Predicting Prolonged ICU Length of Stay Using Centralized and Federated Machine Learning

ARI 510 - Machine Learning, University of Michigan-Flint, Fall 2025
Jamie Ontiveros, Jim Prantzalos

1. Project Overview

The goal of this project was to build machine-learning models capable of predicting whether a patient's ICU stay would be prolonged, defined in our work as three or more days. Predicting prolonged ICU stays is clinically and operationally important because early identification of high-risk patients can support staffing decisions, resource allocation, and escalation-of-care planning.

We evaluated this prediction task in two settings:

1. **Centralized machine learning:** a traditional approach where all data is pooled together and trained on a single machine.
2. **Federated learning:** a distributed approach where separate clients train models on local data, and only model updates—not raw patient data—are shared with a central server.

The dataset used was the publicly available eICU Collaborative Research Database (eICU-CRD v2.0), containing detailed clinical records for ICU stays across U.S. hospitals from 2014–2015. Our evaluation metrics included accuracy, precision, recall, and F1-score to provide a balanced view of performance given the mild class imbalance in prolonged vs. non-prolonged stays.

This project offers a clear comparison between centralized and federated learning and demonstrates how an ML application can be designed to respect privacy-preserving constraints while maintaining strong clinical predictive value.

2. Data Collection

2.1. Dataset Source and Licensing

The dataset used in this project was eICU-CRD, a large-scale, open-access critical care dataset released by the MIT Laboratory for Computational Physiology under a permissive data-use agreement. The data dictionary and schema are fully documented and available on the associated project website:

https://eicu-crd.mit.edu/about/eicu

All data is de-identified according to HIPAA Safe Harbor standards.

2.2. Collection and Structure of the Data

The eICU-CRD dataset contains 31 tables and more than 450 million rows. Each row represents a clinically meaningful event - labs, vitals, nursing notes, APACHE scores, medications, care plans, and more. The key identifier for nearly all tables is patientUnitStayID, corresponding to a single ICU stay.

We focused primarily on the tables that offered clinically meaningful information early in the ICU stay:

- **patient** - demographics, admission details, discharge details
- **hospital** - hospital characteristics (numbedscategory, region, teaching status)
- **apacheapsvar** / **apachepredvar** - physiology and severity-of-illness variables
- **vitalperiodic** / **vitalaperiodic** - vital signs used to derive 24-hour summary features
- **lab** - laboratory measurements used to generate 24-hour lab features
- **infusiondrug** - infusion medication data from the first 24 hours
- **respiratorycare** - ventilation and airway variables from the first 24 hours

2.3. What Each Instance Represents

In this project, one instance corresponds to one ICU stay, represented by a single row in the patient table. To build a structured dataset suitable for machine learning, we engineered features by aggregating or deriving values from several eICU tables, focusing on information available early in the stay.

These engineered features were created through a series of SQL views developed for this project, available in our repository:

https://github.com/umdnp/ari510capstone-fedlearn/tree/main/db/duckdb/ddl/views

The final modeling dataset was constructed from the consolidated view `v_features_icu_stay_clean`, which merges demographic variables, APACHE physiology variables, early vital signs, laboratory summaries, infusion data, respiratory features, and hospital characteristics into a single table suitable for centralized and federated learning:

https://github.com/umdnp/ari510capstone-fedlearn/blob/main/db/duckdb/ddl/views/10_v_features_icu_stay_clean.sql

Many of the engineered features in this view reflect data collected within the first 24 hours of ICU admission, giving the model access to early-stay information that is relevant for predicting prolonged length of stay.

2.4. Dataset Size

After filtering, cleaning, and merging, our final dataset contained:

- 200,859 ICU stays in total
- Approximately 200k usable rows after merging engineered features
- ~60/20/20 split into training, validation, and test sets:
  - Train: 119,787
  - Validation: 39,929
  - Test: 39,930

2.5. Sampling Procedure

All stays were included except those with corrupted or missing core identifiers or obvious outliers (e.g., negative Length of Stay).

2.6. Missing Data Handling

The eICU dataset contains substantial missingness due to irregular sampling of vitals, labs, and recorded interventions. Our preprocessing included:

- Simple imputation for numeric features
- "Unknown" category handling for categorical features
- Binary missingness indicators for some features
- APACHE variables included where available as proxies for illness severity

3. Data Annotation

Although this project used a structured clinical dataset rather than natural text or images, annotation was still required to establish consistent, human-defined mappings for categorical ICU data and to clearly document the meaning of numeric and binary features. This process ensured that machine learning models would treat equivalent real-world concepts consistently and that federated clients would apply identical preprocessing rules.

This consistency was especially important for federated learning, where mismatched categorical mappings across clients would lead to incompatible model updates.

3.1. Annotation Guidelines

The annotation guidelines for this project (annotation_guidelines.pdf) specify the rules annotators must follow when standardizing categorical values and defining numeric feature semantics. These rules include:

- Treating raw text case-insensitively and trimming whitespace
- Mapping hospital-specific or inconsistent labels into a unified set of canonical categories
- Standardizing fallback categories such as unknown, or other
- Preventing category explosion by prohibiting the creation of new categories
- Using the notes column to document ambiguous cases or inconsistencies

The resulting configuration was implemented in Python via the ANNOTATION_CONFIG dictionary, which defines the approved canonical categories and mappings from raw to canonical values.

These guidelines ensured that both centralized and federated learning pipelines used exactly the same encodings.

3.2. Annotation Interface

Per HW3 requirements, the annotation work was completed using an Excel spreadsheet (annotations.xlsx) containing two tabs:

- **Categorical Features** - parent rows represent a feature; collapsed child rows list each raw category value, its canonical mapping, and a description
- **Numeric Features** - feature definitions including value ranges, meaning of binary indicators, and clarifying notes

The notes column in both tabs is intended for annotators to leave comments, flag ambiguous values, or document decisions requiring review.

This structured interface enabled consistent labeling across annotators and guaranteed that each federated client used the same category definitions and encodings.

All annotation artifacts, including the guidelines and Excel spreadsheet, are stored in the Git repository here:

https://github.com/umdnp/ari510capstone-fedlearn/tree/main/docs/annotations

These files serve as the authoritative source for preprocessing rules used across the entire modeling pipeline.

4. Benchmark Models

Our benchmarks were established using centralized models trained on the full dataset. These models provide a baseline to compare federated performance against and to validate the predictability of the task itself.

4.1. Model Selection

We trained four distinct model types, chosen to represent a range of machine learning algorithms:

Logistic Regression - A linear model offering interpretability and probabilistic outputs. We configured logistic regression with class_weight="balanced" to account for the 75/25 class imbalance and used the lbfgs solver with 1,000 maximum iterations. This model provided a strong interpretable baseline (F1 = 0.661, training time ~26 seconds) and served as the foundation for our Codabench competition baseline. Coefficient weights allowed us to identify which clinical features most strongly influenced predictions, making this model valuable for clinical validation and feature selection.

Random Forest - A collection of decision trees that balances accuracy with training efficiency. We configured the forest with 100 trees, maximum depth of 10, and minimum leaf sizes of 10 to prevent overfitting on the training set. This model achieved the best F1 score (0.680) among all centralized approaches while training in only 4.3 seconds—faster than logistic regression. The speed-performance trade-off made Random Forest our recommended choice for production deployment, and its built-in feature importance scores provided clinical insights without sacrificing predictive power.

Gradient Boosting - A sequential ensemble method that iteratively corrects prediction errors. We used 100 boosting stages with a learning rate of 0.1 and maximum depth of 5 per tree. Gradient boosting achieved the highest overall accuracy (0.797) and demonstrated that the prolonged-stay prediction task has substantial learnable signal. However, training required 128 seconds—approximately 30× slower than Random Forest—and the model showed sensitivity to hyperparameter choices. Despite strong performance, the computational cost made this model less practical for real-time or iterative experimentation.

SGDClassifier - A linear model trained via stochastic gradient descent, supporting incremental learning through the partial_fit interface. We configured SGD with log loss (equivalent to logistic regression), L2 regularization, and balanced class weights. This model underperformed relative to others (F1 = 0.516, accuracy = 0.550) and required careful tuning to avoid convergence warnings. However, SGD was essential for federated learning experiments, as it is one of the few scikit-learn classifiers supporting incremental updates required for distributed training. Its inclusion allowed us to directly compare centralized and federated performance using the same underlying algorithm.

These models vary widely in training speed, interpretability, and capacity, allowing us to evaluate trade-offs between performance and practical deployment considerations.

4.2. Training Configuration

All models were trained on the full training set (119,787 samples) and evaluated on the held-out test set (39,930 samples). Key hyperparameters were tuned using the validation set (39,929 samples) to balance bias-variance trade-offs and prevent overfitting. Training was performed on a high-performance workstation featuring a 24-core Intel i9 CPU, 128 GB DDR5 RAM, and an NVIDIA 3090 Ti GPU (24 GB), providing substantial computational resources for model development.

Class imbalance (75.2% normal stays, 24.8% prolonged stays) was addressed using class_weight="balanced" for linear models and stratified sampling for tree-based ensembles, ensuring that both classes received appropriate attention during training.

4.3. Public Benchmark: Codabench Competition

To support reproducibility and community engagement, we created a public benchmark competition hosted on Codabench:

Competition URL: https://www.codabench.org/competitions/12116/

Title: ARI 510 - ICU Prolonged Stay Prediction Challenge

The competition uses a subset of the eICU dataset (2,520 samples: 1,512 training, 1,008 test) and provides participants with:

- Training data with ground-truth labels
- Test data with labels withheld
- Automated scoring using F1-score (macro-averaged), accuracy, precision, and recall
- A baseline Logistic Regression model achieving F1 = 0.7547 on the test set

The Codabench baseline outperformed our full-dataset results (F1 = 0.661 for centralized Logistic Regression), likely because the demo dataset is a cleaner, more curated subset with fewer missing values and better feature coverage. This competition allows external researchers to evaluate their approaches and compare against our baseline using standardized metrics and data splits.

All competition materials, including the starting kit, scoring program, and data preparation scripts, are available in our repository:

https://github.com/umdnp/ari510capstone-fedlearn/tree/main/codabench_bundle

4.4. Benchmark Summary

The benchmark models established that prolonged ICU stay prediction is a learnable task with moderate-to-strong performance achievable using traditional machine learning. These results provided confidence that federated learning could achieve meaningful performance despite the constraints of distributed training and limited model choices. Detailed results for each model are presented in Section 5.1.

5. Model Development and Results

5.1. Centralized Machine Learning Results

Below is a summarized version of the final test-set metrics:

Model	Accuracy	Precision	Recall	F1
Logistic Regression	0.713	0.655	0.692	0.661
Random Forest	0.731	0.671	0.709	0.679
Gradient Boosting	0.797	0.738	0.658	0.678
SGD Classifier	0.550	0.543	0.558	0.516

Key Takeaways:

1. **Gradient Boosting** achieved the strongest overall performance (≈0.80 accuracy) but required the longest training time (~130–145 seconds).
2. **Random Forest** performed well and trained quickly (~4.3 seconds), making it an attractive practical choice.
3. **Logistic Regression** provided stable, interpretable performance but required more iterations to converge.
4. **SGDClassifier** underperformed consistently, even though it is suitable for large-scale learning.

The centralized models show that predicting prolonged ICU stay is a learnable task, with traditional ML models achieving moderate to strong accuracy.

5.2. Federated Learning Results (Flower Framework)

We implemented FedAvg with three clients, each representing a region-based partition of hospitals. Each client trained an SGDClassifier, chosen because:

- SGD supports incremental updates suitable for federated settings
- Logistic Regression does not support `partial_fit`, making it unsuitable for FL without custom work

- SGD models align with Flower tutorials and expected workflows

Training Summary (5 rounds)

- Training accuracy across rounds ranged from 0.56 - 0.60
- Evaluation accuracy across rounds ranged from 0.53 - 0.62
- Final aggregated evaluation accuracy: ~0.61

Observations and Lessons Learned

1. Federated SGD performed about the same as centralized SGD and even slightly better, suggesting that averaging updates across clients may have helped the model generalize more effectively.
2. Federated learning remained competitive despite model constraints. We used SGD because it supports `partial_fit`, but other incremental-learning models (such as Perceptron or Passive-Aggressive Classifiers) could also be used in a federated setup and may offer stronger performance.
3. The low number of rounds (5) limited convergence; additional rounds would likely improve stability and accuracy.
4. Convergence warnings indicate that tuning SGD hyperparameters (learning rate, max_iter) could strengthen local training on each client.
5. Overall, federated learning delivered performance comparable to centralized SGD, showing that distributing training across clients did not degrade model quality and may even provide a mild regularizing benefit.
6. These results suggest that federated learning can achieve centralized-level performance while preserving data locality, which is valuable in settings where data sharing is restricted.

What Worked Well

- Flower successfully coordinated all clients
- Model aggregation occurred without failures
- Training remained stable despite Ray warnings (non-critical)

What Needs Improvement

- Increase the number of training rounds (20–50+)
- Explore more expressive models that support `partial_fit`

- Consider methods like FedProx or weighted FedAvg if client data distributions become more heterogeneous.

6. Deployment

Per the project requirements, the final system was packaged into a Streamlit application that allows users to:

- Select a patient record with a slider
- View the true prolonged-stay label
- Switch between models
    - Logistic Regression
    - Random Forest
    - Gradient Boosting
    - SGD Classifier
    - Federated (Flower SGD)
- View predicted probabilities and classification outcomes
- Compare centralized vs. federated predictions interactively

The application is available in our Git repository here: streamlit_app.py

To launch the app, run:

```
streamlit run src/fedlearn/app/streamlit_app.py
```

Note that the app runs inside the project's virtual environment and requires the dependencies listed in requirements.txt.

7. Discussion and Conclusions

7.1. Centralized vs. Federated Learning

	Centralized Learning	**Federated Learning**
Accuracy	High (0.73–0.80)	Moderate (~0.61)
Model Choice	Any ML model	Models with incremental updates only
Data Requirements	Requires full data access	Preserves data locality/privacy

Training Speed	Fast for RF/SGD, slower for GB	Slower overall due to coordination
Real-World Feasibility	Not always possible (privacy)	Very relevant for healthcare settings

* These accuracy ranges compare the strongest centralized models to federated SGD. When comparing SGD to SGD, federated performance was slightly higher than centralized.

7.2. Why Centralized Models Performed Better Overall

Although federated learning is often expected to perform worse than centralized training, our results showed that federated SGD performed slightly better than centralized SGD. This indicates that the federated averaging process may have provided a regularizing effect, helping the model generalize more effectively.

The main reason centralized models achieved higher overall accuracy was not due to federation, but due to model choice: centralized training was able to use more powerful classifiers such as Random Forest and Gradient Boosting, which generally outperform linear models like SGD. Because federated learning requires incremental updates (partial_fit), we were limited to SGD in the federated setting.

In short, the gap in performance was driven by model capability, not by the federated learning process itself. When comparing the same model class (SGD), federated learning was competitive and even slightly superior.

7.3. Overall Findings

1. Prolonged ICU length of stay is predictable using early-stay features, with centralized Gradient Boosting achieving the strongest performance.
2. Federated SGD matched, and slightly exceeded, centralized SGD, showing that distributing training across clients did not reduce model quality.
3. Model choice, not federation, explained the broader performance gap, since centralized training could use more powerful classifiers while federated learning was limited to incremental models like SGD.
4. Federated learning proved feasible for this task, successfully training across multiple clients without sharing raw patient data.
5. Even with only five training rounds, the federated model converged well enough to match centralized SGD, suggesting additional rounds could improve performance further.

8. Future Work

- Train more federated rounds (20–50+)
- Implement a neural network with partial_fit or custom PyTorch FL model
- Explore FedProx, FedNova, or Scaffold to improve convergence
- Expand feature engineering beyond early-stay features
- Evaluate model fairness across demographic groups
- Explore privacy-preserving techniques

9. Resources

- ARI 410-510 Final Project Guidelines (PDF)
- eICU Collaborative Research Database (Data)
- eICU Collaborative Research Database (Data Dictionary)
- GitHub Repository (ari510capstone-fedlearn)