IMPLEMENTATION OF SPLIT FEDERATED LEARNING ON RESOURCE CONSTRAINED DEVICES

by

Barış Giray Akman	**150121822**
Yasin Küçük	**150120062**
Ömer Deligöz	**150120035**

CSE4197 Engineering Project 1

Project Specification Document

Supervised by:

Assoc. Prof. Ömer Korçak

Marmara University, Faculty of Engineering

Computer Engineering Department

15.11.2024

1 PROBLEM STATEMENT

Traditional centralized machine learning approaches require data to be transferred to a central server, which might cause privacy concerns, high communication costs, and latency issues. As a solution, Federated Learning (FL) approach has emerged, enabling devices to collaboratively learn a shared model without sharing the training data. Most existing FL studies are designed with the assumption of relatively powerful devices and are not suitable for resource constrained devices. In this project, we focus on implementing FL on resource constrained devices, such as Raspberry Pi, Arduino, and smartphones. To reduce the computational load on client devices, we will implement a combination of split learning and federated learning approaches.

2 PROBLEM DESCRIPTION AND MOTIVATION

Current machine learning models are increasingly employed to predict outcomes based on specified parameters; however, they often lack considerations for privacy and security policies. Federated Learning addresses this gap by enabling the model to function without sharing datasets with external servers. Instead, only model weights and gradients are transmitted to the server, significantly reducing the risk of privacy leakage.

However, in many applications where data is collected/generated by resource constrained devices such as IoT devices, training a deep learning model in the client-side may lead to prolonged training times and potential overloading. To address this, we will apply split learning by training part of the model on the client device and the remainder on a more powerful server. Following this split learning phase, model aggregation will occur on a federated learning server, and this approach is known as Split Federated Learning (SFL). Our objective is to implement and test SFL on resource-constrained devices. Prior to training, we will select the appropriate devices, models, and determine the optimal number of layers to train on the client side.

We will establish an experimental environment consisting of several resource-constrained devices and a server, implementing the SFL model training and communication protocols across these devices. The setup will allow us to test training time and model accuracy in

various scenarios, evaluating the feasibility of our approach by comparing its performance with centralized learning and traditional FL methods deployed on high-performance devices.

3 MAIN GOAL AND OBJECTIVES

The main goal of this project is to develop and evaluate a Split Federated Learning (SFL) framework that enables resource-constrained devices such as Raspberry Pi to perform secure, privacy-preserving machine learning that optimizes model performance while addressing limitations in computational power, memory, and energy.

3.1 Objective 1: Split Learning Architecture Design

Develop and implement a split learning model architecture suited for low-power devices, dividing the model into client-side and server-side components to reduce computational load on the device.

3.2 Objective 2: Data Preparation and Distribution

Prepare and pre-process data to be distributed across multiple devices, ensuring each device holds a partitioned dataset suitable for localized training.

3.3 Objective 3: Privacy-Preserving Communication Protocol

Design and implement a communication protocol for parameter exchange that preserves data privacy by only sharing model updates (not raw data) between devices and the server.

3.4 Objective 4: Model Performance Evaluation

Test and evaluate the model's performance on metrics such as accuracy, training time, resource utilization (CPU, memory, and energy consumption), and communication overhead on the clients.

4 RELATED WORK

4.1 Communication-Efficient Learning of Deep Networks from Decentralized Data

McMahan et al. [1] introduces Federated Learning (FL) and proposes the FedAvg algorithm to address the challenges of training models across decentralized data sources while maintaining privacy. FedAvg enhances communication efficiency by allowing client devices to perform multiple local updates before sharing their models with the central server, reducing communication frequency and scaling well in resource-constrained environments. This work also emphasizes privacy-preserving learning, as data never leaves the client device, which sparked further research into mechanisms like differential privacy and secure aggregation. McMahan et al.'s focus on communication efficiency and local computation is foundational for Split-Federated Learning, where similar challenges exist. Their methods of minimizing communication and balancing local and global updates have significantly influenced approaches to distributed learning, providing valuable insights for optimizing the trade-offs between model accuracy, resource efficiency, and privacy in Split-Federated Learning.

4.2 SplitFed: When Federated Learning Meets Split Learning

Thapa et al. [4] proposed a solution which encompasses both Split Learning and Federated Learning. In Federated Learning, the entire model is executed at the clients and these models are running in parallel manner. However, in Split Learning, only the specific fragment of the whole model is executed at each client device.

Figure 1: Split-Federated learning (SFL) system model [4]

In Split-Federated Learning, layers are allocated for each device and weights are updated in parallel. **Figure 1** shows how Split-Federated Learning divides model training across clients, a Fed Server, and a Main Server. Each client holds initial model layers, processes local data, and sends intermediate "smashed data" to the Main Server, which holds the final layers and completes the forward and backward propagation. The Main Server then sends gradients back to clients to update their local model portions. Periodically, the Fed Server aggregates these updates into a global model, shared with all clients to keep them synchronized. This setup optimizes resource use and maintains privacy by distributing model training across devices and servers. Consequently, in the paper, in terms of resource consumption, Split Federated learning performs better than Federated Learning.

4.3 MergeSFL: Split Federated Learning with Feature Merging and Batch Size Regulation

Algorithms like Federated Learning and Split Learning address key security concerns required for training algorithms across distributed environments. However, when client devices contain heterogeneous datasets, the accuracy of these algorithms often suffers. To address this issue, the MergeSFL paper introduces a novel approach. In MergeSFL, the model is divided into three components: the top model, the bottom model, and a parameter server (PS). The features collected from participating devices are merged in a

mixed feature sequence, and batch sizes are dynamically adjusted based on the processing capacities of each worker. In Figure 2 below, differences of MergeSFL and typical SFL were illustrated visually.

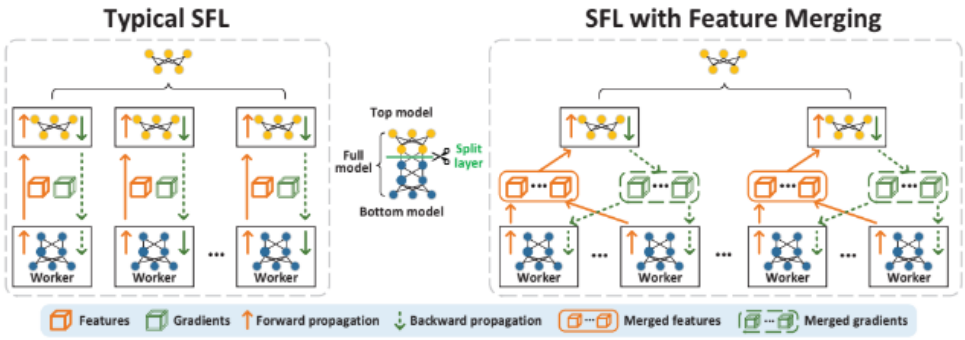

Figure 2: Illustration of typical SFL (left) and SFL with feature merging (right). [2]

In comparison to MergeSFL and SplitFed, which focus on architecture and model partitioning, our approach may involve reworking or modifying the foundational strategies of these algorithms, along with optimizing parameters. We will also examine if these algorithms can be effectively performed on low-performance devices. This could lead to more fine-grained improvements in efficiency, scalability, and adaptability, especially in resource-constrained environments.

5 SCOPE

This project aims to implement Split Federated Learning (SFL) and evaluate its feasibility on resource-constrained devices, such as Raspberry Pi units. By employing SFL, the project enables these devices to participate in FL without transmitting raw data, thus ensuring data privacy and reducing communication overhead. Rather than creating new algorithms, this project emphasizes the application of existing deep learning models—such as those for text or tabular datasets (e.g., RNNs for text classification or feedforward neural networks for tabular regression)—to assess their performance in constrained environments.

The implementation divides the model training process, allowing client devices to handle part of the computation locally, while the server processes the remaining model

components. This split learning approach decreases the computational load on each device while preserving on-device data privacy, as only model parameters or partial updates, rather than raw data, are transmitted to the server.

Key aspects of the project scope include:

- **Target Devices:** Experiments will use Raspberry Pi devices, chosen for their popularity, affordability, and representational constraints. Although this implementation focuses on Raspberry Pi devices, findings are expected to generalize to other resource-limited platforms.

- **Model Training:** Deep learning models suited for text and tabular data will be tested, selected to fit within the computational capabilities of the devices and relevant datasets.

- **Data Privacy:** Data privacy is a fundamental project goal, with SFL ensuring that raw data remains on each device, reinforcing privacy by transmitting only model updates.

- **Performance Evaluation:** The project will assess SFL performance on metrics such as model accuracy, training time, communication overhead, and resource utilization (CPU, memory, and energy consumption) on the Raspberry Pi devices. These will be compared with traditional federated learning and centralized training baselines.

Out of scope:

- **Security and Encryption:** The project assumes a trusted communication environment where data privacy is preserved through local data processing without additional encryption. No additional security protocols (such as encryption) are applied, as all communication is assumed to be secure within the experimental setup.

Constraints:

- **Hardware Limitation**: The project will use only CPU-based processing on the Raspberry Pi devices, without exploring GPU or hardware accelerations to maintain simplicity and ensure the feasibility of implementation on low-cost devices.
- **Small-Scale Deployment**: This project is limited to a small number of Raspberry Pi devices (up to 5) and does not address scalability challenges associated with deploying SFL across larger, more complex network topologies.
- **Limited Model Complexity**: Due to the constraints of Raspberry Pi devices, model complexity is limited, focusing on lightweight models that can operate within the hardware's limited memory and processing power.

This scope highlights the practical application of split federated learning in real-world constrained environments, aiming to validate the approach's feasibility and effectiveness on devices with limited resources.

6 METHODOLOGY AND TECHNICAL APPROACH

6.1 Workflow

The SFL process follows these steps:

6.1.1 Data preparation

The dataset (e.g., IMDb Movie Reviews [7] for text classification, Sentiment140 [8] for sentiment analysis, or UCI Diabetes [9] for predictive analysis) is pre-processed. This may involve cleaning, normalizing, and splitting the data into smaller portions. Once prepared, the data is distributed to the Raspberry Pi clients, with each client holding a unique subset of the data. This allows each device to independently train its local portion of the model on specific samples, enabling localized learning and maintaining data privacy across clients.

6.1.2 Model Partitioning

The deep learning model is divided into two parts—client-side and server-side—to create a split learning environment. This segmentation allows the client devices (e.g., Raspberry Pi) to handle the initial layers of the model locally, while the server processes the deeper layers. The **client-side segment** consists of the initial layers responsible for extracting basic features from the data, while the **server-side segment** holds the deeper layers that aggregate higher-level representations and complete the model's training. By limiting the depth of model layers processed on each Raspberry Pi device, this approach significantly reduces the computational load on these resource-constrained clients.

To optimize this process, we will experiment with where the model should be split. **The number of layers assigned to the client may vary depending on the device's capability**, such as memory and processing power. Devices with higher capacities may handle more layers, whereas those with stricter limitations may only process a few initial layers. Choosing the appropriate split point based on the device's capabilities is a key factor in this approach, allowing us to adapt the model structure to suit various resource constraints effectively.

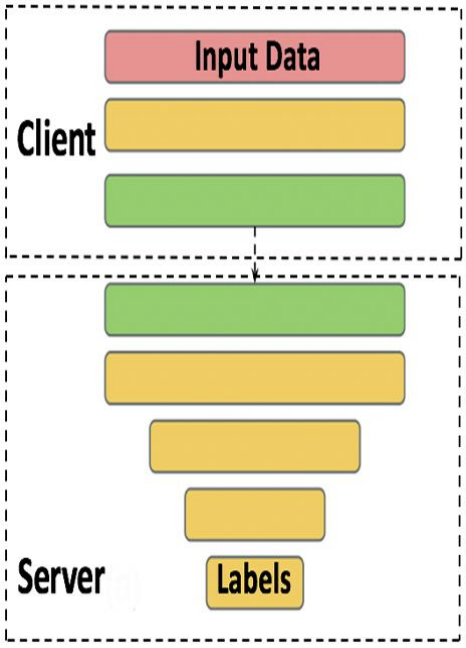

Figure 3: Split learning architecture [5]

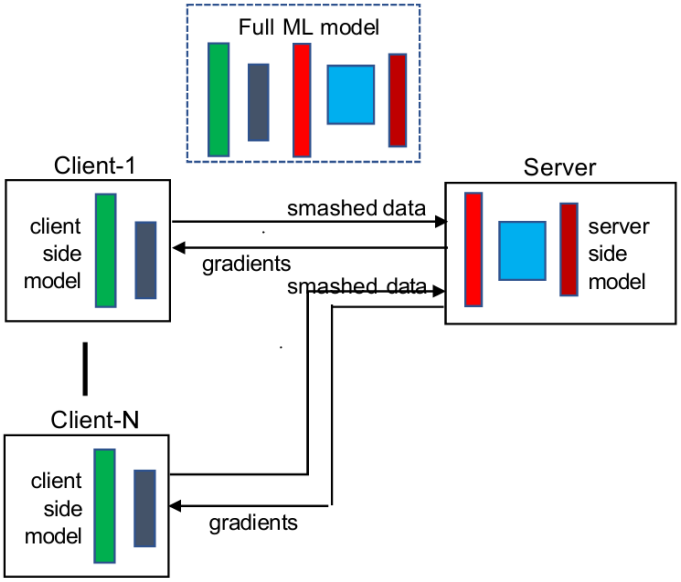

Figure 4: Data flow between client-server [6]

- **Figure 3** illustrates the split learning architecture, showing the model divided into client and server segments. The client side handles basic feature extraction, while the server side completes the higher-level processing.
- **Figure 4** demonstrates the data flow between the client and server. Only intermediate representations (not raw data) are transmitted to the server, ensuring privacy while reducing the data transfer load.

These figures provide a visual representation of the model segmentation and data flow, emphasizing the flexibility of split learning in adjusting the model split point to suit each device's capacity. This approach ensures that each device operates efficiently, balancing computational demands with resource availability.

6.1.3 Client-Side Training

Each client trains its model with its local data. This training focuses on feature extraction in the early model layers, transforming the raw data into intermediate representations. The training is limited to the client's portion of the model, allowing the device to process the data within its hardware constraints. During this step, the client adjusts its model parameters based on the local data, optimizing for accuracy on its subset of data.

6.1.4 Parameter Exchange

After local training, each client device sends the updated parameters of its model segment (i.e., the intermediate layer outputs or partial gradients) to the central server. This exchange transmits only the model updates (not the raw data) ensuring that data privacy is preserved. The transferred parameters represent the learning each client has derived from its local data and are essential for server-side aggregation.

6.1.5 Server-Side Aggregation & Training

The central server collects model updates (intermediate representations or gradients) sent by each client after local training on the initial layers of the model. These updates are then aggregated to update the global model, improving its ability to generalize across

all client data. A common aggregation method used for this purpose is **Federated Averaging (FedAvg)**, which is particularly effective in federated learning environments.

How FedAvg Works

- **Local Training**: Each client trains its model locally on its data, generating weight or gradient updates.
- **Sending Updates**: Clients send these updates to the server, ensuring data privacy by not sharing raw data.
- **Weighted Averaging**: The server applies FedAvg by averaging updates from all clients, with weights based on each client's data size:

$$Global\ Update\ = \frac{1}{N}\sum_{i=0}^{N} \omega_i\ *\ Client\ Update_i$$

  Here, ω_i is based on each client's data size, ensuring proportional influence on the global model.

- **Global Model Update**: The server updates the global model and sends it back to the clients for the next round.

As an example, if three clients contribute updates, with data sizes of 1000, 2000, and 500 samples, the server applies weighted averaging (e.g., weights of 0.285, 0.571, and 0.143) to generate an updated global model that reflects data diversity and improves generalization.

This FedAvg approach optimizes split federated learning by reducing communication, ensuring scalability, and enhancing the model's ability to generalize across diverse, privacy-preserving client data.

6.1.6 Model Update Distribution

The server sends the updated global model parameters back to each client device. These updates may involve adjustments to the parameters of the client-side model segment,

which are sent as incremental updates. By receiving these refined parameters, each client benefits from the global learning, enhancing local model accuracy without direct access to data from other clients.

6.1.7 Iteration

Steps 3 through 6 are repeated in multiple rounds, allowing the model to improve over time. In each round, client devices learn from their local data and benefit from the server's aggregated learning. This continues until the model reaches a satisfactory accuracy or a set training limit is reached.

7 PROFESSIONAL CONSIDERATIONS

7.1 Methodological considerations/engineering standards:

Table 1 below shows key tool categories for the project: Project Management for code control and document sharing, Literature Survey for research resources, Federated Learning Libraries for decentralized model training, Algorithm Implementation for model development, Data Processing and Visualization for data handling and performance insights, and Deployment and Testing on edge devices to evaluate performance in resource-limited environments.

Table 1: Project tools and software.

Category	**Tools and Software**
Project Management	• **Git & GitHub:** Source code control • **Google Drive**: Document and resource storage
Literature Survey	• **Google Scholar:** Research articles and academic papers • **IEEE Xplore:** Access to publications on federated learning and machine learning • **Elsevier - ScienceDirect:** Research resources for privacy-preserving machine learning

Table 1: continued

Federated Learning Libraries	• **TensorFlow Federated (TFF):** Framework for federated learning model training • **PySyft:** Library for privacy-preserving machine learning implementations • **Flower (FLwr):** Flexible framework for federated learning across different ML libraries • **OpenFL**: Federated learning framework focused on interoperability and privacy
Algorithm Implementation	• **Python:** Programming language for model development and implementation
Data Processing and Visualization	• **Pandas:** Data manipulation and pre-processing • **NumPy:** Numerical computation for data processing • **Matplotlib:** Plotting and visualizing data trends and model performance
Deployment and Testing on Edge Devices	• **Raspberry Pi**: Hardware platform for testing and performance evaluation

7.2 Realistic Constraints

7.2.1 Economical

The project is designed to be cost-effective by utilizing readily available and affordable hardware (Raspberry Pi devices). The use of open-source software further minimizes costs.

7.2.2 Environmental

The project's focus on resource efficiency contributes to environmental sustainability by reducing energy consumption during the training process. The use of low-power devices like the Raspberry Pi further minimizes the environmental impact.

7.2.3 Ethical

Data privacy is a central ethical consideration. The SFL approach, by design, keeps sensitive data localized on individual devices, mitigating privacy risks. The project adheres to ethical data handling practices and ensures that no data is used without proper consent

7.2.4 Health and Safety

There are no significant health and safety concerns associated with this project as it primarily involves software development and experimentation with low-power devices. Standard laboratory safety practices will be followed.

7.2.5 Sustainability

The focus on resource efficiency promotes computational sustainability by minimizing energy consumption. The use of open-source software promotes long-term sustainability by ensuring accessibility and maintainability of the project.

7.2.6 Social

The project contributes to the broader social good by advancing research in privacy-preserving machine learning. This technology has the potential to empower individuals and communities by enabling data analysis without compromising privacy.

7.3 Legal considerations:

This project is focused on research and proof-of-concept, so no immediate legal concerns about market entry are expected. However, if commercialized, proper licenses for the software and datasets would be needed. If sensitive data (like medical or financial data)

is involved, ethical approvals and data regulations would apply. Currently, the project uses publicly available or synthetic data, reducing legal issues.

8 MANAGEMENT PLAN

8.1 Project Structure and Roles

Team Members and Core Responsibilities

- **Barış Giray Akman** (Project Manager & Lead Developer): Oversees project coordination, timeline, architecture, implementation, code quality, and supervisor communication.
- **Yasin Küçük** (System Administrator & Infrastructure Specialist): Manages hardware setup, network security, performance optimization, and resource monitoring.
- **Ömer Deligöz** (Data Specialist & ML Engineer): Handles data preprocessing, model evaluation, performance analysis, and documentation.

8.2 Milestones and Timeline

The project is divided into phases, each with specific deliverables:

- **Phase 1: Initial Research and Planning**
    - Research on SFL, federated learning frameworks, and related privacy-preserving techniques.
    - Choose machine learning libraries (TensorFlow, PyTorch) and finalize the federated learning frameworks (e.g., TensorFlow Federated, Flower).
    - Setup GitHub for source control and collaboration.
    - **Deliverable**: Project plan, literature review, and finalized technology stack.
- **Phase 2: Design and Prototype Development**
    - Design SFL architecture, including model partitioning between client (Raspberry Pi) and server.
    - Build and test a small-scale prototype on Google Colab to benchmark model performance.

- **Deliverable**: Prototype model architecture, initial performance results on Google Colab.

- **Phase 3: Implementation on Raspberry Pi**
  - Deploy the prototype on Raspberry Pi devices, optimize the model for constrained resources.
  - Implement data privacy features to ensure only model updates (not raw data) are transmitted to the server.
  - **Deliverable**: Fully functional SFL setup on Raspberry Pi, with privacy-preserving protocols in place.
- **Phase 4: Testing and Performance Evaluation**
  - Run model training on both Google Colab and Raspberry Pi to compare performance metrics, including accuracy, runtime, resource utilization, and privacy compliance.
  - Address any performance issues and optimize for Raspberry Pi.
  - **Deliverable:** Performance evaluation report comparing SFL and federated learning, and improvements made for constrained devices.
- **Phase 5: Documentation and Final Report**
  - Prepare project documentation, including system architecture, code comments, user guide, and detailed analysis of results.
  - Complete final project report with insights into the feasibility and efficiency of SFL on resource-limited devices.
  - **Deliverable:** Project documentation and final report.

	October	November	December	January	February	March	April	May
Phase 1								
Phase 2								
Phase 3								
Phase 4								
Phase 5								

Figure 5: GANTT Chart for the Project TimeLine

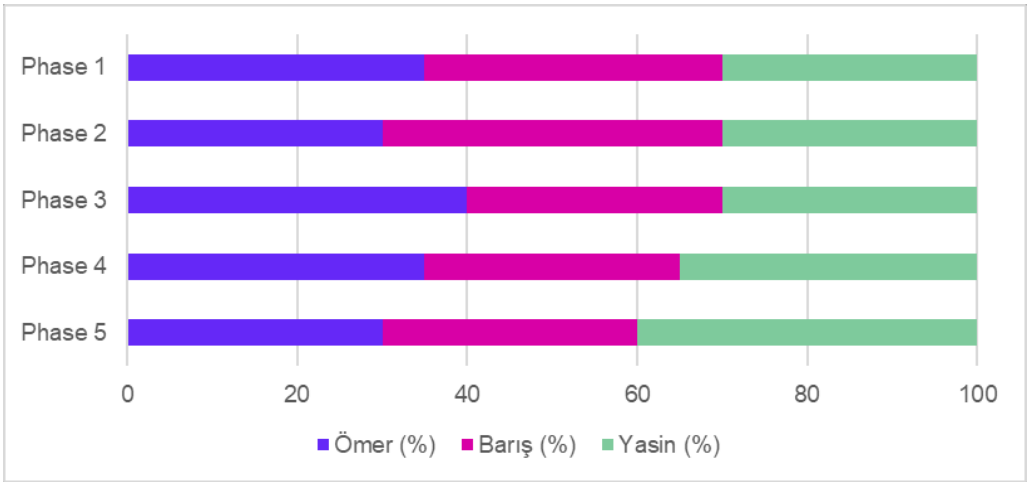

Figure 6: Work Distribution of Team Members.

9 SUCCESS FACTORS AND RISK MANAGEMENT

9.1 Measurability/Measuring Success

9.1.1 Objective 1: Split Learning Architecture Design

Develop and implement a split learning model architecture suited for low-power devices, dividing the model into client-side and server-side components to reduce computational load on the device.

Success Factor: The success of this objective will be measured by monitoring the computational load on the client devices. A successful outcome will show **at least a 30% reduction in CPU usage** on the client devices compared to a non-split architecture.

9.1.2 Objective 2: Data Preparation and Distribution

Prepare and pre-process data to be distributed across multiple devices, ensuring each device holds a partitioned dataset suitable for localized training.

Success Factor: This objective will be met if each device is assigned a unique subset of the data, with each partition containing at least 1,000 samples per device for adequate training representation. Data loading times should be less than 5 seconds per device.

9.1.3 Objective 3: Privacy-Preserving Communication Protocol

Design and implement a communication protocol for parameter exchange that preserves data privacy by only sharing model updates (not raw data) between devices and the server.

Success Factor: The communication protocol should ensure that 100% of data transmissions contain only model updates (intermediate layer outputs or gradients) without transferring raw data. Privacy verification tests will confirm **zero instances of raw data sharing**, ensuring that sensitive data remains strictly on the local devices during each training iteration.

9.1.4 Objective 4: Model Performance Evaluation

Test and evaluate the model's performance on metrics such as accuracy, training time, resource utilization (CPU, memory, and energy consumption), and communication overhead on the clients.

Success Factor: Model accuracy should achieve at least **90% of the centralized baseline accuracy**. Training time per round on each client device should be under **1 minute**, with CPU and memory usage as outlined above. Communication overhead should not exceed **15% of the total processing time** per training iteration, maintaining efficiency in data exchange.

9.2 Risk Management

9.2.1 Limited Device Resources

- **Risk**: The Raspberry Pi devices may lack sufficient memory or processing power to handle certain model complexities.

- **Resolution**: Regularly monitor resource usage to ensure models fit within device limits. Optimize the model by reducing layer size or complexity if needed.
- **B-Plan**: If resource constraints become too restrictive, explore using alternative lightweight models specifically designed for low-power devices, or reduce the batch size and adjust training parameters to fit available resources.

9.2.2 Heterogeneous Data Distribution Among Devices

- **Risk**: Data may not be evenly distributed across devices, leading to imbalanced training and affecting model accuracy.
- **Resolution**: Monitor model performance to detect any imbalances in training contributions. Use weighted averaging to account for data discrepancies.
- **B-Plan**: Implement a more robust aggregation method that adjusts for data distribution variability or redistribute the data among devices as needed to achieve balance.

10 BENEFITS AND IMPACT OF THE PROJECT

10.1 Scientific Impact

This project represents a significant advancement in federated learning by optimizing split learning specifically for resource-limited environments. By addressing the unique challenges of deploying machine learning on constrained devices, such as Raspberry Pi, it contributes to important areas of research, including privacy-preserving machine learning, data sovereignty, and efficient decentralized computation. The insights gained from this project may serve as a valuable foundation for future studies, potentially driving new methods that enhance the scalability, efficiency, and privacy of AI systems on low-power devices.

To further share and validate our findings within the scientific community, we are planning to present outputs of this research at a relevant conference. This presentation will allow us to discuss our methodologies, share results, and gather feedback from experts in the

field, contributing to the broader dissemination and potential adoption of split federated learning for resource-constrained applications.

10.2 Economic/Commercial/Social Impact

The project is expected to result in a working prototype that demonstrates the feasibility of split federated learning on low-cost, resource-constrained devices. Although it's not designed as a commercial product, the prototype could inspire applications in sectors such as healthcare, smart cities, and personalized education. This approach enables secure, decentralized machine learning, which could improve access to advanced AI in areas with limited infrastructure. Additionally, the emphasis on energy-efficient computation aligns with sustainability goals, helping to minimize AI's environmental footprint and support the development of environmentally friendly technology solutions.

10.3 Potential Impact on New Projects

This project has the potential to pioneer further developments in federated learning for resource-constrained devices. Its successful demonstration could inspire a wave of research focused on decentralized AI for low-power environments, promoting advancements in edge computing. By highlighting practical techniques for privacy-preserving machine learning on accessible hardware, this work may lead to a new generation of projects dedicated to making AI more inclusive and applicable across diverse, resource-limited settings.

REFERENCES

[1] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., & Agüera y Arcas, B. (2017). Communication-Efficient Learning of Deep Networks from Decentralized Data. *AISTATS*.

[2] Liao, Y., Xu, Y., Xu, H., Wang, L., Yao, Z., & Qiao, C. (2024). MergeSFL: Split Federated Learning with Feature Merging and Batch Size Regulation. *ICDE*.

[3] Li, T., Sahu, A. K., Talwalkar, A., & Smith, V. (2020). Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine*, 37(3), 50-60.

[4] Thapa, C., Mahawaga Arachchige, P. C., Camtepe, S., & Sun, L. (2022). SplitFed: When Federated Learning Meets Split Learning. *AAAI Conference on Artificial Intelligence*, 36, 8485-8493.

[5] Gupta, O., et al. (2020). Split Learning for Health: Distributed Deep Learning without Sharing Raw Patient Data. *Journal of the American Medical Informatics Association*, 27(9), 1312-1319.

[6] Duan, Q., et al. (2022). Combined Federated and Split Learning in Edge Computing for IoT. *Sensors (Basel)*.

[7] Maas, A. L., et al. (2011). Learning Word Vectors for Sentiment Analysis. *ACL-HLT*.

[8] Go, A., Bhayani, R., & Huang, L. (2009). Twitter Sentiment Classification using Distant Supervision. *Stanford University*.

[9] Strack, B., et al. (2014). Impact of HbA1c Measurement on Hospital Readmission Rates. *UCI Machine Learning Repository*.