples, of which 1,327 are used for training, 100 for validation, and 129 for testing.

Each example is formatted as an instruction-response pair. The system prompt defines the grading rules, error taxonomy, and output format, while the user input combines the task description with the student code. The feedback serves as the assistant response. This instruction-tuning formulation enables the model to learn how to generate structured evaluations conditioned on the code, rather than simply assigning a score.

4 Model Architecture and Fine-Tuning

4.1 Base Model and FL Strategies

For the experiment, we selected Qwen3-4B-Base model since the base variant is well suited for fine-tuning, and the Qwen3 series already demonstrates strong coding capability from pre-training. In federated learning scenarios, client devices may not be able to host very large models, and the 4B configuration offers a practical balance between model effectiveness and deployment constraints. The experiments were conducted on two A100 GPUs (80 GB VRAM each), with each GPU running a separate federated client in parallel to simulate distributed training and accelerate model updates. Three federated optimization strategies were used during training: FedAvg [5], FedProx [4], and FedAdam [9].

4.2 Scoring Criteria and Evaluation Process

Model evaluation was conducted using a scoring framework tailored for the Java error-detection task. The test set contained pairs of reference feedback and model-generated feedback, and each generated output was compared against the reference in terms of error identification, classification, and explanatory adequacy. The scoring was performed using a GPT-assisted evaluation function that applied a set of predefined criteria shown in Table 1.

Table 1: Evaluation Metrics

Metric (0–10)	Description
Count score	Correct number of errors.
Type score	Correct error categories.
Content score	Semantic consistency with the reference feedback.

5 Results and Discussion

5.1 Baselines and Model Scaling Analysis

To establish a performance upper-bound for our FL experiments, we first conducted centralized training on both Qwen3-4B-Base and Qwen3-8B-Base models. Under identical LoRA configurations ($r = 16, \alpha = 32$) and an 80-minute training budget, the results yielded a counterintuitive insight into model efficiency. As detailed in Table 2, the smaller Qwen3-4B-Base achieved a total score of 19.95 (66.5%), slightly outperforming the 8B variant at 19.44 (64.8%). The 4B model demonstrated superior convergence in "Error Type" classification (8.94 vs. 8.53), suggesting that under restricted computational resources and data volumes, smaller models may offer better immediate adaptability than their larger counterparts, which likely remained under-fitted.

Table 2: Performance Comparison of Centralized Baselines: Qwen3-4B vs. Qwen3-8B. Both models were fine-tuned using LoRA ($r = 16, \alpha = 32$) for approximately 80 minutes.

Model	Parameters	Type	Count	Content	Total Score (%)
Qwen3-4B-Base	4B	**8.94**	**4.95**	**6.06**	**19.95 (66.5%)**
Qwen3-8B-Base	8B	8.53	4.91	6.00	19.44 (64.8%)

5.2 Hyperparameter Sensitivity Analysis

Table 3: Hyperparameter sensitivity analysis for centralized Qwen3-8B-Base model by comparison of learning rate (LR), LoRA rank (r), and training epochs across three evaluation metrics.

Config	Hyperparameters				Performance Metrics			
	LR	Rank	Alpha	Epochs	Type	Count	Content	Total
Config A	1e-4	16	32	3	8.53	4.91	6.00	19.44
Config B	2e-4	32	64	3	8.14	5.14	6.14	19.42
Config C	5e-5	**32**	**64**	**5**	**9.16**	**5.20**	**6.26**	20.62

To determine the theoretical performance limit of the Qwen3-8B-Base model in a centralized setting, we conducted a hyperparameter sensitivity analysis involving variations in learning rate (LR), LoRA Rank (r), and training epochs. The results, shown in Table 3, indicate performance saturation. Despite significant adjustments to the model's capacity (doubling the rank from 16 to 32) and optimization dynamics (varying LR from 5e-5 to 2e-4), the performance metrics exhibited remarkable stability rather than drastic fluctuations. This marginal variance suggests that the centralized model has reached its *empirical upper bound* for this specific instruction-tuning task. Establishing this stable ceiling is pivotal as it defines the *theoretical limit* for our investigation.

5.3 Federated Instruction Tuning Performance

Table 4: Comparison between centralized and FL approaches on Qwen3-4B

Training Type	Type	Count	Content	Total Score (%)
Base Model	3.28	3.84	3.12	10.24 (34.13%)
FedAvg	7.8	4.67	5.73	18.20 (60.67%)
FedProx	7.58	4.26	5.26	17.10 (57.00%)
FedAdam	6.23	5.21	4.97	16.41 (54.70%)

Table 4 compares the performance between the Qwen3-4B base model (utilizing few-shot CoT) and various FL strategies. Most notably, federated instruction tuning strategies achieved performance levels comparable to the centralized training baseline, effectively bridging the gap between privacy-preserving distributed learning and traditional centralized optimization.

Significantly outperforming the Qwen3-4B-Base model equipped with few-shot Chain-of-Thought (CoT), our federated approach demonstrates robust convergence. Among the evaluated strategies, FedAvg delivered the most balanced performance, elevating error-type score from a baseline of 3.28 to 7.8 and feedback content score to 5.73. FedAdam has the best error-count score at 5.21. These distributed methods approximate the efficacy of centralized training, which shows the practical viability of deploying LLMs in federated settings without significant performance degradation.

6 Conclusion

This study demonstrates that FL strategies can effectively **approximate the performance of centralized training baselines** for LLM instruction-tuning. Our experimental results confirm that by adopting decentralized aggregation methods, it is possible to achieve high-fidelity code assessment capabilities without physically aggregating sensitive student data. This establishes a critical technical validation: high-utility model adaptation and rigorous data privacy are not mutually exclusive, but can be simultaneously achieved through the proposed framework.

Acknowledgments

This work is partially supported by the Northwestern Mutual Data Science Institute (NMDSI) under grant number SS123.

References

[1] Umar Alkafaween, Ibrahim Albluwi, and Paul Denny. 2025. Automating Autograding: Large Language Models as Test Suite Generators for Introductory Programming. *Journal of Computer Assisted Learning* 41, 1 (2025).
[2] Yun-Wei Chu, Seyyedali Hosseinalipour, Elizabeth Tenorio, Laura Cruz, Kerrie Douglas, Andrew Lan, and Christopher Brinton. 2022. Mitigating Biases in Student Performance Prediction via Attention-based Personalized Federated Learning. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 3033–3042.
[3] Umer Farooq, Shahid Naseem, Jianqiang Li, Tariq Mahmood, Amjad Rehman, Tanzila Saba, and Luqman Mustafa. 2023. Analysis of the Factors Influencing the Predictive Learning Performance Using Federated Learning. *The Journal of Supercomputing* (2023).
[4] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. In *Proceedings of Machine Learning and Systems (MLSys)*, Vol. 2. MLSys, 429–450.
[5] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, Vol. 54. PMLR, 1273–1282.
[6] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to Help With Code Understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 13 pages.
[7] James Prather, Paul Denny, Juho Leinonen, et al. 2023. The Robots Are Here: Navigating the Generative AI Revolution in Computing Education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*. 108–159.
[8] James Prather, Juho Leinonen, Natalie Kiesler, et al. 2025. Beyond the Hype: A Comprehensive Review of Current Trends in Generative AI Research, Teaching Practices, and Tools. *2024 Working Group Reports on Innovation and Technology in Computer Science Education* (2025), 300–338.
[9] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. 2021. Adaptive Federated Optimization. In *International Conference on Learning Representations*.
[10] Maya Usher. 2025. Generative AI vs. Instructor vs. Peer Assessments: A Comparison of Grading and Feedback in Higher Education. *Assessment & Evaluation in Higher Education* (2025), 1–16.
[11] Jinze Wu, Qi Liu, Zhenya Huang, Yuting Ning, Hao Wang, Enhong Chen, Jinfeng Yi, and Bowen Zhou. 2021. Hierarchical Personalized Federated Learning for User Modeling. In *Proceedings of the Web Conference 2021*. 957–968.
[12] Jingying Zhang and Kang Liu. 2024. JavaLLM: A Fine-Tuned LLM for Java Programming Education. In *2024 8th International Symposium on Computer Science and Intelligent Control (ISCSIC)*. 276–280.