Enhancing the IntelliFL Federated Learning Framework

AJ Barea | **Rochester Institute of Technology**

1. Project Objectives

This project extended the IntelliFL federated learning framework, a robust Python simulation engine developed by Dr. Leon Reznik's PhD research team. The framework supports 10 aggregation strategies, multiple datasets, and Byzantine attack mitigation. However, the original CLI-only design lacked the interfaces, documentation, and test coverage needed for broad student use. The primary goal was to improve accessibility, reliability, and reproducibility for new student researchers by engineering a production-ready, full-stack infrastructure layer around the core simulation engine.

2. Key Enhancements

A comprehensive pytest suite was built to validate the core engine and ensure reproducible results. This suite includes 1,136 total automated tests covering unit, integration, and performance testing, achieving over 95% code coverage on all critical simulation logic. A total of 280 parameterized test combinations validate the 10 aggregation strategies against various datasets and attack scenarios. PyTorch and Flower mocks enabled fast, isolated testing without requiring distributed infrastructure.

A FastAPI backend decouples the engine from the user interface and enables programmatic access through 12 RESTful endpoints for managing simulations, tracking status, and retrieving results. The backend includes secure integration with the Hugging Face Hub for dataset validation and implements non-blocking async/await patterns for I/O operations to prevent server hangs.

A React 19 single-page application complements the CLI workflow with an accessible graphical interface. The frontend comprises over 50 React components building a cohesive dashboard, a configuration wizard with 30+ validated fields, interactive Recharts visualizations for real-time metrics, and extensive educational tooltips that explain complex federated learning concepts. A modern developer experience toolchain was established with automated quality checks using Ruff, mypy, and ESLint. Parallel test execution via pytest-xdist reduced total test runtime to under 90 seconds.

3. Process and Lessons Learned

The project followed an iterative, specification-driven development approach with continuous integration. Writing detailed technical specifications upfront reduced ambiguity and improved development efficiency. Writing tests concurrently with feature development created a safety net and caught edge cases early. Designing from the perspective of a student new to federated learning directly informed helpful tooltips and validation logic. Enforcing automated quality checks reduced repetitive tasks and maintained high code quality throughout development.

Key technical challenges included testing distributed components, which was solved by creating mocks for PyTorch and Flower. Implementing non-blocking async/await patterns for the Hugging Face Hub API prevented server hangs during dataset validation. Complex dynamic form state management in React was mastered to support the 10 different aggregation strategies and their unique parameters.

4. Impact and Outcomes

This project transformed a research prototype into a robust and accessible platform. For student researchers, the guided web UI and educational tooltips dramatically lower the learning curve, while the test suite increases confidence in the simulation's correctness and reproducibility. The UI allows users to configure and compare strategies without needing deep CLI expertise. For framework maintainers, the CI test suite automatically catches bugs before they reach users, the API-first design enables future integrations such as Jupyter notebooks or cloud deployments, and enforced code quality standards with a 3:1 test-to-code ratio make the project easier to maintain and extend.

5. Technologies

The backend was built with Python, FastAPI, and pytest, utilizing the Flower AI framework for federated learning simulation. The frontend uses React 19, Vite, and Recharts for data visualization, following Material Design 3 principles. Data management leverages Hugging Face Datasets, and the CI/CD pipeline uses GitHub Actions with CodeCov for coverage reporting.