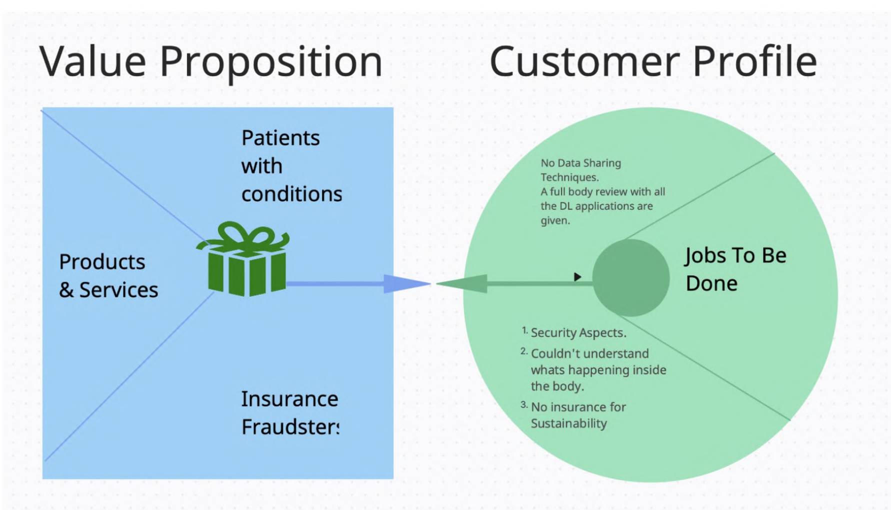
Chapter 2

Minimum Viable Product

2.1 Ideation Principles

Developing a Minimum Viable Product (MVP) involves addressing the root cause of privacy concerns in IoT devices. The focus is on countering the alarming statistic from market research, where 90% of Deep Learning (DL) models fail to reach production. To tackle issues like data leaks to pharmaceutical companies for advertisements, the MVP aims to optimize Edge devices. The strategy involves making communication lightweight, ensuring it occurs when the device is at rest, facilitating the deployment of new weights. This streamlined approach addresses core problems while laying the foundation for a robust and privacy-preserving IoT solution.



2.2 Minimum Viable Features

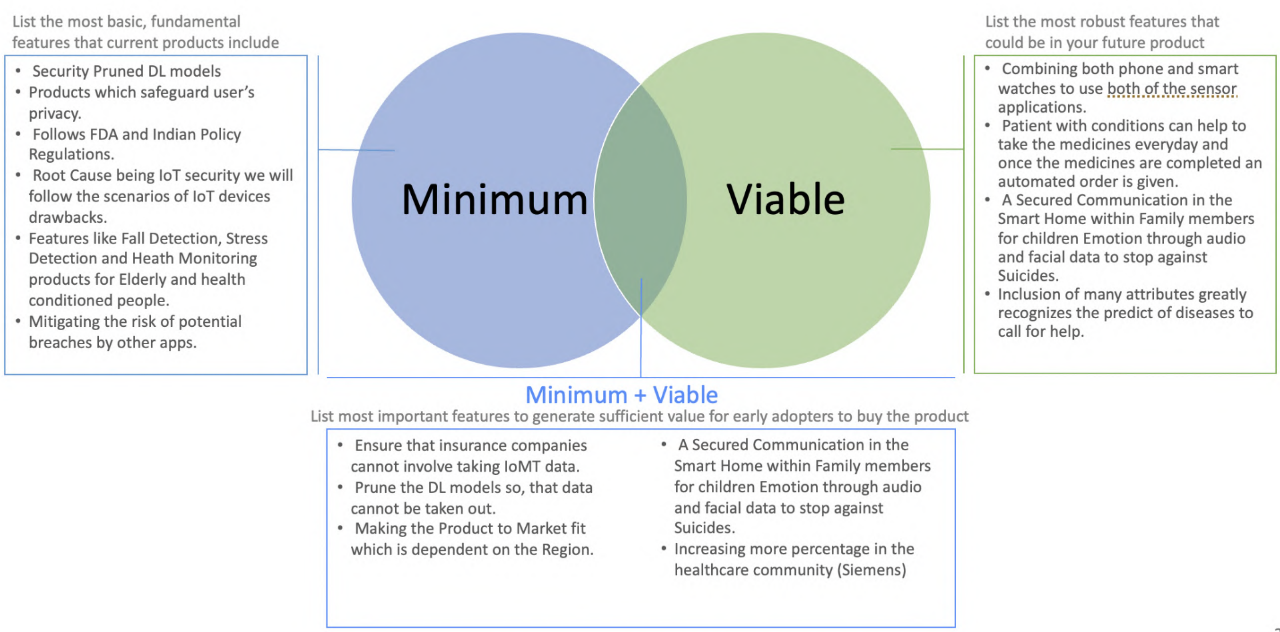
Our primary goal is to deliver value quickly, test the product in the market, and learn from user interactions. An MVP helps in minimizing development time and resources while allowing for rapid iteration based on user responses.

2.2.1 Robust with Fundamental Features

The Security Pruned DL models focus on safeguarding user privacy, adhering to FDA and Indian Policy Regulations. Addressing IoT security concerns, the product targets scenarios of IoT device drawbacks. Key features include Fall Detection, Stress Detection, and Health Monitoring for elderly and health-conditioned individuals, mitigating the risk of breaches by other apps.

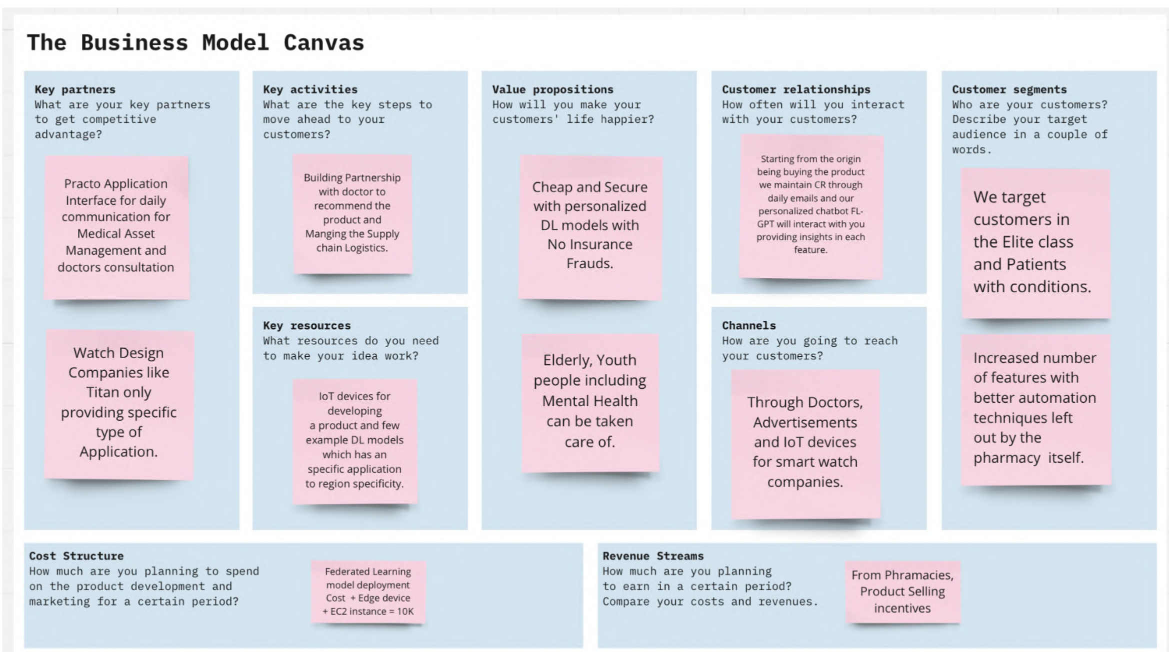
2.2.2 Minimum + Viable

For early adopters, the MVP emphasizes features crucial for generating sufficient value. Ensuring insurance companies cannot access IoT data, pruning DL models to prevent data extraction, and tailoring the product for regional market fit are paramount. The integration of both phone and smartwatch sensors for comprehensive applications, automated medicine orders for patients with conditions, and a secure communication platform within smart homes for emotion detection contribute to a robust and market-worthy product. Increasing engagement in the healthcare community, exemplified by partnerships with companies like Siemens, further strengthens the product's viability.



Chapter 3

Business Model Development



Key Partners

To gain a competitive advantage, our key partners include the Practo Application Interface for daily communication, facilitating Medical Asset Management, and enabling doctor consultations.

Key Activities

The key steps to move ahead to our customers involve building partnerships with doctors to recommend our product and efficiently managing the supply chain logistics. Additionally, designing the watch interface with companies like Titan ensures a tailored application.

Key Resources

To make our idea work, we need IoT devices for product development and a collection of example DL models with region-specific applications.

Value Propositions

We aim to make our customers' lives happier by providing a cheap and secure solution with personalized DL models, eliminating insurance fraud risks.

Customer Relationships

We maintain continuous interaction with customers, starting from product purchase. Daily emails and our personalized chatbot FL-GPT offer insights into each feature.

Customer Segments

Our target customers include the elite class, patients with conditions, and elderly individuals. We extend our services to youth, focusing on mental health care.

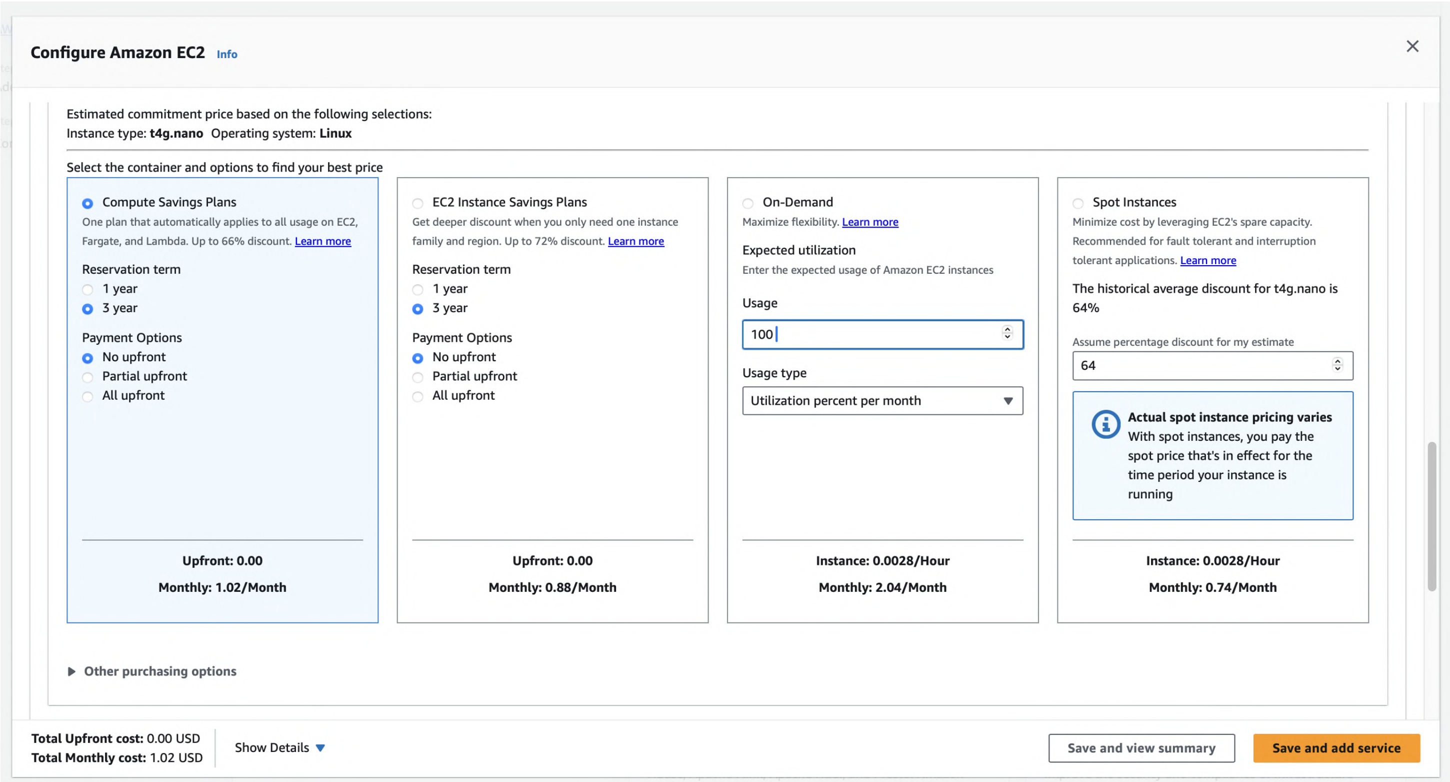
Channels

We reach our customers through doctors, advertisements, and IoT devices for smartwatch companies.

Revenue Streams

We plan to earn revenue through increased features and automation techniques, left out by pharmacies. Product selling incentives from pharmacies also contribute to our revenue streams.

3.1 Cost Structure



Chapter 4

Business Plan and Access to Funding

4.1 Executive Summary

Expanding our focus on the Internet of Medical Things, our primary target is individuals dealing with health issues and patients. However, we aim to go beyond by introducing innovative applications:

1. **Smart Doctor Integration:** Our system will not only monitor health but also act as a Smart Doctor. It will provide personalized health recommendations and facilitate seamless connections with healthcare platforms like Practo, ensuring direct delivery of prescribed medicines.

2. **Emotional Friendliness:** Recognizing the prevalence of mental health issues, especially in America, our model will incorporate emotional friendliness. By understanding and addressing negative consequences, we aspire to offer support and comfort, contributing to a holistic approach to well-being.

4.2 Company Description

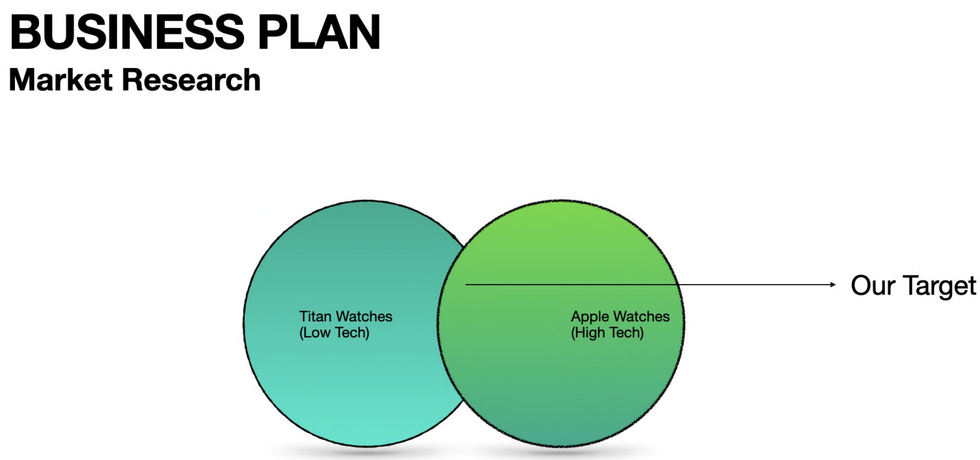
An IoMT startup working towards the goal for...

1. **Privacy Preserving DL models:** Investigating the level of privacy required for unrestricted use of an individual's health data across various applications is a critical research question. If privacy is guaranteed at 100%, what is the potential number of applications that could harness a person's health data?

2. **Continuous Learning:** Our vision involves perpetual learning models, eliminating the need for recurrent training on new data. This approach ensures that our systems stay updated and relevant without the necessity for constant retraining.

3. **Rich Personalization with Affordable Cost:** We aim to achieve extensive personalization in our IoMT applications while maintaining an affordable cost structure. This includes providing advanced features similar to premium products like the Apple Watch but at a more accessible and budget-friendly price point.

4.3 Market Research



4.4 Competitive Analysis

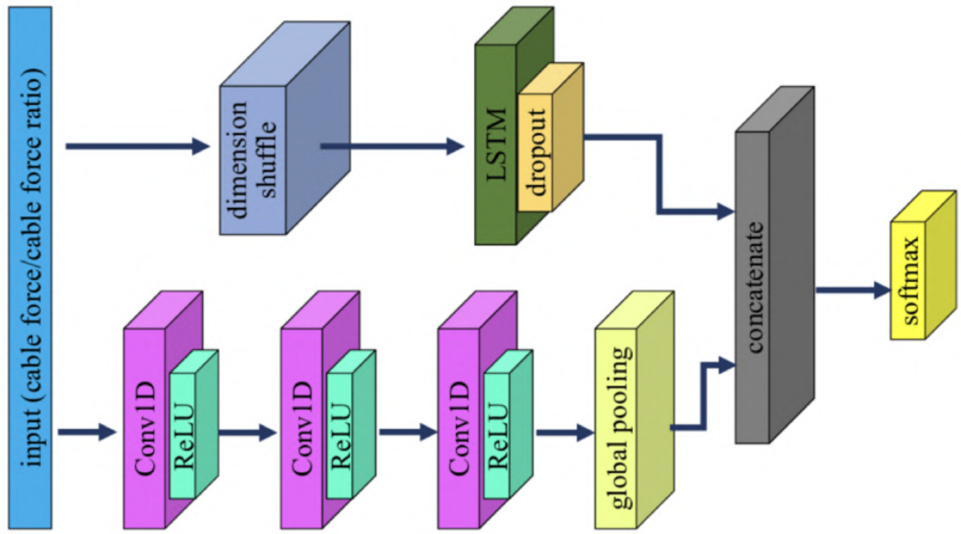
In order for our product to survive in the market, we adopt a strategic approach:

1. **Focus on Healthcare Professionals:** Our initial emphasis is on doctors, and we invest in compensating them to endorse and recommend our devices. This strategic partnership ensures that our main goal of providing essential healthcare solutions, especially for the elderly, is effectively promoted.

2. **Diverse Applications with Performance and Battery Life Optimization:** We strive to include a diverse set of applications in our devices. By enhancing performance and optimizing battery life, we ensure that our devices cater to a broad range of user needs. This approach allows us to address various health concerns and scenarios, making our product more versatile in the market.

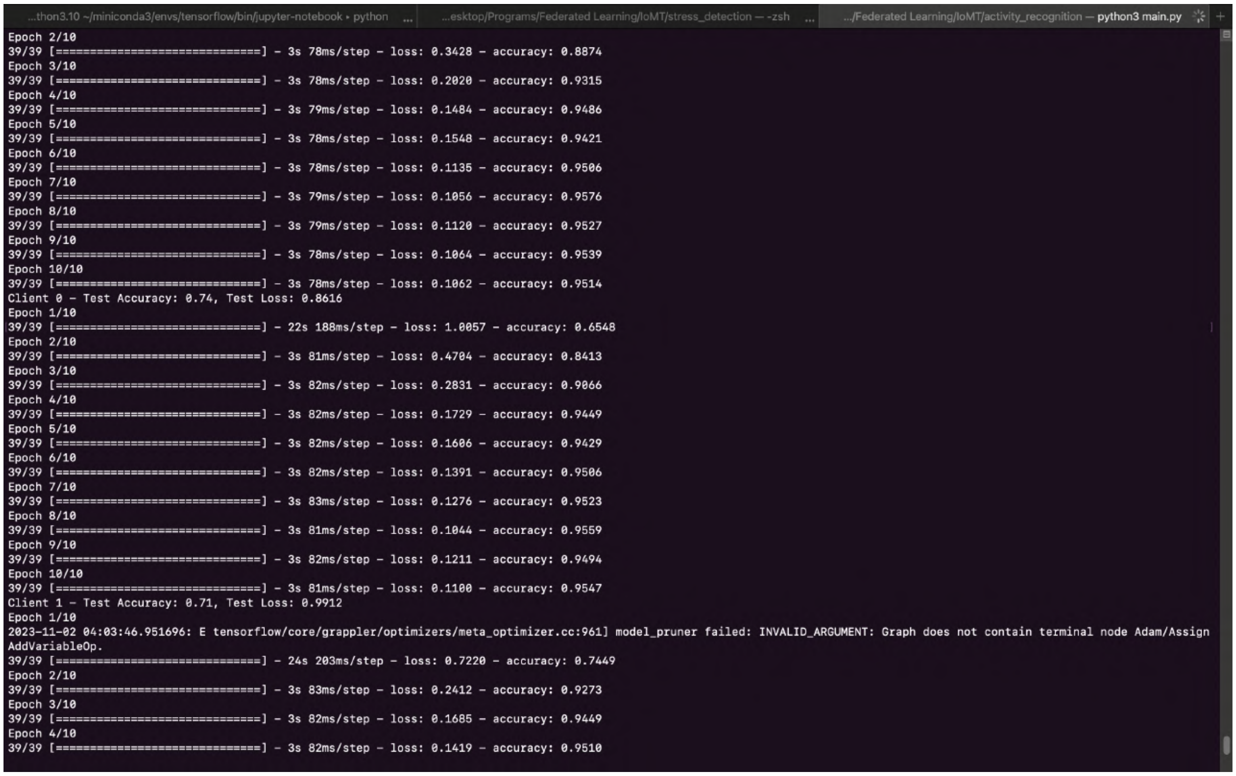
4.5 Arcitecture of Our Model

Although we have selected the minimal applications to show that when connected these devices to the internet or through a phone to decrease the vulnerability we use Federated Learning to improve the Personalization, Recommendations and User Experience.



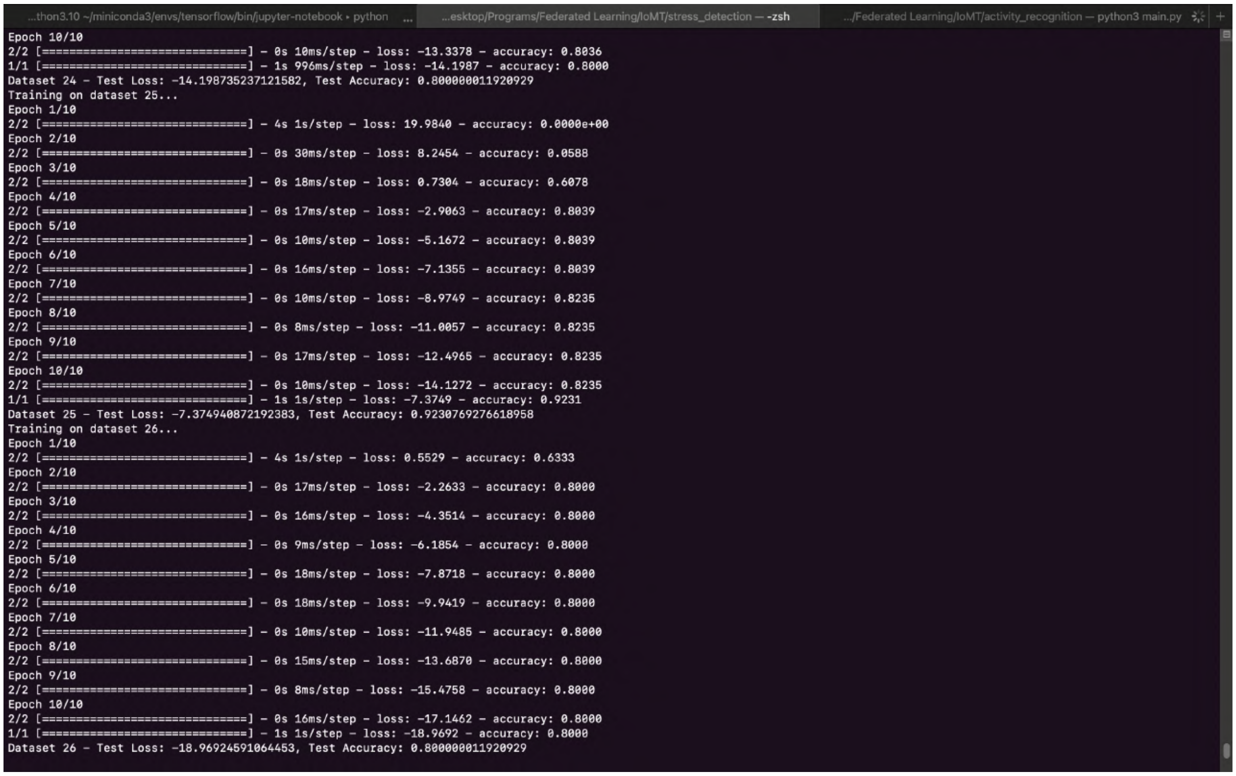
4.6 Execution Plan (Case Study : HAR) Output

We implemented Human Activity Recognition (HAR) using the LSTM-FCN model, combining Long Short-Term Memory (LSTM) and Fully Convolutional Networks (FCN). It classifies activities like walking, sitting, etc., based on inertial sensor data. The model is trained using federated learning across multiple clients and achieves a commendable 80% accuracy on the test data, showcasing its effectiveness in privacy-preserving machine learning for activity recognition.



4.7 Execution Plan (Case Study : HR Stress) Output

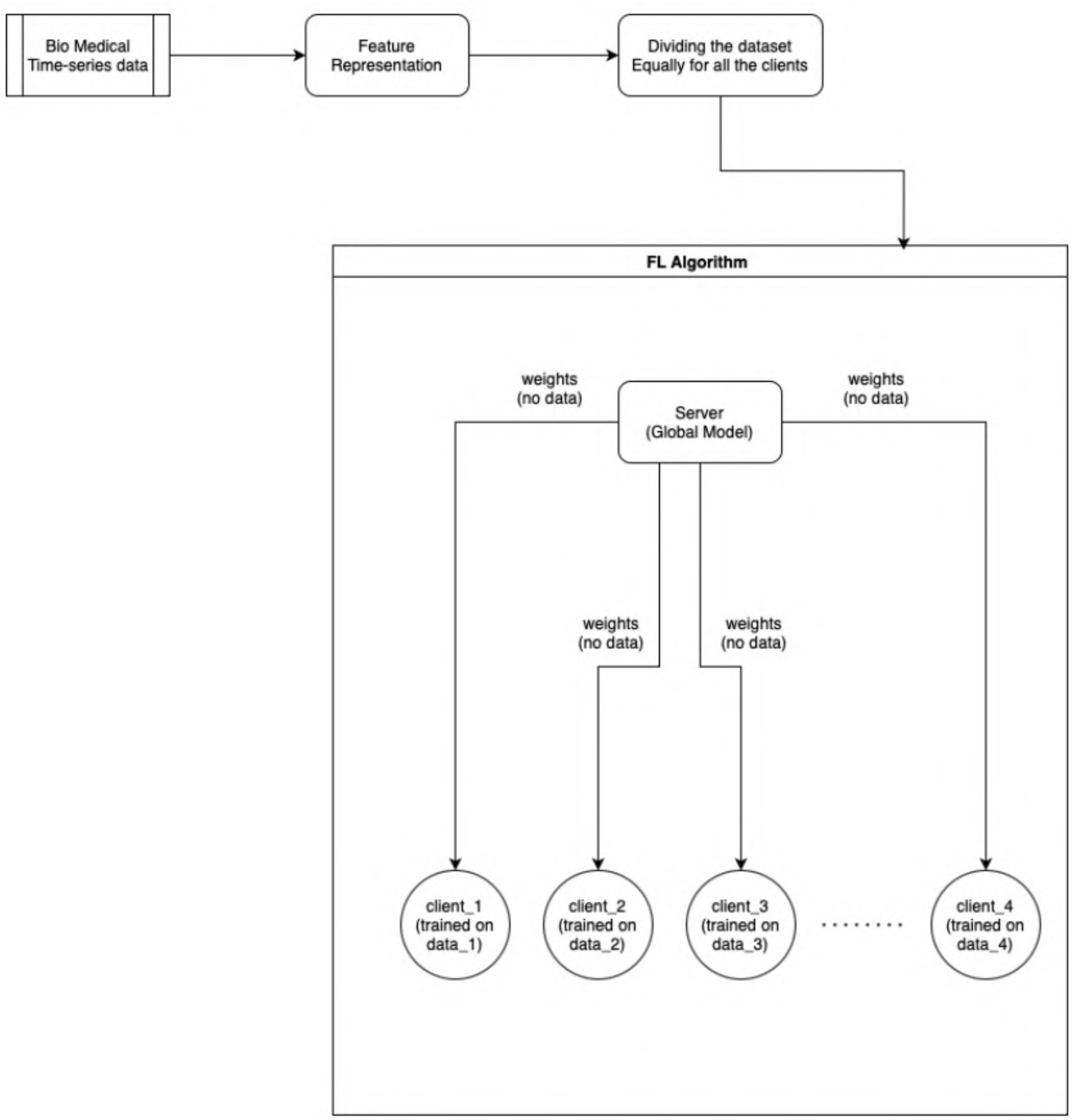
We implemented personal heart rate datasets for federated learning, employing a global BiLSTM (Bidirectional Long Short-Term Memory) and LSTM- FCN neural network model. The global model is trained on 26 individual client datasets, demonstrating robustness and privacy preservation. After training, the model achieves an impressive 94% accuracy on the test data, showcasing the efficacy of the BiLSTM architecture in capturing temporal dependencies for heart rate classification.



...thon3.10 ~/miniconda3/envs/tensorflow/bin/jupyter-notebook ▸ python
...esktop/Programs/Federated Learning/IoMT/stress_detection — -zsh
.../Federated Learning/IoMT/activity_recognition — python3 main.py
Epoch 10/10
2/2 [==============================] - 0s 10ms/step - loss: -13.3378 - accuracy: 0.8036
1/1 [==============================] - 1s 996ms/step - loss: -14.1987 - accuracy: 0.8000
Dataset 24 - Test Loss: -14.198735237121582, Test Accuracy: 0.800000011920929
Training on dataset 25...
Epoch 1/10
2/2 [==============================] - 4s 1s/step - loss: 19.9840 - accuracy: 0.0000e+00
Epoch 2/10
2/2 [==============================] - 0s 30ms/step - loss: 8.2454 - accuracy: 0.0588
Epoch 3/10
2/2 [==============================] - 0s 18ms/step - loss: 0.7304 - accuracy: 0.6078
Epoch 4/10
2/2 [==============================] - 0s 17ms/step - loss: -2.9063 - accuracy: 0.8039
Epoch 5/10
2/2 [==============================] - 0s 10ms/step - loss: -5.1672 - accuracy: 0.8039
Epoch 6/10
2/2 [==============================] - 0s 16ms/step - loss: -7.1355 - accuracy: 0.8039
Epoch 7/10
2/2 [==============================] - 0s 10ms/step - loss: -8.9749 - accuracy: 0.8235
Epoch 8/10
2/2 [==============================] - 0s 8ms/step - loss: -11.0057 - accuracy: 0.8235
Epoch 9/10
2/2 [==============================] - 0s 17ms/step - loss: -12.4965 - accuracy: 0.8235
Epoch 10/10
2/2 [==============================] - 0s 10ms/step - loss: -14.1272 - accuracy: 0.8235
1/1 [==============================] - 1s 1s/step - loss: -7.3749 - accuracy: 0.9231
Dataset 25 - Test Loss: -7.374940872192383, Test Accuracy: 0.9230769276618958
Training on dataset 26...
Epoch 1/10
2/2 [==============================] - 4s 1s/step - loss: 0.5529 - accuracy: 0.6333
Epoch 2/10
2/2 [==============================] - 0s 17ms/step - loss: -2.2633 - accuracy: 0.8000
Epoch 3/10
2/2 [==============================] - 0s 16ms/step - loss: -4.3514 - accuracy: 0.8000
Epoch 4/10
2/2 [==============================] - 0s 9ms/step - loss: -6.1854 - accuracy: 0.8000
Epoch 5/10
2/2 [==============================] - 0s 18ms/step - loss: -7.8718 - accuracy: 0.8000
Epoch 6/10
2/2 [==============================] - 0s 18ms/step - loss: -9.9419 - accuracy: 0.8000
Epoch 7/10
2/2 [==============================] - 0s 10ms/step - loss: -11.9485 - accuracy: 0.8000
Epoch 8/10
2/2 [==============================] - 0s 15ms/step - loss: -13.6870 - accuracy: 0.8000
Epoch 9/10
2/2 [==============================] - 0s 8ms/step - loss: -15.4758 - accuracy: 0.8000
Epoch 10/10
2/2 [==============================] - 0s 16ms/step - loss: -17.1462 - accuracy: 0.8000
1/1 [==============================] - 1s 1s/step - loss: -18.9692 - accuracy: 0.8000
Dataset 26 - Test Loss: -18.96924591064453, Test Accuracy: 0.800000011920929

Chapter 5

Proof of Concept



Chapter 6

Python Implementation

6.1 Libraries

```
import matplotlib.pyplot as plt
import numpy as np
import os
from patchify import patchify
import cv2
from keras.models import load_model
import random
from PIL import Image



#Imports
import os
from google.colab import drive
from PIL import Image
import os
from pathlib import Path
from google.auth.transport.requests import AuthorizedSession
from google.oauth2 import service_account
from pprint import pprint
import json
import ee
from IPython.display import Image


import matplotlib.pyplot as plt
import numpy as np
import os
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
import cv2
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.callbacks import ModelCheckpoint, LearningRateScheduler,
    EarlyStopping
```

```python
import random



import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
from PIL import Image
from patchify import patchify
import splitfolders
import random
from keras.utils import to_categorical
```

6.2 Code

6.2.1 Human Stress Recognition

```python

# Load and preprocess your datasets
datasets = [
    pd.read_csv('./personal_HR/1.csv'),
    pd.read_csv('./personal_HR/2.csv'),
    pd.read_csv('./personal_HR/3.csv'),
    pd.read_csv('./personal_HR/4.csv'),
    pd.read_csv('./personal_HR/5.csv'),
    pd.read_csv('./personal_HR/6.csv'),
    pd.read_csv('./personal_HR/7.csv'),
    pd.read_csv('./personal_HR/8.csv'),
    pd.read_csv('./personal_HR/9.csv'),
    pd.read_csv('./personal_HR/10.csv'),
    pd.read_csv('./personal_HR/11.csv'),
    pd.read_csv('./personal_HR/12.csv'),
    pd.read_csv('./personal_HR/13.csv'),
    pd.read_csv('./personal_HR/14.csv'),
    pd.read_csv('./personal_HR/15.csv'),
    pd.read_csv('./personal_HR/16.csv'),
    pd.read_csv('./personal_HR/17.csv'),
    pd.read_csv('./personal_HR/18.csv'),
    pd.read_csv('./personal_HR/19.csv'),
    pd.read_csv('./personal_HR/20.csv'),
    pd.read_csv('./personal_HR/21.csv'),
    pd.read_csv('./personal_HR/22.csv'),
    pd.read_csv('./personal_HR/23.csv'),
    pd.read_csv('./personal_HR/24.csv'),
    pd.read_csv('./personal_HR/25.csv'),
    pd.read_csv('./personal_HR/26.csv')
]

# Create and compile your Keras model
global_model = Sequential()
global_model.add(Dense(64, input_dim=16, activation='relu'))
```

```
global_model.add(Dense(32, activation='relu'))
global_model.add(Dense(1, activation='sigmoid'))






num_clients = 26


test_loss_clients = []
test_accuracy_clients = []
clients_weights = []
client_models = []

# Training and evaluation loop
for i, dataset in enumerate(datasets):
    print(f"Training on dataset {i + 1}...")

    # Preprocess the dataset
    X = dataset.drop(columns=["class"]).values
    y = dataset["class"].values

    model = Sequential()
    model.add(Dense(64, input_dim=16, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
    accuracy'])


    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)

    # Train the model on the dataset
    model.fit(X_train, y_train, epochs=10, batch_size=32)

    # Evaluate the model on the test data
    loss, accuracy = model.evaluate(X_test, y_test)
    test_loss_clients.append(loss)
    test_accuracy_clients.append(loss)
    client_models.append(model)
    print(f"Dataset {i + 1} - Test Loss: {loss}, Test Accuracy: {accuracy}
    ")




global_weights = global_model.get_weights()
for i in range(len(clients_weights)):
    for j in range(num_clients):
        client_weights = client_models[j].get_weights()
```

```
        global_weights[i] += client_weights[i]




global_model.set_weights(global_weights)


global_model.compile(loss='binary_crossentropy', optimizer='adam', metrics
    =['accuracy'])



for i, dataset in enumerate(datasets):
    print(f"Global Model testing on dataset {i + 1}...")

    # Preprocess the dataset
    X = dataset.drop(columns=["class"]).values
    y = dataset["class"].values
    # Evaluate the global model
    test_loss, test_accuracy = global_model.evaluate(X, y, verbose=0)
    print(f'Global Test Accuracy: {test_accuracy:.2f}, Global Test Loss: {
    test_loss:.4f}')
```

6.2.2 Human Activity Recognition

```

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import tensorflow as tf  # Version 1.0.0 (some previous versions are used
    in past commits)
from sklearn import metrics
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.layers import LSTM, Dense, BatchNormalization
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.layers import LSTM, Dense, BatchNormalization

# Useful Constants
DATA_PATH = "data/"
DATASET_PATH = DATA_PATH + "UCI HAR Dataset/"
print("\n" + "Dataset is now located at: " + DATASET_PATH)

# Those are separate normalised input features for the neural network
INPUT_SIGNAL_TYPES = [
    "body_acc_x_",
    "body_acc_y_",
    "body_acc_z_",
```

```
    "body_gyro_x_",
    "body_gyro_y_",
    "body_gyro_z_",
    "total_acc_x_",
    "total_acc_y_",
    "total_acc_z_"
]

# Output classes to learn how to classify
LABELS = [
    "WALKING",
    "WALKING_UPSTAIRS",
    "WALKING_DOWNSTAIRS",
    "SITTING",
    "STANDING",
    "LAYING"
]



TRAIN = "train/"
TEST = "test/"


# Load "X" (the neural network's training and testing inputs)

def load_X(X_signals_paths):
    X_signals = []

    for signal_type_path in X_signals_paths:
        file = open(signal_type_path, 'r')
        # Read dataset from disk, dealing with text files' syntax
        X_signals.append(
            [np.array(serie, dtype=np.float32) for serie in [
                row.replace('  ', ' ').strip().split(' ') for row in file
            ]]
        )
        file.close()

    return np.transpose(np.array(X_signals), (1, 2, 0))





X_train_signals_paths = [
    DATASET_PATH + TRAIN + "Inertial Signals/" + signal + "train.txt" for signal in INPUT_SIGNAL_TYPES
]
X_test_signals_paths = [
    DATASET_PATH + TEST + "Inertial Signals/" + signal + "test.txt" for signal in INPUT_SIGNAL_TYPES
]

```

```python
X_train = load_X(X_train_signals_paths)
X_test = load_X(X_test_signals_paths)




# Load "y" (the neural network's training and testing outputs)

def load_y(y_path):
    file = open(y_path, 'r')
    # Read dataset from disk, dealing with text file's syntax
    y_ = np.array(
        [elem for elem in [
            row.replace('  ', ' ').strip().split(' ') for row in file
        ]],
        dtype=np.int32
    )
    file.close()

    # Substract 1 to each output class for friendly 0-based indexing
    return y_ - 1

y_train_path = DATASET_PATH + TRAIN + "y_train.txt"
y_test_path = DATASET_PATH + TEST + "y_test.txt"

y_train = load_y(y_train_path)
y_test = load_y(y_test_path)



# Input Data

training_data_count = len(X_train)  # 7352 training series (with 50% overlap between each serie)
test_data_count = len(X_test)  # 2947 testing series
n_steps = len(X_train[0])  # 128 timesteps per series
n_input = len(X_train[0][0])  # 9 input parameters per timestep


# LSTM Neural Network's internal structure
n_hidden = 32 # Hidden layer num of features
n_classes = 6 # Total classes (should go up, or should go down)


# Training
learning_rate = 0.01
lambda_loss_amount = 0.0015
training_iters = training_data_count * 300  # Loop 300 times on the dataset
batch_size = 1500
display_iter = 30000  # To show test set accuracy during training


# Some debugging info
```

```python
print("Some useful info to get an insight on dataset's shape and normalisation:")
print("(X shape, y shape, every X's mean, every X's standard deviation)")
print(X_test.shape, y_test.shape, np.mean(X_test), np.std(X_test))
print("The dataset is therefore properly normalised, as expected, but not yet one-hot encoded.")

def LSTM_RNN(input_shape, n_hidden, n_classes):
    model = keras.Sequential()
    model.add(LSTM(n_hidden, return_sequences=True, input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(LSTM(n_hidden, return_sequences=True))
    model.add(BatchNormalization())
    model.add(LSTM(n_hidden, return_sequences=True))
    model.add(BatchNormalization())
    model.add(LSTM(n_hidden))
    model.add(BatchNormalization())
    model.add(Dense(n_classes, activation='softmax'))
    return model

# Training parameters
batch_size = 64
num_epochs = 10
num_clients = 3

# Define the model and placeholders
n_input = 9
n_steps = 128  # The shape of your input data

n_hidden = 32
n_classes = 6
input_shape = (n_steps, n_input)  # Define the global input shape

X_train_clients = np.array_split(X_train, num_clients)
y_train_clients = np.array_split(y_train, num_clients)
X_test_clients = np.array_split(X_test, num_clients)
y_test_clients = np.array_split(y_test, num_clients)

test_loss_clients = []
test_accuracy_clients = []
clients_weights = []
client_models = []
for i in range(num_clients):
```

```
    model = LSTM_RNN(input_shape, n_hidden, n_classes)
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    y_train_one_hot = keras.utils.to_categorical(y_train_clients[i], n_classes)
    model.fit(X_train_clients[i], y_train_one_hot, batch_size=batch_size, epochs=num_epochs, verbose=1)
    test_loss, test_accuracy = model.evaluate(X_test_clients[i], keras.utils.to_categorical(y_test_clients[i], n_classes), verbose=0)
    test_loss_clients.append(test_loss)
    test_accuracy_clients.append(test_accuracy)
    clients_weights.append(model.get_weights())
    client_models.append(model)
    print(f'Client {i} - Test Accuracy: {test_accuracy:.2f}, Test Loss: {test_loss:.4f}')

global_model = LSTM_RNN(input_shape, n_hidden, n_classes)






global_weights = global_model.get_weights()
for i in range(len(clients_weights)):
    for j in range(num_clients):
        client_weights = client_models[j].get_weights()
        global_weights[i] += client_weights[i]

global_model.set_weights(global_weights)
y_train_one_hot = keras.utils.to_categorical(y_train, n_classes)
# Compile the global model
global_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
global_model.fit(X_train, y_train_one_hot, batch_size=batch_size, epochs=num_epochs, verbose=1)

# Evaluate the global model
test_loss, test_accuracy = global_model.evaluate(X_test_clients[0], keras.utils.to_categorical(y_test_clients[0], n_classes), verbose=0)
print(f'Global Test Accuracy: {test_accuracy:.2f}, Global Test Loss: {test_loss:.4f}')
```