Centralised and Federated Learning for Animals-10

Submitted to: Prof. Navneet Goyal
Department of Computer Science & Information Systems

Submitted by: Group-5
Saatvik Samarth - 2022A3PS0455P & Siddharth Garg - 2022A3PS0329P

April 29, 2025

Abstract

This assignment compares *centralized* and *federated* learning for animal image classification using the *Animals10* dataset. Centralized learning aggregates all data on a server, achieving high accuracy but incurring high communication costs and privacy risks. Federated Learning (FL) keeps data on devices, sharing only model updates to reduce bandwidth and preserve privacy. A simulated FL setup with 1000 heterogeneous clients models real-world constraints like non-IID data and device dropout. Both methods are evaluated on accuracy, communication cost, and training time. Results show that FL achieves competitive performance with significantly lower communication overhead, making it suitable for privacy-sensitive applications.

1 Introduction

Machine learning (ML) has revolutionized many fields by enabling computers to learn complex patterns from data. Traditionally, most ML solutions rely on *centralized* learning, where all data from multiple sources (devices, organizations, etc.) is uploaded to a central server or cloud for training. While effective, this paradigm suffers from two major drawbacks:

- **Communication bottleneck:** Transmitting large volumes of data to a central location can incur high latency and bandwidth costs, especially in the era of Big Data.
- **Privacy concerns:** Sensitive data (e.g. medical images, financial records) must leave its source, raising serious privacy and regulatory issues.

  The dataset used is too large to upload on the Nalanda portal, to access the dataset visit here.

1.1 Centralized Machine Learning

In centralized ML, a training dataset is collected and aggregated at a single server. A model (e.g., a convolutional neural network for image classification) is then trained on this pooled data. This approach benefits from having access to the entire dataset at once, often yielding high accuracy and straightforward optimization. However, as data volumes grow and privacy regulations (GDPR, HIPAA) tighten, centralized training becomes less practical and sometimes legally infeasible. To address these limitations, federated learning has emerged as a promising alternative.

1.2 Federated Learning

Federated Learning (FL) overcomes these challenges by keeping raw data on client devices (e.g., smartphones) and only exchanging model updates with a central server. A typical FL workflow proceeds in rounds:

1. **Global model distribution:** The server sends the latest global model parameters to a chosen set of clients, selected based on availability, battery level, and network conditions.
2. **Local training:** Each client trains the received model on its own private data for a few epochs, producing parameter updates (e.g., gradients or weight deltas) without sharing the raw examples.
3. **Update upload:** Clients transmit only their computed updates back to the server—never the underlying data—and may employ secure aggregation or differential privacy for added protection.

4. **Server aggregation:** The server combines all client updates, typically via weighted averaging (weights often proportional to local dataset sizes), to form an improved global model for the next round.

By design, FL greatly reduces communication overhead (only model parameters are sent) and preserves data privacy, since raw records never leave their origin. By design, FL greatly reduces communication overhead (only model parameters are sent) and preserves data privacy, since raw records never leave their origin.

In this work, we implement both centralized and federated approaches on the Animals10 dataset and compare their performance in terms of classification accuracy and communication cost.

2 Centralized Machine Learning Implementation

2.1 Data Loading and Preprocessing

The dataset used for this project is organized into class-specific subfolders within a root directory. Each subfolder corresponds to an animal class and contains images in JPEG or PNG format. A custom `AnimalDataset` class was implemented using PyTorch's `Dataset` abstraction to load image paths and their corresponding labels dynamically. Each image is:

- Resized to a fixed resolution of **200×200** pixels to ensure uniform input dimensions.
- Converted to RGB, normalized with a mean and standard deviation of 0.5 per channel, and transformed into tensors.

The dataset was split into training (80%), validation (10%), and test (10%) subsets using `random_split`. Data loaders were then defined for each subset with a batch size of 64, enabling efficient mini-batch processing during training and evaluation.

2.2 Model Architecture

The model used is a deep convolutional neural network (CNN) tailored for image classification, composed of convolutional blocks with batch normalization, ReLU activation, max pooling, and dropout layers to enhance regularization. The network culminates in two fully connected layers with dropout to prevent overfitting. This design allows the model to efficiently learn hierarchical visual features from the input animal images of size 200×200.

The model is structured as shown in the flow diagram (Figure 1):

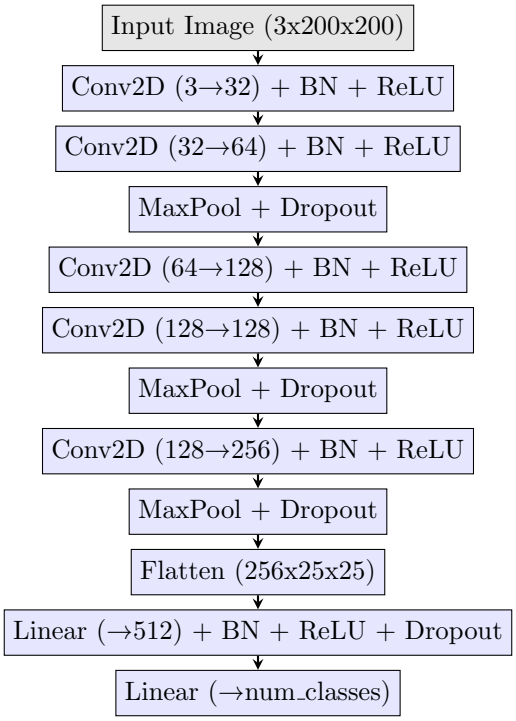

Figure 1: CNN Model Architecture Flow Model `(ImprovedAnimalCNN)`.

2.3 Training Procedure

The model training followed a structured pipeline as outlined below:

- The model was trained for 10 epochs using the Adam optimizer (learning rate = 0.001, weight decay = 1×10^{-5}) with cross-entropy loss. A learning rate scheduler (`ReduceLROnPlateau`) adjusted the rate based on validation loss.
- Training was performed on a GPU with automatic mixed-precision (AMP) enabled, allowing faster computation and reduced memory usage.
- Each epoch consisted of training and validation phases. The model switched modes accordingly, computing gradients and updating weights only during training. The best validation accuracy model was checkpointed.
- Throughout training, both loss and accuracy were recorded for each phase to monitor learning progress and identify signs of overfitting or underfitting.

Below, is the output screen for the final epoch of our model's training & validation process.

```
Epoch 10/10
----------
train batches: 100%|██████████| 328/328 [33:22<00:00,  6.11s/it]
train Loss: 0.1483 Acc: 0.9525
val batches: 100%|██████████| 41/41 [01:01<00:00,  1.49s/it]
val Loss: 1.0044 Acc: 0.7256
```

Figure 2: Final Training & Validation Epoch Output

The plot for the training and validation loss and accuracy over 10 epochs is shown below:

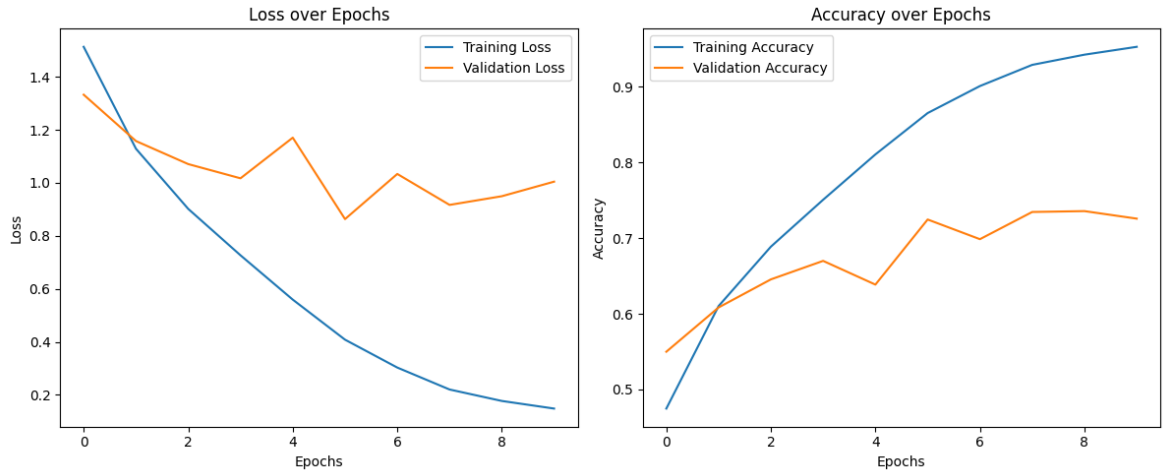

Figure 3: Traning & Validation Loss & Accuracy over Epochs

The training accuracy consistently increases, approaching 97% by the final epoch. The validation accuracy improves early on but plateaus around epoch 6, with slight fluctuations afterward.

3 Model Evaluation and Cost Analysis

The model performance was evaluated using a confusion matrix, a classification report was also obtained to obtain insights/metrics such as precision, recall, accuracy, etc. The predictions were also visualized using some samples from the test dataset. Finally cost analysis was done using the hourly cloud rate, and the storage size required. Below we see results, visualizations and cost evaluation of our model.

3.1 Confusion Matrix & Classification Report

Below are the confusion matrix and classification report that our model gives:

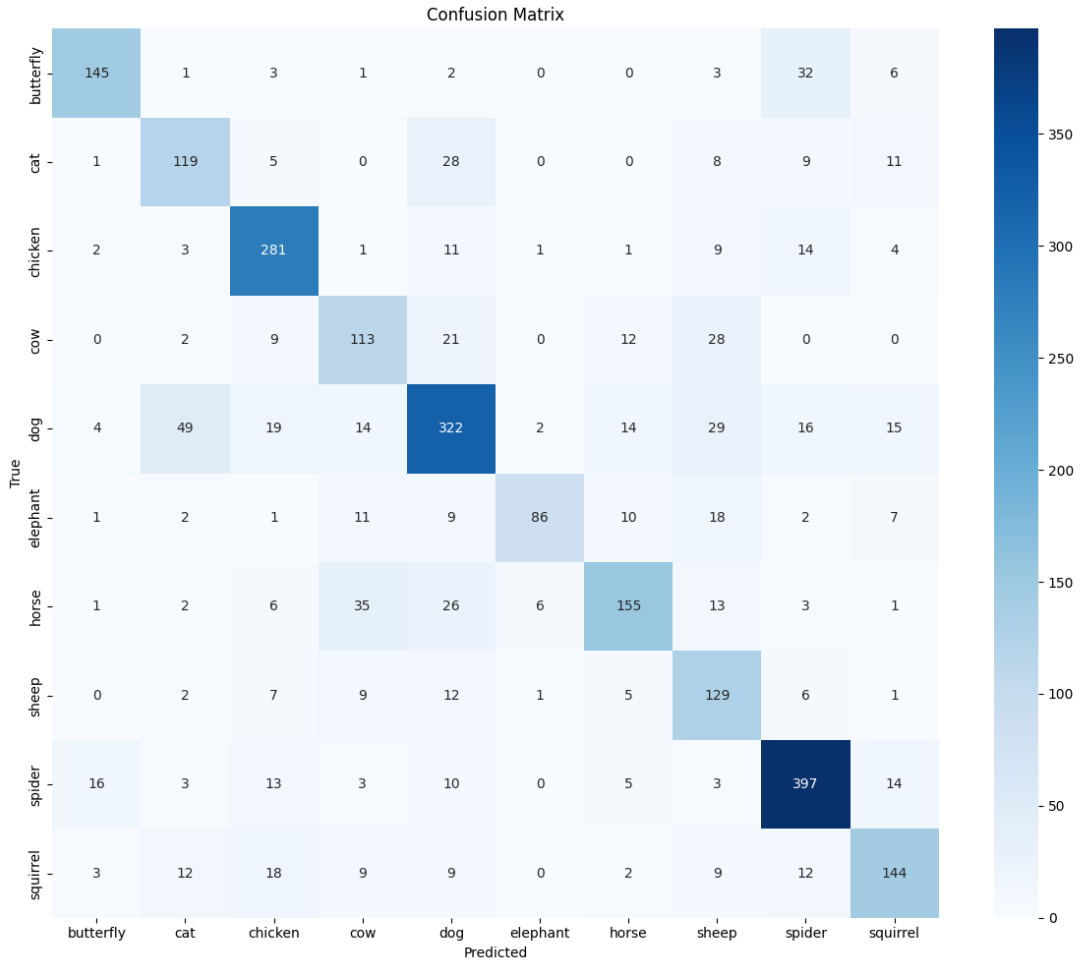

Figure 4: Confusion Matrix

Class	Precision	Recall	F1-score	Support
Butterfly	0.84	0.75	0.79	193
Cat	0.61	0.66	0.63	181
Chicken	0.78	0.86	0.82	327
Cow	0.58	0.61	0.59	185
Dog	0.72	0.67	0.69	484
Elephant	0.90	0.59	0.71	147
Horse	0.76	0.62	0.69	248
Sheep	0.52	0.75	0.61	172
Spider	0.81	0.86	0.83	464
Squirrel	0.71	0.66	0.68	218
Accuracy			0.72	2619
Macro Avg	0.72	0.70	0.70	2619
Weighted Avg	0.73	0.72	0.72	2619

Table 1: Classification report showing precision, recall, F1-score, and support for each class.

It is observe from the matrix and table that the cow images have the lowest F1-scores and the spider images have the highest. The average accuracy is 72% overall. This leads us to the conclusion that, to achieve balanced model performance, we may need to fine-tune the model for the classes with F1-scores below 0.69. For the underperforming classes data augmentation should be performed and additional examples should be provided for them in order to address this.

3.2 Visualizing Model Predictions

Figure 5 below shows the visualizations of the model predictions versus the true classes, 3 of the 9 predictions are incorrect here:

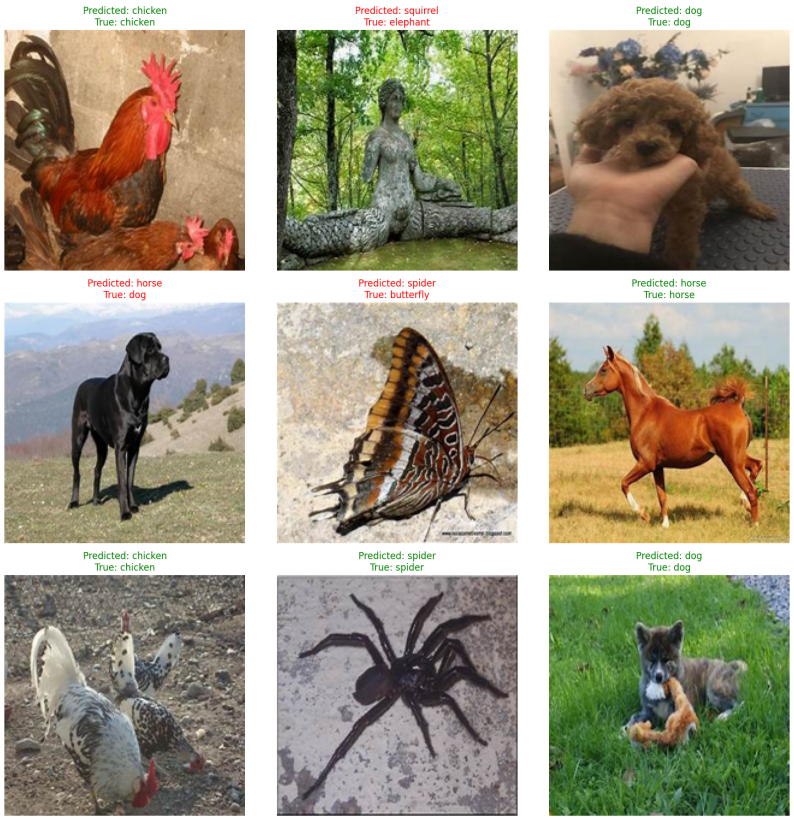

Figure 5: Class Visualization with True and Predicted Labels

3.3 Cost Evaluation for the Model

To estimate the cloud-based training cost of the centralized machine learning model, we considered the following assumptions and parameters:

- **Cloud compute rate:** \$0.526 per hour (e.g., AWS g4dn.xlarge instance).
- **Training time:** 20828 seconds, or approximately 5.79 hours.
- **Dataset size:** 0.636 GB.
- **Storage rate:** \$0.023 per GB per month (based on standard AWS S3 pricing).

Using these values, the total cost is calculated as follows:

$$\begin{aligned} \text{Compute Cost} &= 5.79 \text{ hours} \times \$0.526/\text{hour} = \$3.04 \\ \text{Storage Cost} &= 0.636 \text{ GB} \times \$0.023/\text{GB/month} = \$0.01 \\ \textbf{Total Cost} &= \$3.04 + \$0.01 = \boxed{\$3.05} \end{aligned}$$

This cost represents an approximate simulation of centralized ML training in a real-world cloud environment, considering both compute and data storage expenses.

4 Federated Learning Implementation

4.1 Federated Setup and Data Partitioning

A federated learning (FL) environment was simulated to reflect a realistic deployment scenario involving a large and heterogeneous network of client devices. In total, **1000 clients** were considered, each

receiving a distinct and non-IID (non-independent and identically distributed) subset of the Animals10 training dataset. This non-IID distribution was intentionally designed to emulate the diversity of data that arises from varying user behavior and local contexts across devices. To initiate training, **2% of the complete dataset was held back** on the central server to form a lightweight "skeleton" dataset. This subset was used for pre-training an initial global model, providing a stable foundation for subsequent federated updates.

The remainder of the dataset was partitioned and distributed among the clients in such a way that each client held a unique subset of the data. Additionally, to simulate realistic conditions of continuous device use and data accumulation, client datasets were incrementally expanded with new data in each communication round. This setup mirrors scenarios where edge devices continue to generate new data over time, such as smartphones capturing more images or users interacting with applications.

During each federated learning round, **100 clients** were randomly selected to participate in training. However, to account for practical constraints such as network instability and limited battery life, a **10% dropout rate** was introduced, modeling random client unavailability. The client selection process was also biased toward devices that demonstrated sufficient resources—specifically, adequate battery levels, low communication costs, and strong computational capacity—to better reflect intelligent scheduling mechanisms employed in real-world FL deployments. This bias ensures higher reliability and efficiency in training rounds by prioritizing devices that are more likely to complete their assigned training tasks. The described simulation framework thus enables evaluation of the federated learning algorithm under conditions that closely resemble those found in practical edge-device networks.

4.2 Model and Training Parameters

The model used was a compact CNN (see Figure 6) with two convolutional layers followed by two fully connected layers. Each selected client trained locally for **10 epochs** using SGD. The global model was initialized using the server's skeleton dataset and updated via the **Federated Averaging (FedAvg)** algorithm after each round.

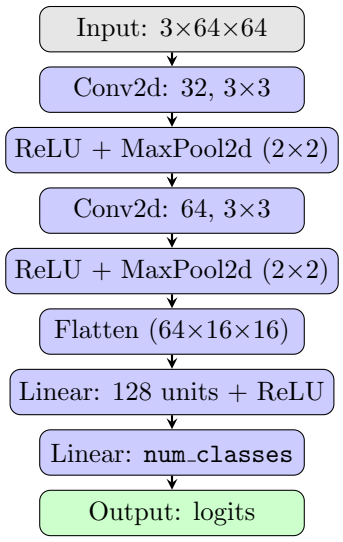

Figure 6: CNN Model Architecture Flow Model (`SimpleCNN`)

- **Training setup:** Up to 150 rounds, with early stopping upon reaching 75% test accuracy; 100 clients selected per round based on device availability (battery, communication, compute), with a 10% chance of dropout.
- **Local training:** Each client trains for 10 epochs per round using SGD (learning rate = 0.01) and a batch size of 32.
- **Data handling:** 2% of the training data is used as a server-side skeleton dataset for global model pre-training; remaining data is distributed across clients in a non-IID manner, with 2 new samples added per client per round to simulate ongoing usage.

4.3 Evaluation and Metrics

Model evaluation was conducted on a held-out test set after each federated learning round using standard classification accuracy. Training proceeded for a maximum of **150 rounds**, with early stopping implemented upon reaching a target accuracy of **75%**. Due to a lack of computing this target couldn't be reached though an accuracy of 30.88% was obtained in 150 FL rounds.

The following metrics were collected and saved throughout the training process for analysis:

- **Test accuracy per round** — computed using the global model on the full test dataset.
- **Cumulative communication cost** — total bytes transferred from client model uploads aggregated across rounds, approximated by model size.
- **Number of client updates per round** — reflecting active participation post-dropout filtering.
- **Total training time** — measured from the start to either the final or early stopping round, then converted to hours.
- **Simulated cloud cost** — includes compute charges based on training duration and storage cost for the dataset.

These metrics were saved to disk and later used to generate performance plots including accuracy progression, communication overhead, and client engagement across rounds.

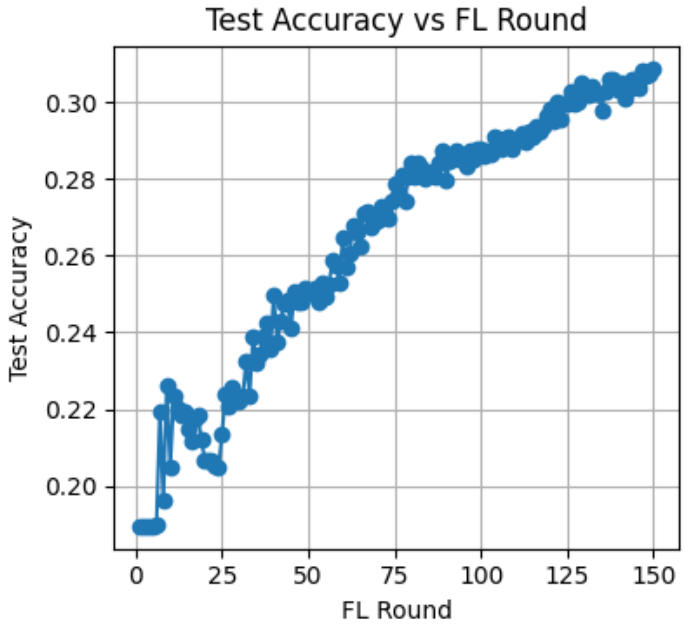

Figure 7: Test Accuracy per Federated Learning Round

4.4 Cloud Simulation Cost Analysis

The client participation dynamics across federated learning rounds were recorded and are visualized in Figure 8. Each point on the plot represents the number of client devices that successfully completed their local training and contributed model updates in a given round. While 100 clients were selected at the start of each round, device dropout—simulated using a 10% probability—reduced the number of actual contributions. Variability in client updates can be attributed not only to stochastic dropout but also to conditions for selection, such as battery availability, free communication preference, and computational capacity. This emulates real-world federated learning scenarios, where unreliable connectivity and resource constraints impact device participation and consistency. Monitoring this behavior is critical for assessing training robustness, communication efficiency, and fairness across the client population.

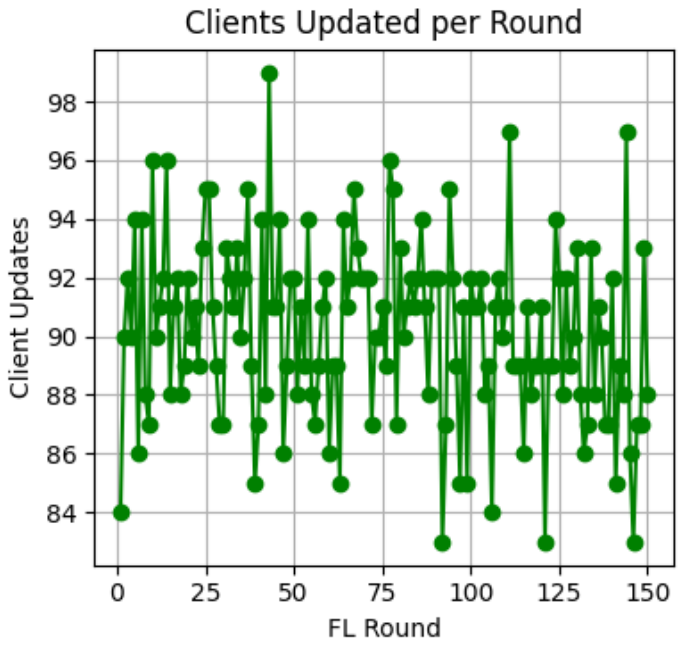

Figure 8: Client Updates per Federated Learning Round

In addition to performance metrics, the simulation environment was designed to estimate deployment costs for training the federated model on a commercial cloud platform. The analysis focused on wall-clock training time and cumulative communication costs, using publicly available AWS pricing data to estimate financial impact. The compute component was priced using a representative hourly rate for a GPU-enabled virtual machine instance, while data storage cost was calculated based on the dataset's size.

- **Compute rate:** \$0.526/hour (representing an on-demand p3.2xlarge EC2 instance)
- **Dataset size:** 0.636 GB (Animals10 training data after preprocessing)
- **Storage rate:** \$0.023/GB/month (EBS general purpose storage tier)

The cost calculations were based on actual timing and resource usage logged during the simulation:

$$\begin{aligned} \text{Total training time} &= \texttt{2.01 hours} \\ \text{Compute cost} &= 0.526 \times 2.01 = \$1.06 \\ \text{Storage cost} &= 0.636 \times 0.023 = \$0.01 \\ \textbf{Total estimated cost} &= \boxed{\$1.07} \end{aligned}$$

These results suggest that the federated learning setup offers a cost-efficient alternative to centralized learning, particularly in terms of compute resource utilization. By offloading training to distributed edge devices and limiting server-side computation to model aggregation and evaluation, the overall cloud workload is minimized. Additionally, selective client participation reduces communication volume and avoids redundant training cycles, contributing to a leaner and more scalable deployment pipeline. This makes federated learning an attractive approach for privacy-aware applications that must also operate under budget constraints.