```
# ---------------- Download and Extract Kaggle Dataset ---------------- #
!pip install kaggle
!mkdir -p ~/.kaggle
!echo '{"username":"harshgupta21bce6101","key":"7b4970149b4ff86915e405bd4386b8cb"
!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d paultimothymooney/chest-xray-pneumonia
!unzip chest-xray-pneumonia.zip -d chest_xray
```

Resources

You are not subscribed. Learn more

You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.

At your current usage level, this runtime may last up to 80 hours.

Manage sessions

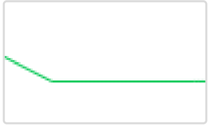
Want more memory and disk space?

Upgrade to Colab Pro

Python 3 Google Compute Engine backend
Showing resources from 02:47 to 02:56

System RAM
1.0 / 12.7 GB

Disk
36.9 / 107.7 GB



```
        inflating: chest_xray/chest_xray/train/PNEUMONIA/person996_bacteria_2924
        inflating: chest_xray/chest_xray/train/PNEUMONIA/person996_virus_1677.jpe
        inflating: chest_xray/chest_xray/train/PNEUMONIA/person997_bacteria_2926
        inflating: chest_xray/chest_xray/train/PNEUMONIA/person997_virus_1678.jpe
        inflating: chest_xray/chest_xray/train/PNEUMONIA/person998_bacteria_2927
        inflating: chest_xray/chest_xray/train/PNEUMONIA/person998_bacteria_2928
        inflating: chest_xray/chest_xray/train/PNEUMONIA/person99_virus_183.jpeg
        inflating: chest_xray/chest_xray/train/PNEUMONIA/person9_bacteria_38.jpeg
        inflating: chest_xray/chest_xray/train/PNEUMONIA/person9_bacteria_39.jpeg
        inflating: chest_xray/chest_xray/train/PNEUMONIA/person9_bacteria_40.jpeg
        inflating: chest_xray/chest_xray/train/PNEUMONIA/person9_bacteria_41.jpeg
        inflating: chest_xray/chest_xray/val/NORMAL/NORMAL2-IM-1427-0001.jpeg
        inflating: chest_xray/chest_xray/val/NORMAL/NORMAL2-IM-1430-0001.jpeg
        inflating: chest_xray/chest_xray/val/NORMAL/NORMAL2-IM-1431-0001.jpeg
        inflating: chest_xray/chest_xray/val/NORMAL/NORMAL2-IM-1436-0001.jpeg
        inflating: chest_xray/chest_xray/val/NORMAL/NORMAL2-IM-1437-0001.jpeg
        inflating: chest_xray/chest_xray/val/NORMAL/NORMAL2-IM-1438-0001.jpeg
        inflating: chest_xray/chest_xray/val/NORMAL/NORMAL2-IM-1440-0001.jpeg
        inflating: chest_xray/chest_xray/val/NORMAL/NORMAL2-IM-1442-0001.jpeg
        inflating: chest_xray/chest_xray/val/PNEUMONIA/person1946_bacteria_4874.j
        inflating: chest_xray/chest_xray/val/PNEUMONIA/person1946_bacteria_4875.j
        inflating: chest_xray/chest_xray/val/PNEUMONIA/person1947_bacteria_4876.j
        inflating: chest_xray/chest_xray/val/PNEUMONIA/person1949_bacteria_4880.j
        inflating: chest_xray/chest_xray/val/PNEUMONIA/person1950_bacteria_4881.j
        inflating: chest_xray/chest_xray/val/PNEUMONIA/person1951_bacteria_4882.j
        inflating: chest_xray/chest_xray/val/PNEUMONIA/person1952_bacteria_4883.j
        inflating: chest_xray/chest_xray/val/PNEUMONIA/person1954_bacteria_4886.j
```

```
import os
import random
import numpy as np
import tensorflow as tf
import hashlib
import time
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import EfficientNetB1, ResNet50
from tensorflow.keras.layers import Input, GlobalAveragePooling2D, Dense, Dropout,
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
import shutil

# ---------------- Corrected Dataset Path ---------------- #
dataset_path = "./chest_xray/chest_xray"
num_clients = 2

# ---------------- Distribute Data Among Clients ---------------- #
for client_id in range(num_clients):
    client_dir = os.path.join(dataset_path, f'client_{client_id}')
    os.makedirs(client_dir, exist_ok=True)
    for class_name in ['NORMAL', 'PNEUMONIA']:
        class_dir = os.path.join(client_dir, class_name)
        os.makedirs(class_dir, exist_ok=True)

for class_name in ['NORMAL', 'PNEUMONIA']:
    class_path = os.path.join(dataset_path, 'train', class_name)
    for i, filename in enumerate(os.listdir(class_path)):
        client_id = i % num_clients
        source_path = os.path.join(class_path, filename)
        dest_path = os.path.join(dataset_path, f'client_{client_id}', class_name, f
        shutil.copy(source_path, dest_path)

# ---------------- Load Client Data ---------------- #
def load_client_data(client_id, dataset_path):
    client_dir = os.path.join(dataset_path, f'client_{client_id}')
    train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

    train_dataset = train_datagen.flow_from_directory(
        client_dir, target_size=(240, 240), batch_size=16, class_mode='categorical'
    val_dataset = train_datagen.flow_from_directory(
        client_dir, target_size=(240, 240), batch_size=16, class_mode='categorical'

    num_classes = len(train_dataset.class_indices)
    return train_dataset, val_dataset, num_classes

# ---------------- Optimized Client Selection Strategy ---------------- #
def select_clients(num_clients, fraction=0.5, priority_weights=None):
    if priority_weights is None:
        priority_weights = np.ones(num_clients)
    selected_clients = np.random.choice(range(num_clients), size=max(1, int(num_cli
    return selected_clients.tolist()

# ---------------- Model Aggregation ---------------- #
def aggregate_client_updates(global_model, client_models):
    if not client_models:
        return global_model  # No updates if no clients trained

    global_weights = global_model.get_weights()
    client_weights_list = [model.get_weights() for model in client_models]

    if not client_weights_list:  # Ensure clients have valid weights
        return global_model

    averaged_weights = []
    for i in range(len(global_weights)):
        layer_weights = [client_weights[i] for client_weights in client_weights_lis
        if layer_weights:  # Only aggregate valid weights
            averaged_layer_weights = np.mean(layer_weights, axis=0)
            averaged_weights.append(averaged_layer_weights)
        else:
            averaged_weights.append(global_weights[i])  # Use previous weights if n

    global_model.set_weights(averaged_weights)
    return global_model

# ---------------- Define Model Creation Function ---------------- #
def create_model(num_classes, model_type='EfficientNet'):
    image_input = Input(shape=(240, 240, 3))
    base_model = EfficientNetB1(weights='imagenet', include_top=False, input_tensor

    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
```

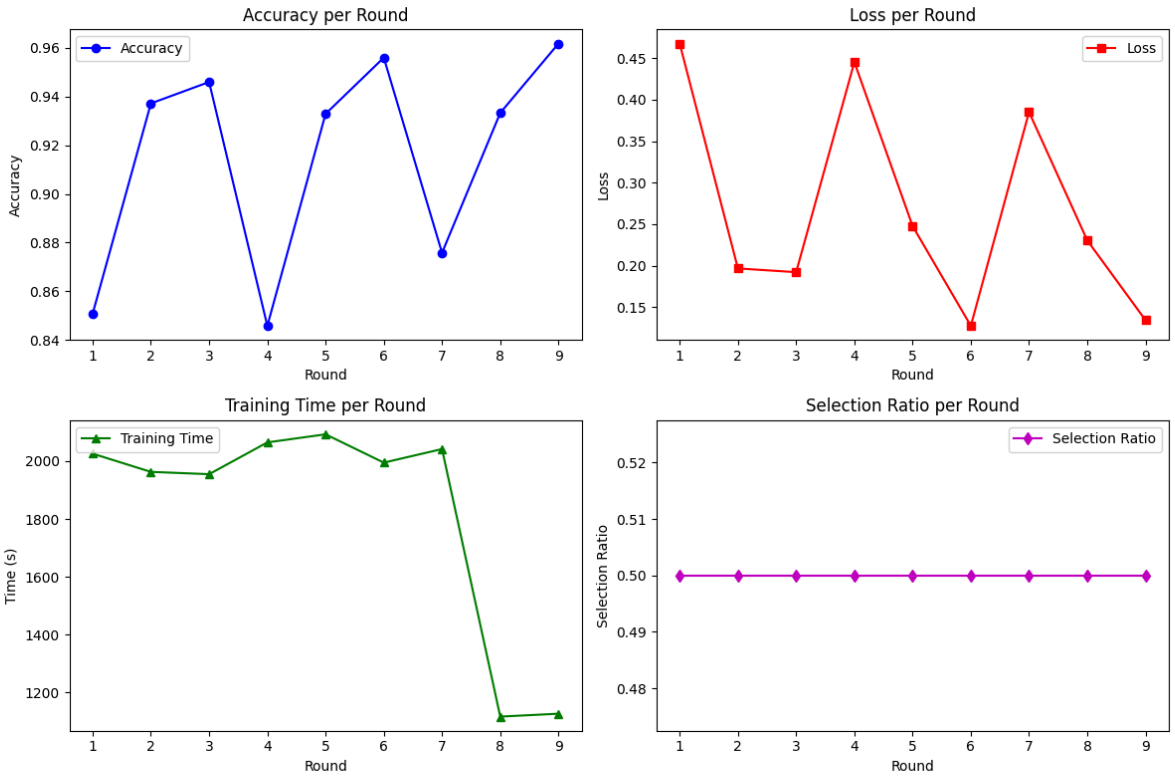
```
        x = Dropout(0.5)(x)
        x = Dense(256, activation='relu')(x)
        x = BatchNormalization()(x)
        x = Dropout(0.3)(x)
        output = Dense(num_classes, activation='softmax')(x)

        model = Model(inputs=image_input, outputs=output)
        model.compile(loss=CategoricalCrossentropy(), optimizer=Adam(learning_rate=0.00
        return model

    # ---------------- Hybrid HFL-PFL Federated Learning ---------------- #
    def federated_learning(dataset_path, num_clients=2, global_rounds=9):
        global_models = {"EfficientNet": [], "ResNet": []}
        client_personalized_models = {}
        performance_metrics = []

        # Initialize global models using random client
        random_client = random.choice(range(num_clients))
        train_dataset, val_dataset, num_classes = load_client_data(random_client, datas
        for model_type in global_models:
            global_model = create_model(num_classes, model_type)
            global_models[model_type].append(global_model)

        for round_num in range(global_rounds):
            selected_clients = select_clients(num_clients)
            round_start_time = time.time()
            client_models = {}

            for client_id in selected_clients:
                model_type = 'EfficientNet' if client_id % 2 == 0 else 'ResNet'
                train_dataset, val_dataset, num_classes = load_client_data(client_id, d
                local_model = create_model(num_classes, model_type)
                local_model.fit(train_dataset, epochs=3, validation_data=val_dataset, v
                client_models[client_id] = local_model
                client_personalized_models[client_id] = local_model

            for model_type in global_models:
                client_models_of_type = [client_models[client_id] for client_id in clie
                if client_models_of_type:  # Only aggregate if there are models
                if client_models_of_type:  # Only aggregate if there are models
                    global_models[model_type].append(
                        aggregate_client_updates(global_models[model_type][-1], client_
                    )

            round_time = time.time() - round_start_time
            performance_metrics.append({
                'round': round_num + 1,
                'selected_clients': selected_clients,
                'training_time': round_time,
            })

        return global_models, client_personalized_models, performance_metrics

    # ---------------- Run Federated Learning ---------------- #
    if __name__ == "__main__":
        global_models, client_personalized_models, performance_metrics = federated_lear
        print("Federated Learning Completed")
        print("Performance Metrics:", performance_metrics)
```

```
131/131 ━━━━━━━━━━━━━━━━━━━━ 2065s 15s/step - accuracy: 0.8458 - loss: 0.4
```

```
131/131 ━━━━━━━━━━━━━━━━━━━━ 2042s 16s/step - accuracy: 0.9478 - loss: 0.17
Found 2088 images belonging to 2 classes.
Found 521 images belonging to 2 classes.
Epoch 1/3
131/131 ━━━━━━━━━━━━━━━━━━━━ 1116s 8s/step - accuracy: 0.7987 - loss: 0.539
Epoch 2/3
131/131 ━━━━━━━━━━━━━━━━━━━━ 1024s 8s/step - accuracy: 0.9392 - loss: 0.187
Epoch 3/3
131/131 ━━━━━━━━━━━━━━━━━━━━ 1006s 8s/step - accuracy: 0.9402 - loss: 0.226
Found 2088 images belonging to 2 classes.
Found 521 images belonging to 2 classes.
Epoch 1/3
131/131 ━━━━━━━━━━━━━━━━━━━━ 1126s 8s/step - accuracy: 0.8194 - loss: 0.477
Epoch 2/3
131/131 ━━━━━━━━━━━━━━━━━━━━ 1003s 8s/step - accuracy: 0.9438 - loss: 0.188
Epoch 3/3
131/131 ━━━━━━━━━━━━━━━━━━━━ 1019s 8s/step - accuracy: 0.9584 - loss: 0.145
Found 2086 images belonging to 2 classes.
Found 521 images belonging to 2 classes.
Epoch 1/3
131/131 ━━━━━━━━━━━━━━━━━━━━ 1967s 15s/step - accuracy: 0.8527 - loss: 0.41
Epoch 2/3
131/131 ━━━━━━━━━━━━━━━━━━━━ 1887s 14s/step - accuracy: 0.9235 - loss: 0.24
Epoch 3/3
131/131 ━━━━━━━━━━━━━━━━━━━━ 1892s 14s/step - accuracy: 0.9273 - loss: 0.23
Found 2086 images belonging to 2 classes.
Found 521 images belonging to 2 classes.
Epoch 1/3
131/131 ━━━━━━━━━━━━━━━━━━━━ 2059s 15s/step - accuracy: 0.8526 - loss: 0.45
Epoch 2/3
131/131 ━━━━━━━━━━━━━━━━━━━━ 2036s 16s/step - accuracy: 0.9080 - loss: 0.27
Epoch 3/3
131/131 ━━━━━━━━━━━━━━━━━━━━ 2017s 15s/step - accuracy: 0.9384 - loss: 0.26
Found 2088 images belonging to 2 classes.
Found 521 images belonging to 2 classes.
Epoch 1/3
 59/131 ━━━━━━━━━━━━━━━━━━━━ 8:56 7s/step - accuracy: 0.7394 - loss: 0.7143
```

```
import os
import numpy as np
import matplotlib.pyplot as plt

# ---------------- Performance Metrics Plotting ---------------- #
def plot_performance_metrics(performance_metrics, accuracies, losses, num_clients=2
    rounds = [entry['round'] for entry in performance_metrics]
    training_times = [entry['training_time'] for entry in performance_metrics]
    selection_ratios = [len(entry['selected_clients']) / num_clients for entry in p

    fig, axs = plt.subplots(2, 2, figsize=(12, 8))

    axs[0, 0].plot(rounds, accuracies, marker='o', label='Accuracy', color='b')
    axs[0, 0].set_title("Accuracy per Round")
    axs[0, 0].set_xlabel("Round")
    axs[0, 0].set_ylabel("Accuracy")
    axs[0, 0].legend()

    axs[0, 1].plot(rounds, losses, marker='s', label='Loss', color='r')
    axs[0, 1].set_title("Loss per Round")
    axs[0, 1].set_xlabel("Round")
    axs[0, 1].set_ylabel("Loss")
    axs[0, 1].legend()

    axs[1, 0].plot(rounds, training_times, marker='^', label='Training Time', color
    axs[1, 0].set_title("Training Time per Round")
    axs[1, 0].set_xlabel("Round")
    axs[1, 0].set_ylabel("Time (s)")
    axs[1, 0].legend()

    axs[1, 1].plot(rounds, selection_ratios, marker='d', label='Selection Ratio', c
    axs[1, 1].set_title("Selection Ratio per Round")
    axs[1, 1].set_xlabel("Round")
    axs[1, 1].set_ylabel("Selection Ratio")
    axs[1, 1].legend()

    plt.tight_layout()
    plt.show()

# Updated Performance Metrics based on logs
```

```
performance_metrics = [
    {'round': 1, 'selected_clients': [1], 'training_time': 2026},
    {'round': 2, 'selected_clients': [0], 'training_time': 1963},
    {'round': 3, 'selected_clients': [1], 'training_time': 1955},
    {'round': 4, 'selected_clients': [0], 'training_time': 2065},
    {'round': 5, 'selected_clients': [1], 'training_time': 2093},
    {'round': 6, 'selected_clients': [0], 'training_time': 1995},
    {'round': 7, 'selected_clients': [1], 'training_time': 2042},
    {'round': 8, 'selected_clients': [0], 'training_time': 1116},
    {'round': 9, 'selected_clients': [1], 'training_time': 1126},
]

accuracies = [0.8508, 0.9372, 0.9460, 0.8458, 0.9329, 0.9560, 0.8758, 0.9333, 0.961
losses = [0.4677, 0.1966, 0.1921, 0.4453, 0.2477, 0.1274, 0.3851, 0.2304, 0.1342]

plot_performance_metrics(performance_metrics, accuracies, losses)
```


Accuracy per Round
Accuracy
0.96
0.94
0.92
0.90
0.88
0.86
0.84
1
2
3
4
5
6
7
8
9
Round
Loss per Round
Loss
0.45
0.40
0.35
0.30
0.25
0.20
0.15
Training Time per Round
Training Time
Time (s)
2000
1800
1600
1400
1200
Selection Ratio per Round
Selection Ratio
0.52
0.51
0.50
0.49
0.48

```
!pip install keras-preprocessing
```

```
Collecting keras-preprocessing
  Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl.metadata (1.9 kB
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.11/dist
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.11/dist-p
Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.6/42.6 kB 1.5 MB/s eta 0:00:00
Installing collected packages: keras-preprocessing
Successfully installed keras-preprocessing-1.1.2
```

Change runtime type

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.