A Feasibility Study to implement Next Word Prediction Model using Federated Learning on Raspberry Pi

Shivani Tomar, B.Tech.

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Data Science)

Supervisor: Meriel Huggard

August 2021

Declaration

I, Shivani Tomar, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Shivani Tomar

August 31, 2021

Permission to Lend and/or Copy

I, Shivani Tomar, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Shivani Tomar

August 31, 2021

A Feasibility Study to implement Next Word Prediction Model using Federated Learning on Raspberry Pi

Shivani Tomar, Master of Science in Computer Science
University of Dublin, Trinity College, 2021

Supervisor: Meriel Huggard

Federated Learning (FL) is a distributed form of machine learning where models are trained on the local data available on edge/mobile devices without collecting user-sensitive data on the server. However, implementing FL on edge devices such as a Raspberry Pi has been challenging due to the limited computational resources available on these devices. This dissertation demonstrates the feasibility of implementing a next word prediction model using FL on resource-constrained Raspberry Pi devices. Long-Short Term Memory (LSTM), a variant of Recurrent Neural Network (RNN) is designed to predict the next word on the Reddit dataset. Cross entropy loss and accuracy of the trained model are evaluated. During the federated training, the performance metrics of the Raspberry Pi devices including memory usage, training time, and temperature were measured to establish the practicality of the designed solution. It was observed that next-word prediction LSTM model could be trained in a federated setting on real edge devices (Raspberry Pi). However, there is a need to scale the system to atleast 50 to 100 clients to get state-of-the-art results.

Acknowledgments

I would like to thank my supervisor, Dr. Meriel Huggard for her constant support and guidance throughout the course of this dissertation. Her insightful feedback and expertise provided the right direction to my research. I would also like to extend my heartfelt gratitude to my family and friends for their support and encouragement throughout the course.

Shivani Tomar

University of Dublin, Trinity College
August 2021

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the massive growth of the amount of data generated by the devices in the world around us, there has been a pressing need to incorporate this user-information rich data to enhance the end users' experience. This can be achieved by training machine learning (ML) models using this data to drive various mobile applications running on Artificial Intelligence (AI). Classical machine learning approaches require training data to be collected at a centralized location, most likely on servers located in a data center, which raises concerns regarding users' data privacy and the costs associated with moving the data over the network. These concerns have now been addressed by the introduction of a distributed machine learning technique called Federated Learning (FL) [8] which eliminates the need to collect and store training data at a central location. It allows models to be trained on end-user devices in a federated setting, and these are then combined by an orchestrating server to form a shared global model.

Natural Language Processing (NLP) is a branch of AI that aims to make computers understand and interpret the written text as well as spoken human language in a similar way as humans do. NLP has undergone tremendous improvements due to the advancement in deep learning and neural networks in the past few years. It finds applications in text processing, machine translation, and speech recognition. However, due to the complexity and size of the language models, it becomes extremely difficult to deploy and run these models on mobile and other edge devices. This is due to the limited resources available when dealing with such devices. However, federated learning comes to the rescue in this scenario, providing the ability to train language models on the user device itself without compromising the users' data privacy.

1.1 Motivation

The motivation for this study has been the challenge of implementing federated learning and language modelling on a resource-constrained device, i.e., a Raspberry Pi. A similar problem has been commercially realised on an Android mobile device through the development of Gboard [17]. This mobile keyboard application that is powered by features like auto-correction and next word prediction has already crossed over 1 billion downloads as recorded in 2019. Other similar applications like Swype and SwiftKey have also employed neural networks for making predictions [33] and gained popularity based on their improved performance.

There is an ever increasing number of users using hand-held devices like smartphones, smartwatches, sensors, and edge devices like Raspberry Pis. These generate and store a wealth of data that is mostly private but can be utilized to develop models which can prove to be much more accurate in making predictions about the users' behavior. This can be used to optimize the performance of various intelligent applications. This has been one of the key motivations for this research which is focused on implementing next word prediction using language models on Raspberry Pi clients.

A Raspberry Pi is a cheap and lightweight mini-computer board with limited computational capacity when compared to high-end laptops and desktops powered by fast CPUs and GPUs. Nevertheless, it is still quite powerful when compared with other mobile edge devices. It is currently being used in Internet-Of-Things (IoT) applications in home automation and human activity detection to name a few. Raspberry Pis have the potential to be used in various intelligent applications, which make them an ideal candidate to explore the possibilities of implementing next-word prediction task on this device through federated learning. However, Raspberry Pis have their own set of challenges [19] when it comes to training language models on them. They have limited memory available which reduces their capacity to perform the higher order computations involved in data processing and model training. They are also potentially bounded in terms of energy consumption and repeated communication of model updates to the server over the network can prove costly and slow down the learning process.

1.2 Research Goals

The objective of this research is to implement and analyse the feasibility of training a next word prediction model on a resource-constrained device like a Raspberry Pi. Instead of using a simulation framework, the solution needs to be implemented using a framework that supports deploying ML models on real devices in a federated setting leading to a

realistic evaluation of the factors affecting the performance of the FL system.

The next word prediction model will need to be designed considering the nature of the data on which the predictions have to be made. Since conversational text data is sequential in nature, neural networks capable of storing and processing contextual information need to be used. Since the idea of implementing next word prediction on a Raspberry Pi is novel, the research question that is being investigated here is the analysis of the hardware limitations of the Raspberry Pi clients based on RAM usage and the training time taken to complete the federated learning process which involves running a language neural model.

1.3 Dissertation Layout

Below is a summary of the outline of this research document:

Chapter 2: A Literature Review covering the concepts required for understanding the federated learning system along with the challenges of implementing the same on a resource-constrained device such as a Raspberry Pi.

Chapter 3: Details the design and implementation of a next word prediction model on resource-constrained devices, including the framework used and the challenges faced.

Chapter 4: The results obtained and how they are evaluated against the State-of-the-art.

Chapter 5: Concludes this dissertation with final reflections and possible directions for future work.

Chapter 2

Literature Review

This chapter presents a detailed description of the concept of federated learning and how it can be implemented on resource-constrained devices such as Raspberry Pi. It further discusses how it is being currently used in next word prediction out of the many applications in other areas like image recognition, smart manufacturing, and financial risk prediction to name a few. It further explores the different implementations of federated learning including both federated simulations and those involving real devices like Android phones. The chapter then moves onto a discussion of the Flower Framework which overcomes the shortcomings of current simulation frameworks and allows for an understanding of the computational costs associated with FL on edge devices like Raspberry Pi and NVIDIA Jetson devices. Consequently, the chapter summarises the related work done and how it can be used in this research to build a federated implementation of a Long Short Term Memory (LSTM) neural network for predicting the next word on a Raspberry Pi using the flower framework. In particular, this chapter reviews the main papers which have motivated this dissertation on the feasibility of implementing Federated learning for next word prediction on Raspberry Pi.

2.1 Federated Learning Overview

Federated Learning (FL) is a distributed machine learning technique where model training is decentralised. This eliminates the need to transfer sensitive user data to a central server for model training. Instead, the clients send model updates to the server while the training is performed locally. This idea was introduced by Brendan McMahan et al. [8] in 2016 and since then it has been an active area of research as it exploits the rich data available at user-held devices without compromising the security and privacy of that data. This has led to the development of highly intelligent applications [27] on smartphones, smartwatches,

and other sensor devices which are powered by the user's own data, providing them with enhanced user experiences.

A large number of clients participate in the training process to send model updates to the server, which then aggregates the client updates using FederatedAveraging algorithm [8] to produce an improved global model. This whole cycle makes one pass of the training process. Once the client updates are applied to the global model, they are then discarded as they serve no purpose after being applied.

The main objectives that led to the development of this federated style of learning were to achieve the training of machine learning models on user-held data and harness the power of this data, which in most cases is unbalanced and non-IID (identically and independently distributed) in nature. This is because data belonging to each client is different and does not follow the same distribution. Maintaining the privacy of the user data was one of the biggest motivators in the development of federated learning.

2.1.1 Stages of a Federated Learning Process

There are a set of well-defined steps followed in developing a model in a federated setting as shown in Figure 2.1.

The use case for each application scenario may differ slightly, but the broad idea remains the same. A typical process that is followed by any FL scenario comprises of the following steps as outlined in [25]:

1. The group of clients participating in the training process based on the selection criteria is decided by the centralised server.

2. Common global model parameters are shared with the selected clients by the server, which becomes the starting point for each client.

3. Each client trains the base model as per the training instructions provided by the server on its local dataset and sends the updated model parameters back to the server.

4. On receiving the updates from each of the selected clients, the server aggregates them using an algorithm.

5. The server updates the initial global model by applying the aggregated update received from the set of clients, which completes one round of the training process.

2.1.2 FederatedAveraging Algorithm

Stochastic gradient descent (SGD) is an optimization approach adopted in machine learning in place of simple gradient descent methods. Simple gradient descent method involves iteratively optimizing the objective function that needs to be minimized by finding its

gradient corresponding to each data point. However, due to the large computational overhead of calculating the gradient on the entire dataset, simple gradient descent becomes unsuitable for problems with large datasets. SGD is an approximation of the gradient descent method as it uses a single data point randomly in each iteration to calculate the gradient, which leads to faster convergence. However, calculating the gradient averaged over a random minibatch of samples provides a better approximation than the true SGD where the gradient is computed on a single random sample. In [8], it was proposed that SGD be used in preference to a simple gradient descent approach to solve the federated optimization problem. When SGD is used in FL, only a fraction C of clients is selected from the entire set K of clients that receive the global model parameters. This makes C a parameter that decides the batch size. The resulting algorithm is termed FederatedSGD algorithm.

Each client i calculates the average gradient descent $g_i = \nabla F_i(w_t)$ where w_t is the initial global model provided by the server and submits an update $w^i_{t+1} \longleftarrow w_t - \eta g_i$ to the server. A weighted average of all these client updates is performed by the server to get the resulting global model. For each client, it is possible to have multiple iterations of the update before it is sent to the server for averaging. This algorithm is termed as FederatedAveraging (FedAvg) [8].

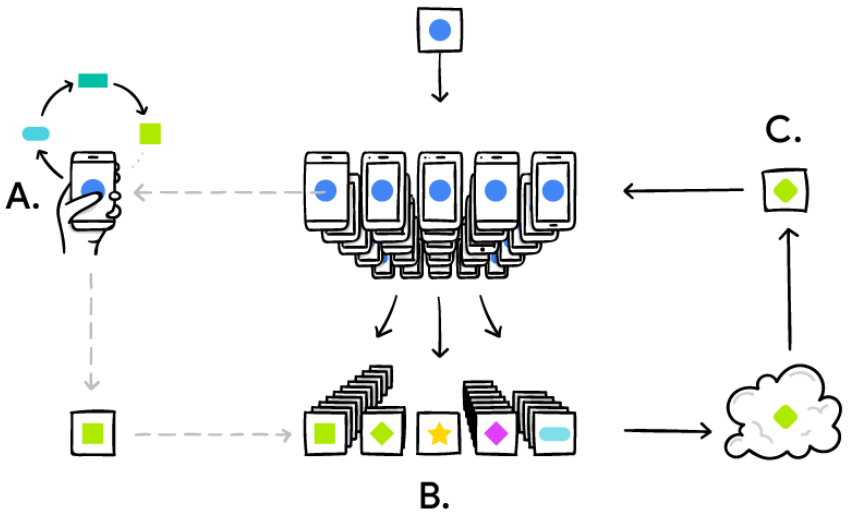

Figure 2.1: A pictorial representation of the Federated Learning Process from [27] (A) Each client device computes an update based on training with the local data, (B) Updates from all the clients are aggregated to form a shared global model, (C) The new global model is shared again with all the clients and the process is then repeated.

2.2 Next Word Prediction using LSTM RNN

Language models are now extensively used in Natural Language Processing (NLP) applications like text generation, speech recognition and translation. Next word prediction is one of the most important tasks in NLP, which is now a widely used feature in Google keyboard application called Gboard. Al-Rfou et al.[1] described how the context of the conversation, as well as the history of the participants, improves the prediction accuracy of the user's response in the conversation. Recurrent Neural networks (RNN) have proved useful in modelling sequential data like conversational text and for generating novel responses. However, RNN suffers from the gradient vanishing problem which was resolved by Long Short-Term Memory(LSTM) developed by Hochreiter et al. [18] in 1997. This has been discussed in more detail in Section 3.1 of the next chapter. LSTMs have achieved state-of-the-art results [22] in language models as they introduced a mechanism to factor in long-term dependencies by preserving the contextual information in the input.

2.2.1 Initial implementation of FL in next word prediction

The first practical implementation of federated learning for next word prediction was brought together by Rajiv et al. in the paper, "Federated Learning for Mobile Keyboard Prediction" [17]. This paper demonstrates the real world application of federated learning in predicting the next word to be typed by users on mobile keyboards implemented by the Gboard application. Prior to the use of LSTM, next-word prediction models were built using finite state transducers (FSTs) [30]. FST is a finite state machine which operates on two tapes, namely, input and output. It reads the input tape which is essentially a sequence of characters in a word and maps it to the output tape. It is used for predicting the next word which involves searching for the highest order n-gram state from the previous text typed by the user and returning n-best output labels. However, in [17], it is highlighted that the LSTM RNN performs better than n-gram FSTs both in terms of CPU-consumption and inference time latency. The LSTM trained by the authors proves advantageous over a simple RNN as the former uses a variable-sized context window to capture the long-term dependencies when predicting the next word. It also reduces the number of parameters without having an adverse impact on the model accuracy. This choice proves beneficial when the model needs to be trained over computationally constrained devices. The focus of this dissertation is to check whether it is feasible to implement an LSTM RNN on a Raspberry Pi.

2.2.2 Enhancements in FL and related work

The initial implementation of federated learning on a commercial scale sparked extensive further research in the area, which led to enhancements in the field through the development of new techniques. These will be reviewed later in this section.

Federated learning has also been used to improve Gboard query suggestions [37] without exposing the user's private data to the server and keeping the mobile constraints under strict consideration. The authors have demonstrated the process of improving the query suggestions via two stages: the first stage uses the baseline model which creates predictions for the query, while the second stage comprises the triggering model trained in FL fashion which filters out the less accurate suggestions.

To enhance the users' experience, Rajiv et al. showcased the effect of personalization on the global model in [35]. The authors utilised the same next word prediction model developed in [17] which was trained using FL on a number of mobile devices. The research presented by the authors was a first attempt at analysing personalization strategies and evaluating the performance of the federated model. The variation in the model performance was studied by tuning the following hyperparameters: training batch size, learning rate, and stopping criteria. Since these strategies used the users' on-device data which varied both in quality and quantity, it led to differences in the model improvement accuracy. As expected, the study suggested that users with more data saw significant improvement. Increasing the learning rate from 0.1 to 1.0 led to improved performance accuracy for most users (39%) even when a sizeable chunk of users encountered degradation in their performance when the learning rate was increased to 1.0 for a smaller batch size of 20. Since this research initially evaluates the federated setting using only 2 clients (Raspberry Pis), personalization may not lead to similar results as stated by the authors in the above study.

In another interesting work done by McMahan et al.[23] showed ways in which to improve the communication efficiency in Federated Learning. A large number of devices connected via a network are involved in training a federated model, which makes the communication of the model updates to the server the biggest bottleneck if it is not performed in an optimised way. The paper focuses on the reduction of uplink communications as the uplink speed of the network provider is always slower than the downlink speed. An LSTM model for next word prediction is trained on the Reddit comments dataset [5] by the authors. They used two common approaches to reduce the communication overhead, namely, structured and sketched updates. The sketched approach used in LSTM training involves calculating the full update initially. This later undergoes lossy compression before being sent to the server for aggregation. The results show that increasing the number

of clients participating in a training round can help to reduce communication while maintaining a similar level of accuracy with negligible drop. As this dissertation is focused on training an LSTM on a Raspberry Pi where communicating model updates to the server can be expensive, the above techniques prove beneficial in achieving the desired results without compromising on the model accuracy.

In [38], Yu et al. developed a highly efficient word prediction neural model that not only optimises the memory usage but also meets the time constraints needed for a seamless experience when using a mobile keyboard. Out of the various compression methods employed so far, this work involved the use of a shared matrix factorization technique in combination with knowledge distillation. Shared matrix factorization helps reduce the number of parameters and along with knowledge distillation, it reduces the iterations required for updating those parameters. The resulting model has been successfully launched for commercial use on Samsung mobile devices.

Having discussed the various approaches being studied by the researchers, it is clear that some techniques might help improve the predictions of language models like LSTM RNNs, while others prove to have little impact on the final models. Training the same LSTM variant used in [17] on a Pi can yield similar results as long as the available dataset and the training parameters have minor differences but it needs to be ascertained how the hardware limitations of the Pi might affect the final outcome.

2.2.3 Security considerations related to FL

Federated Learning has proved to be extremely helpful to address the data privacy risks associated with sending and storing user data in a centralised location. However, there are still concerns about the potential leak of user data from the model updates. It is also potentially prone to adversarial attacks.

A number of security and privacy concerns which have been highlighted in past research. For example, Carlini et al.[11] demonstrated that an RNN trained on user's language data can reveal sensitive information like social security numbers as a result of unintentional memorization of unique or rare sequences in the training data by the language model. There have also been scenarios where having the previous model and current update from a particular user can reveal the raw training example for that user [21] [32]. Another important category of attacks against the FL model includes adversarial attacks which can be aimed at deteriorating model performance as well as at inferring sensitive data. Data poisoning is an adversarial attack aimed to degrade model performance by sending malicious updates to the server to manipulate the globally aggregated model by a suspicious client. Focusing on the security aspect, FL is also susceptible to

Byzantine attacks where the intent of the malicious actor(s) is to prevent the final model from converging.

Some of the main strategies [25] that can be used to address the issues outlined in the previous section are the use of differential privacy [14] to communicate noisy data along with the intended updates, homomorphic encryption [39] which involves operating on encrypted data, and secure multiparty computation protocol (SMC)[13].

- Differential Privacy (DP) as a technique to provide higher level of privacy quantifies the amount of sensitive information that can be leaked. It introduces some level of uncertainty into the final model which is capable of hiding the contribution of a single user's device data. It can be achieved in two flavours namely, global differential privacy and local differential privacy[21].

  Global DP means that the client updates shared are made private from all third parties except the orchestrating central server which aggregates these client updates, whereas local DP involves masking these updates from the third parties including the central server. It should be noted that differential privacy completely rules out the possibility of unintended memorisation of sensitive information, which is a major concern in the area of language models trained on public dataset [28]. It is important to keep the model updates/iterates masked from the various parties involved in the system which includes the central server and the remaining clients taking part in the model training.

  There is a need to come up with the combination of differential privacy, which creates a balance between a strict global DP and a pure local DP. There are a number of approaches in which DP can be guaranteed in a distributed manner to reap the benefits of both global DP without relying on the central server. These include secure aggregation[16][7], secure shuffling [6] etc, however, it depends on the specific scenario we are faced with.

- Multiparty Computation (MPC)[13] is also a cryptographic protocol that can be used in an FL setting to hide the model updates from the different clients involved. The server encrypts the model updates before sending to the clients and the clients in turn compute the updated parameters in an encrypted form without the need to decrypt them. The clients then send back the updates to the server in an encrypted form for aggregation. Homomorphic Encryption[39] can be used as a tool to implement MPC by allowing computation of functions on cypher texts while keeping the real values hidden[21].

- Trusted Execution Environment (TEE) [21] are secure enclaves that allow running

specific components of code on a remote machine without the need to trust even the owner of the machine and this is made possible by limiting the capabilities of any parties involved like the client or the server. The code run on a TEE can be verified that it was executed faithfully and privately without revealing details at any stage of the execution of the code.

Implementing DP requires deciding which form would be best suited to the specific use case scenario. It can be implemented at the global or local level while extending it to a more granular level, for example, user-specific or even sample specific. As DP will involve adding noise to the updates, it becomes necessary to perform extra computation at each user's device to reduce the communication rounds to train the final model. This extra computation will act as an overhead as this dissertation is focussed on carrying out the computation on a Raspberry Pi 4. This might crash the device due to an increase in processing power required or could lead to possible delays in training. In addition, specific consideration needs to be given to the fact that distortion caused by the additional noise used in DP can lead to lowered model accuracy, therefore a trade-off between security and accuracy is desirable.

The concerns that arise out of the possibility of using multiparty computation are the party that holds the secret key used for encryption. A probable solution could be an external non-colluding party, but such a scenario would not be available in the FL setting being considered in this dissertation where there will only be a central server and participating Raspberry Pi clients. A workaround to this could be distributing the secret key to the clients.

Since this dissertation aims at implementing federated learning on a resource-constrained device such as a Raspberry Pi, it is crucial to understand and analyse the impact of using the different privacy ensuring techniques mentioned above. The specific use case being investigated involves training a language model i.e., an LSTM RNN on a Raspberry Pi where the communication overhead can become a bigger bottleneck than it would have impacted in a classical machine learning scenario. The direction of exploring security and privacy aspects in FL is itself quite vast and widely researched at the time of writing this dissertation. It can be taken up for future research and will not be explored in depth as part of this study.

2.3 Raspberry Pi and the challenges imposed

2.3.1 A Brief Introduction

RaspberryPi is an ATM card-sized computer board as shown in Figure 2.2 that came into existence in 2012 at the Computer Laboratory of the University of Cambridge [9]. The idea behind developing it was to facilitate and foster the interest of school-going children in computing and programming. It provided a tiny but cheap and powerful platform which was made affordable (priced at 35$) by its developers at the Raspberry Pi Foundation to promote computer education among the less privileged.

Figure 2.2: A Raspberry Pi Board

The core components that make up this tiny computer are an ARM processor and the program memory (RAM). It is powered by a USB power supply and has USB sockets for connecting peripherals like a mouse and keyboard. It also has an HDMI port for connecting monitors or televisions which renders it usable as a full-fledged desktop PC. A variety of different linux-based operating systems can be used by the Pi, the most common being Raspbian. The Raspberry Pi can be connected to the internet via both wired (Ethernet cable) and wireless (through WiFi) connections. It also houses GPIO (general purpose input and output) pins which help connecting the Pi with electronic

circuits and embedded devices, facilitating a variety of IoT projects and applications.

2.3.2 Challenges associated with working on Raspberry Pi

The Raspberry Pi is a resource-constrained device when compared with the current generation of desktops and laptops. It is noteworthy that the Raspberry Pi does not have an inbuilt hard drive rather everything gets stored on an SD card that is mounted on it. This includes the image of the OS and the data required.

Limited memory availability makes it difficult to perform operations that are computationally demanding on a Raspberry Pi. The minimum hardware requirements to train a next word prediction model on an Android keyboard application were stated by Hard et al.[17] as a minimum of 2GB of available RAM. For these requirements to be fulfilled in a Pi, the amount of locally stored data needs to be monitored to avoid overloading the device.

The next big challenge when working with a Raspberry Pi is the cost of communication between the Raspberry Pi clients and orchestrating the server over the network. Federated learning involves sending model updates to the server for aggregation. However, communicating more often can lead to faster convergence, but it comes at a cost which might be prohibitive. Therefore, it becomes necessary to formulate ways to achieve model convergence while keeping the number of communication rounds to a minimum.

Training neural networks is particularly challenging on a Raspberry Pi due to the processing capabilities required for weight parameter computation. Not all machine learning frameworks provide support for deploying and running models on these resource-constrained edge devices. There are various approaches being researched to find ways to train complex and accurate models while keeping the computations limited through techniques like model pruning, resizing input etc.

Due to the exceedingly large amount of data available at the edge devices and the value it can add to improve the user experience, it is interesting to explore the possibilities of implementing and evaluating FL on resource-constrained devices like a Raspberry Pi. Training an LSTM through a network of federated client devices like the Raspberry Pi can potentially become a future use case scenario considering the popularity gained by Raspberry Pis in the past few years and the emerging trends that are supporting the use of such devices to drive the Internet-Of-Things (IoT). This will be considered in the next section of this chapter.

2.4 Federated Learning Frameworks for IoT

Recent growth in the area of federated learning has led to the development of several open-sourced frameworks for implementing federated learning. The following sections provide an overview of the ones that are actively used in the ongoing FL research.

2.4.1 TensorFlow Federated (TFF) by Google

One of the most popular open-sourced frameworks for federated learning currently being worked upon is Tensorflow Federated [34]. This has been developed by Google engineers and is open-sourced inviting community contributions. TFF has been designed to execute simulations of federated learning scenarios for developers and researchers to investigate and analyze FL models and use-cases.

Tensorflow provides interfaces to support the implementation of federated learning, including both training and evaluation using two main APIs:

(i) The High-level Federated Learning (FL) API layer of TFF can be used by developers to implement FL training and evaluation on top of their own models. The namespace under which these interfaces have been defined is tff.learning.

(ii) The higher level layer is built upon the lower level Federated Core (FC), which is also responsible for creating a run-time environment and acts as the building block for the FL layer.

Furthermore, TFF provides the users with base classes to implement FL tasks for both training and evaluation of the models generated by the users using TensorFlow and Keras. Different methods are provided for the aggregation of client updates as follows:

- MeanFactory: calculates a weighted mean of model updates for different clients and the resulted mean is placed at the server
- SumFactory: calculates a simple sum of model updates for different clients and the result is produced at the server.
- DifferentiallyPrivateFactory: calculates model updates on different clients based on the DPQuery set accordance to the differential privacy (DP) algorithm and places the resultant aggregate at the server.

To understand how the tensorflow federated simulations work, there are three main annotations used:

- @tf.function: decorates i.e. extends the function to be used inside federated computations.

- @tff.federated_computation: python wrapper used to indicate the enclosed computation will be a federated computation.

- @tff.tf_computation: python wrapper invoked with or without a function returning an instance of tff computation.

The Federated Core API can be used by developers to come up with new federated algorithms apart from the available implementations provided by TFF as it provides templates for the different stages of the federated learning process including aggregation of model updates and evaluation of the accuracy metrics. However, the biggest drawback of the TFF framework is that it does not support the implementation of a real federated learning network, rather it provides just a simulation framework to facilitate FL research. This makes TFF unsuitable for use in this dissertation as it focuses on implementing FL on a real device (i.e., a Raspberry Pi) to understand whether it is feasible to train an LSTM for next word prediction on a Pi.

2.4.2 Flower Framework

To understand the real costs associated with FL in terms of computation and communication associated with the deployment of FL on heterogeneous clients like mobile and edge devices, Daniel et al. developed the Flower Framework [4] which provides the flexibility of furthering the state-of-the-art in FL research. It introduced a platform to execute real-time FL workloads involving heterogeneous devices at scale with the capability of adding up to 1000 clients to the federated training process.

This framework has been developed keeping in mind the following goals:

- To provide compatibility with both existing and currently being developed ML frameworks in order to preserve the efforts invested in the ML technology so that they can be easily integrated with the FL framework.

- To provide compatibility with heterogeneous clients keeping in mind real world scenarios where a range of devices including mobile phones, tablets, embedded devices are used.

- To ensure interoperability with ML frameworks keeping the engineering overheads at the minimum possible costs.

- With the growth of IoT and mobile devices, Flower aims to provide scalability in terms of the number of clients that can participate in the FL training and evaluation.

- Aims to make the framework extensible to new FL approaches and algorithms to facilitate research in the field.

The architecture of the Flower framework is based on two main partitions, namely, the server side and the client side. The server side is responsible for coordinating the entire FL process based on the available set of clients. The server side comprises of three main building blocks as follows:

- RPC Server: It is used for connection establishment between the client and server along with sending and receiving message updates.
- Federated Learning Loop: It is responsible for coordinating the whole process of FL.
- Strategy for Aggregation: This allows the user to choose the strategy for federated averaging, which decides the specifications for clients to be selected, training parameters, and the model evaluation mode.

The client side of the architecture is responsible only for acting on the instructions received through the strategy configuration for both training and evaluation. The messaging protocol used for connection and message exchange between server and clients is called the Flower Protocol. There are two types of messages: Training instructions (sent by the server) and connection messages (exchanged between both server and clients).

The unique property of this framework is that the server remains oblivious to the type of client devices involved in the process.This helps users to experiment with different heterogeneous clients with varying ML frameworks implemented on a variety of platforms and programming languages.

Based on this novel framework, Daniel et al. [26] performed experiments for training on a combination of embedded devices (NVIDIA Jetson and Raspberry Pi) using a flower client implemented in Python. A docker image specific to the client platform was created to enable training on a variety of devices including Android phones, Raspberry Pis, smartwatches, Nvidia devices. Experiments were conducted to analyse the system costs associated with FL using two main datasets, CIFAR-10 [24] and Office-31 [31] on Android smartphones and Nvidia Jetson devices. The results based on the metrics showed that increasing the number of epochs led to increased accuracy, albeit at the cost of more energy and training time. Similarly, training on the Office-31 dataset showed that increasing the number of participating clients led to higher accuracy but again at the cost of higher energy utilization.

2.5 Summary

This chapter reviewed the extensive research being conducted in federated learning, specifically in language modelling tasks like next word prediction and the challenges it poses when implemented on Raspberry Pi. It provided insights into the enhancements and advances in federated learning, discussing in detail the security considerations required when implementing FL. It also included a brief discussion on Raspberry Pi and the issues that may arise during training and inference of deep models due to the limited computational capabilities of such devices. The most popular frameworks for both simulation and deployment on real devices being currently used for experimenting with federated ML workloads were presented. The background research detailed in this chapter will help the reader appreciate the design and implementation of the proof of concept involved in this novel implementation of language models on Raspberry Pi.

Chapter 3

Design and Implementation

This chapter discusses the details of the design and implementation of the next word prediction model on a resource-constrained device, i.e., a Raspberry Pi. It clearly lays out the design choices made during this dissertation and builds upon the knowledge acquired from the previous chapter on background research and related work that has been done in this area. It begins with a discussion of Recurrent Neural Networks (RNN) and explains their suitability to model text data. In addition to that, it also explains the problems with RNNs and how LSTMs resolve those issues. It also highlights the challenges and bottlenecks that were encountered in the effort to train this model on Raspberry Pi FL clients using the Flower framework.

3.1 Recurrent Neural Networks (RNN)

Recurrent Neural Network (RNN) is a type of artificial neural network that is best suited for modelling data that is temporal in nature [36]. In contrast to feed-forward networks which work on a fixed length of context when processing information, RNNs possess the ability to store information to be used later while utilizing variable length contextual window. The input provided at a given time step in RNN is a function of both the current input as well as the information from the previous layer. This leads to the information stored in the previous layers getting passed on to successive layers in the network which is then utilized in making predictions.

RNNs are known to achieve the best results in language modelling tasks as shown in [3] [22]. This motivates the use of RNNs in this dissertation as the text conversational data is sequential in nature. However, as the sequences of words in the text get longer, it tends to forget the information learnt and is unable to handle long-term dependencies due to the problem of exploding and vanishing gradients. This arises due to the repeated

back-propagation of the error gradients as part of the learning process. In this process, the gradients which are greater than 1 tend to become larger at each time step due to repeated matrix multiplication, resulting in an unstable network. Hence the term exploding gradient. Whereas, the gradients with a value smaller than 1 tend to decay exponentially leading to no learning. This led to the development of LSTMs which use gates to control memorization and learn long-term dependencies.

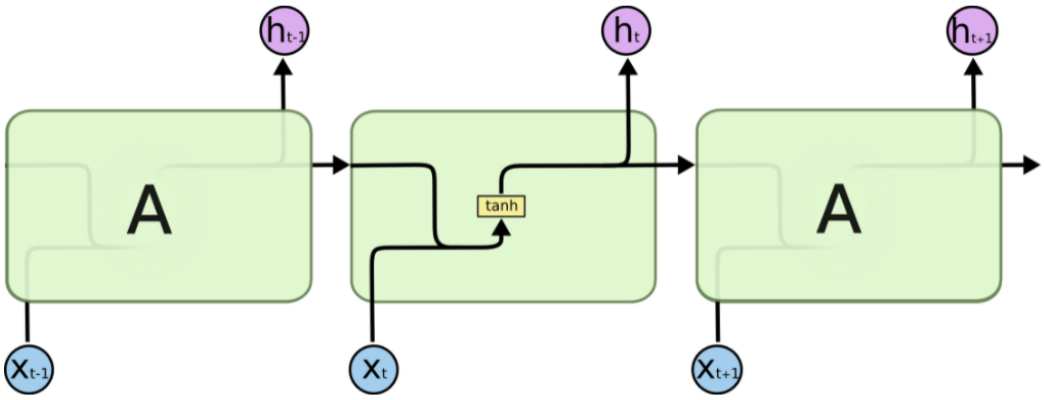

Figure 3.1: A Vanilla RNN Cell

3.1.1 RNN Variant - Long Short Term Memory (LSTM)

LSTM network is a type of Recurrent Neural Network which was developed [22] to resolve the shortcomings of RNN, which led to the exploding and vanishing gradient problem. LSTM consists of gates which control the information that needs to be stored and what needs to be discarded, resulting in a faster learning process. This architecture helps in bridging the long-time lag of up to 1000 discrete time steps. The difference in the internal architecture of a vanilla RNN and an LSTM has been illustrated below using the figures taken from [29].

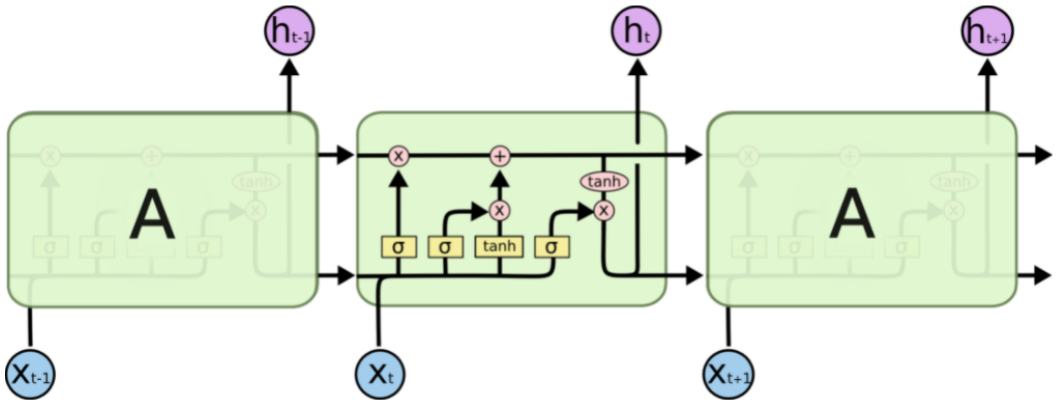

Figure 3.2: An LSTM with four layers

The Figure 3.1 shows a vanilla RNN cell with a single layer of tanh function, input X_t at time t, and the hidden state h_t at time t which is the output of the current cell state. However, an LSTM has a different internal structure where the tanh layer is replaced by four layers which interact with each other regulating the flow of information through the cell as shown in Figure 3.2.

There are three gates which have been implemented to control the flow of information inside the LSTM Cell, which are described as follows:

- **Forget Gate**: The forget gate f(t) helps in forgetting unnecessary information while retaining only important information. The same is achieved using a sigmoid layer which takes in output from the previous layer h_{t-1} and the current input x_t combined with bias b_f and returns the output of 1 which implies "remember completely" or 0 which implies "forget completely". The final output of this gate is $f_t * C_{t-1}$ where C_{t-1} is the previous cell state.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{3.1}$$

Figure 3.3 below shows the flow of information in and out of the forget gate.

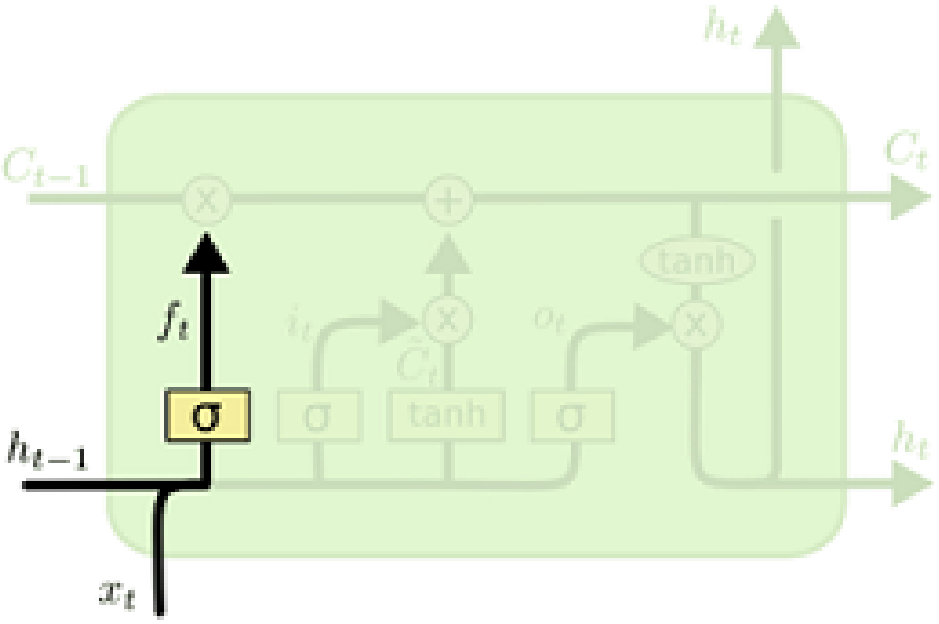

Figure 3.3: Forget Gate of LSTM Cell

- **Input Gate:** The input gate works in two stages as shown in Figure 3.4 and Figure 3.5. During the first stage, the sigmoid layer decides which values need to be updated and the next stage comprises the tanh layer which produces a vector $\tilde{C}_t$ of the values required to be added to the current state.

$$i_t = \sigma(W_f.[h_{t-1}, x_t] + b_i) \tag{3.2}$$

$$\tilde{C}_t = \tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{3.3}$$

Next, we combine the steps performed previously to forget the information from the previous cell state which is not needed given by $f_t * C_{t-1}$ and adding it to the information that is needed given by $\tilde{C}_t * i_t$.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.4}$$

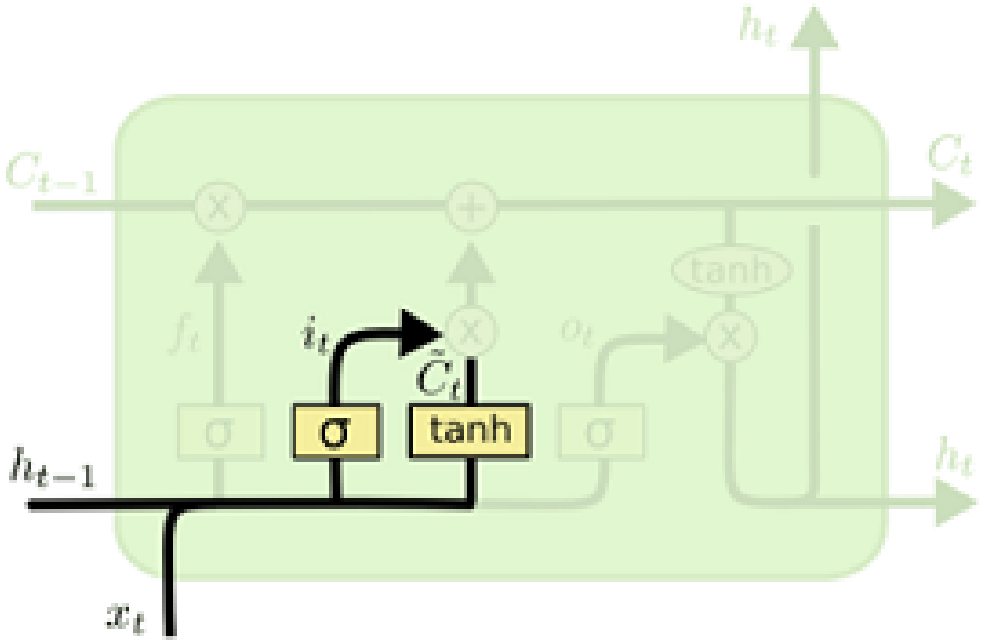

Figure 3.4: Input Gate at Stage 1

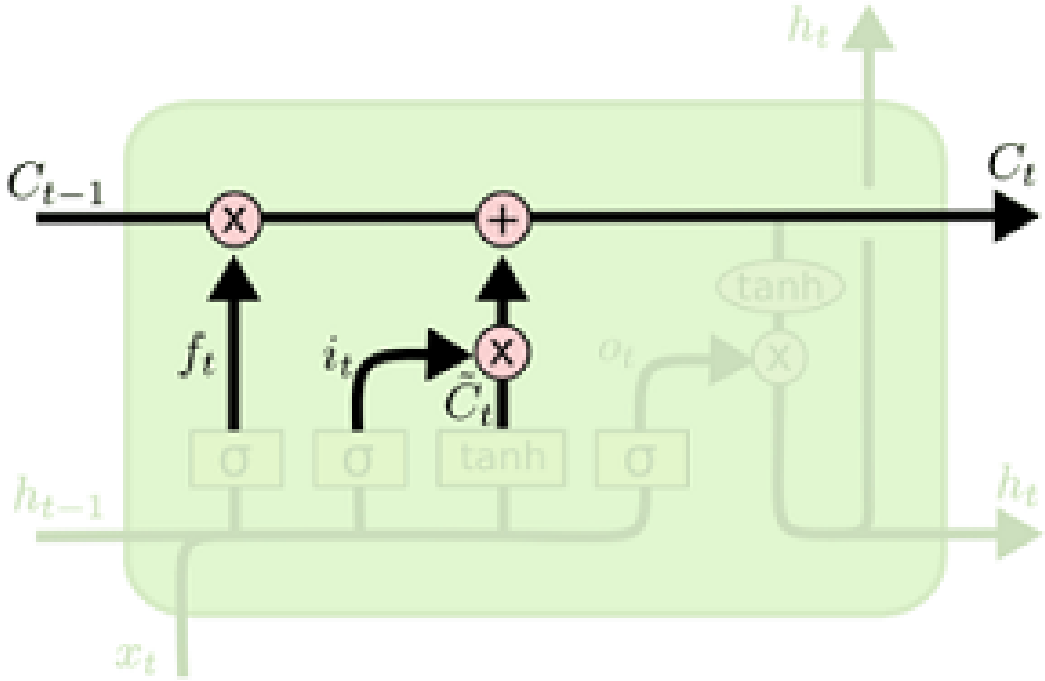

Figure 3.5: Input Gate at Stage 2

- **Output Gate:** The output gate decides the final output of the current cell state. This is done by the sigmoid layer which returns the information to be sent as output. The tanh layer is used to move the values of the cell state in the range of -1 to 1

which are then multiplied by the output of the sigmoid layer to get the final output. Figure 3.6 shows the output gate as described above.

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{3.5}$$

$$h_t = o_t * \tanh(C_t) \tag{3.6}$$

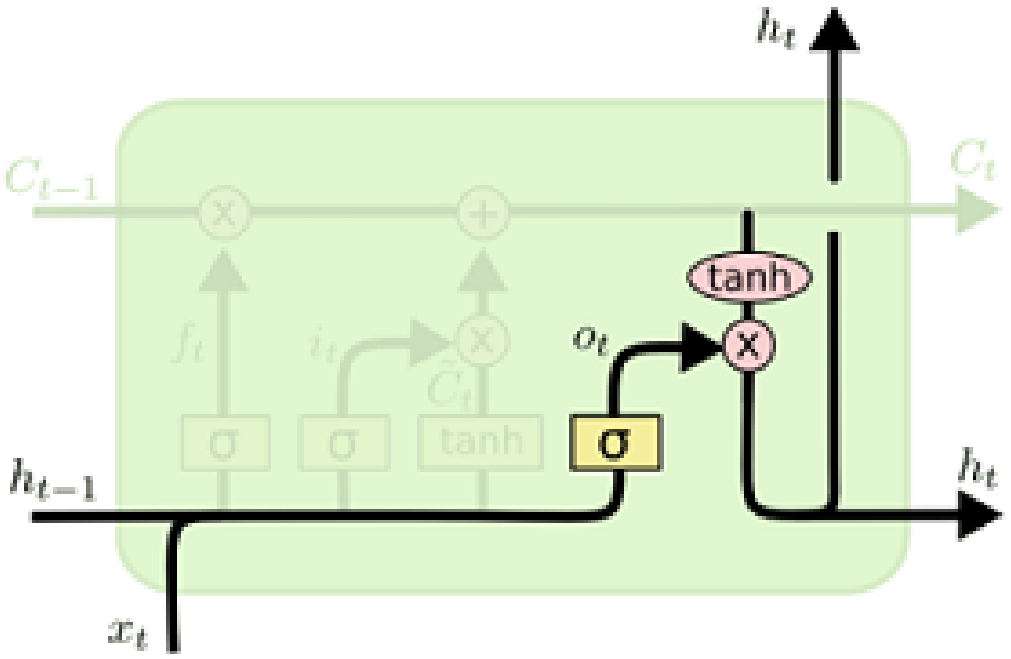

Figure 3.6: Output Gate of LSTM Cell

3.2 FLOWER Framework

Flower framework has been used for this dissertation as described in the previous chapter. The driving force behind using this framework is its ability to integrate federated learning with the real device landscape, which is the main goal of this study.

The architecture of the framework is shown in Figure 3.7 and is broadly divided into two core components of a federated learning environment, namely, the server and the client. There are two sets of computations that are performed as part of functioning of the core framework:

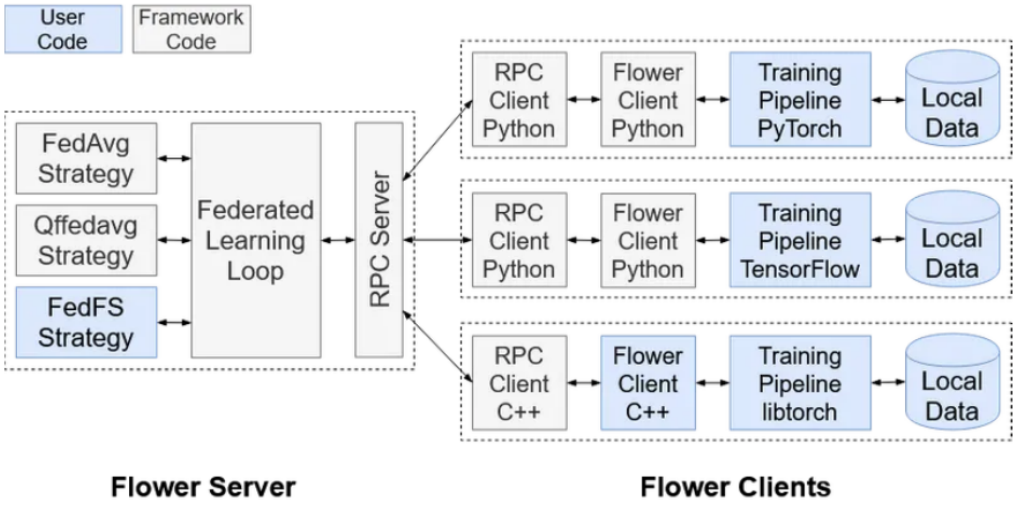

Figure 3.7: Architecture of Flower Framework, image from [4]

- **Global Computations:** The computations that are performed centrally at the server level to control the federated learning process together comprise the global logic implemented.

  The RPC server is responsible for establishing a network connection and initiating communication between the server and the federated clients. Flower Protocol messages are also exchanged using the RPC server. The communication protocol provides interfaces defining the server-side methods that can be called remotely by specifying the serialised model parameters which in turn expects return messages from the client in the form of updated model parameters or evaluation results like loss and accuracy.

  The next core element at the server level is the strategy which refers to the averaging algorithm used to combine the model updates received from the participating clients. FederatedAveraging (FedAvg) is the strategy used in this implementation. However, the framework provides the freedom to choose other algorithms or customize a new strategy altogether. The decision making logic is placed in the strategy which configures each round of the federated training by sending instructions to the clients through the RPC server, receiving updates from them, and aggregating the results.

  The following methods and classes are defined in the framework to support server-level operations described above:

  start_server(): This method is used to start the server with the help of gRPC transport layer. The parameters passed in this method include *server_address*: IPv6 address of the server, $config$: a dictionary which specifies the number of

rounds of federated training to be performed, and *Strategy*: implementing the base class flwr.server.Strategy defaulting to server.strategy.FedAvg if not specified.

server.strategy.FedAvg: This class is used to configure the parameters for the federated averaging strategy, including the fraction of clients used both in training and validation along with the minimum number of clients that need to be specified for both training and evaluation. It also provides the function: $on\_fit\_config\_fn$ which can be called to send training configurations to the client in each round for controlling the parameters like batch size, learning rate from the server side itself. Similarly, the function: $on\_evaluate\_config\_fn$ is used to customise the evaluation at the client.

- **Local Computations:** These computations majorly involve the local logic implemented at the client side to facilitate training and evaluation on the local dataset available at the client. Flower provides a client abstraction class which provides an interface to the server for interacting with the client. The benefit of using this class is that user-provided workload specific code can be executed without worrying about the details of network connection and message exchange formats. The convenience class provided by Flower used in this implementation is NumpyClient, which contains the following three methods:

  get_parameters: This method returns the model weights.

  fit: This method involves training the received model weights on the locally stored dataset and providing the updated model weights back to the server.

  evaluate: This method evaluates the local model weights on the locally held test data.

  Once the above interface class is instantiated, the function *start_numpy_client*() is used to start the client which takes in the server_address and the instance of the class created above as arguments.

3.3 Data Collection

The main objective of this dissertation is to check the feasibility of training an LSTM model on an embedded device like Raspberry Pi. To develop a next word prediction model in [17], the logs of data stored on client devices were used, where each device contains a cache of text representative of the user's typing behavior as captured by various social media and chatting apps. Data from consenting users of the Gboard app have been used after removing any personal information to protect the user privacy.

However, due to the inaccessibility to real user device data, this study uses the textual data from the publicly available Reddit dataset [5]. Reddit is an online social platform which hosts discussions across a wide variety of social and general events and issues happening in society. The data is structured into subreddits which represent hundreds of communities corresponding to different discussion topics under which users can create new posts or comment on the existing ones. The use of this dataset is intentional as it very closely relates to the real-world scenario of user conversations that occur on mobile devices while chatting. The dataset is accessible through the Google BigQuery API and is available in the form of posts or comments associated with particular users identified with the UserID. This provides an inherent partition based on userIDs, which can be used to mimic local data corresponding to a client in the federated setting. Since each device represents a different user, therefore the underlying data will be different based on the user's interaction with the device.

To show the feasibility of training LSTM on Raspberry Pi, this implementation uses the preprocessed Reddit dataset provided by LEAF in [10]. LEAF is a modular framework designed for federated learning simulations. Datasets module of this framework provides preprocessed data to users which can be directly applied to ML pipelines built on different frameworks like Tensorflow and Pytorch. It also provides enough details on how the data has been preprocessed to be used in the next word prediction task unlike the remaining literature where the steps involved in preprocessing have not been discussed in depth for them to be reproduced. The Reddit data in LEAF has been taken from pushshift.io [2] which includes posts and comments from December 2017. The comments dataset comprises of files where each comment is represented in the form of a JSON object. The most relevant keys and their respective description for each comment are shown in the Figure 3.8 below. Each comment in the data has a number of associated fields like comment ID, author of the comment, corresponding parent ID, created timestamp, to name a few. However, not all of these fields have been extracted while processing the data to make it ready to be fed into the model. For the task of next word prediction, for each user, the body of comments is used to create input sequences of text. Corresponding target variable contains the subreddit, created timestamp, score of the comment and the target output sequence.

Field	Description
id	The comment's identifier, e.g., "dbumnq8" (String).
author	The account name of the poster, e.g., "example_username" (String).
link_id	Identifier of the submission that this comment is in, e.g., "t3_5l954r" (String).
parent_id	Identifier of the parent of this comment, might be the identifier of the submission if it is top-level comment or the identifier of another comment, e.g., "t1_dbu5bpp" (String).
created_utc	UNIX timestamp that refers to the time of the submission's creation, e.g., 1483228803 (Integer).
subreddit	Name of the subreddit that the comment is posted. Note that it excludes the prefix /r/. E.g., 'AskReddit' (String).
subreddit_id	The identifier of the subreddit where the comment is posted, e.g., "t5_2qh1i" (String).
body	The comment's text, e.g., "This is an example comment" (String).
score	The score of the comment. The score is the number of upvotes minus the number of downvotes. Note that Reddit fuzzes the real score to prevent spam bots. E.g., 5 (Integer).
distinguished	Flag to determine whether the comment is distinguished by the moderators. "null" means not distinguished[4] (String).
edited	Flag indicating if the comment has been edited. Either the UNIX timestamp that the comment was edited at, or "false".
stickied	Flag indicating whether the submission is set as sticky in the subreddit, e.g., false (Boolean).
retrieved_on	UNIX timestamp that refers to the time that we crawled the comment, e.g., 1483228803 (Integer).
gilded	The number of times this comment received Reddit gold, e.g., 0 (Integer).
controversiality	Number that indicates whether the comment is controversial, e.g., 0 (Integer).
author_flair_css_class	The CSS class of the author's flair. This field is specific to subreddit (String).
author_flair_text	The text of the author's flair. This field is specific to subreddit (String).

Figure 3.8: Reddit Comments Data description, image from [2]

The dataset provided by LEAF contains a total of 56,587,343 comments from 1,660,820 users. However, for this dissertation, only a subsample of the dataset has been used. Since the objective here is to use two Raspberry Pi clients to train the model in a federated fashion, only data specific to two users having the highest number of training sequences were extracted using the script shown in Figure 3.9. A vocabulary size of 10000 tokens has been used while the user data corresponding to each Pi contain 1700 and 2200 tokens respectively.

The following steps are involved in cleaning the data to make it ready to feed into the model:

- The special symbols used in HTML are unescaped, which involve decoding the escaped strings.
- Extra spaces and non-ASCII symbols are also removed from the raw text.
- Any URLs present are replaced with a special token, the Reddit username for each user is replaced by a unique user token, and finally the sub-reddit name is also replaced by a sub-reddit token.
- The resulting text is converted to lowercase throughout.
- The cleaned text is then broken down into a list of tokens/words.
- For each comment converted into tokens after performing the above steps, sequences of 10 tokens are generated for the LSTM. Every input sequence begins with a "BOS"

```
#load reddit dataset
users, groups, train_data, test_data = read_data(train_data_dir, test_data_dir)

mu = [0, 0]
mu_sz = [-1, -1]
for u in users:
    sz = len(train_data[u]["x"])
    if sz >= mu_sz[0]:
        mu[1] = mu[0]
        mu_sz[1] = mu_sz[0]
        mu[0] = u
        mu_sz[0] = sz

    elif sz >= mu_sz[1]:
        mu[1] = u
        mu_sz[1] = sz

# for u in users:
for  i in range(2):
    u = mu[i]
    u_train = train_data[u]
    u_test = test_data[u]
    folder = os.path.join('new_correctdata', 'user_'+ str(u))
    if os.path.exists(folder):
        os.remove(folder)
    os.makedirs(folder)
    train_file_name = 'train_{}.json'.format(u)
    test_file_name = 'test_{}.json'.format(u)
    train_file = os.path.join(folder, train_file_name)
    test_file = os.path.join(folder, test_file_name)

    file1 = open(train_file, 'w')
    json.dump(u_train, file1)
    file1.close()
    file2 = open(test_file, 'w')
    json.dump(u_test, file2)
    file2.close()
```

Figure 3.9: Python script used to extract data for two Raspberry Pi clients

token which refers to beginning of sequence and ends with "EOS" token which refers to the end of sequence. Corresponding target sequences are of the same length except shifted one word to the right. From this target sequence, the last token is extracted to be the output label as it represents the next word for the given input sequence.

- Since the length of each comment is not going to be the same, padding is added to the sequences where there are less than 10 tokens to keep all the sequences of equal length before feeding the input into the neural network.

For a given comment, the input sequence of tokens and the corresponding target sequence in the form of JSON object can be seen in Figure 3.10 below.

```
{
    "x": [
        [
            [
                "<BOS>",
                "thanks",
                ",",
                "i",
                "needed",
                "a",
                "pick",
                "me",
                "up",
                "."
            ],
            [
                "much",
                "appreciated",
                ".",
                "<EOS>",
                "<PAD>",
                "<PAD>",
                "<PAD>",
                "<PAD>",
                "<PAD>",
                "<PAD>"
            ]
        ],
    "y": [
        {
            "subreddit": "DDLC",
            "created_utc": 1512087820,
            "score": 2,
            "target_tokens": [
                [
                    "thanks",
                    ",",
                    "i",
                    "needed",
                    "a",
                    "pick",
                    "me",
                    "up",
                    ".",
                    "much"
                ],
                [
                    "appreciated",
                    ".",
                    "<EOS>",
                    "<PAD>",
                    "<PAD>",
                    "<PAD>",
                    "<PAD>",
                    "<PAD>",
                    "<PAD>",
                    "<PAD>"
                ]
            ],
```


Figure 3.10: The structure of the input and corresponding target sequence in JSON Format

3.4 Model development

The LSTM model is used for the task of next word prediction as described in Section 3.1.1. The model has a total of 1,524,240 trainable parameters. The architectural design of the model developed is described below:

- The first layer in the model is the embedding layer. It is used to create a vector representation for each word in the vocabulary. Therefore, it is necessary to pass the size of the vocabulary as a parameter into the embedding layer. It is also required to specify the dimension, i.e., the size of the embedding vector space such that the model maps each token(word) to the embedding vector of that dimension. The model specifies a dimension of 16 as the data available on each client is small. Increasing the dimension of the embedding vector can help in capturing the relationship between similar words, but it would need more data to learn that relationship.

- The embedding vector is then passed through a single hidden layer of LSTM with 128 units.
- The output embedding returned from the LSTM layer is passed through a fully connected dense layer with the number of units equal to the size of the vocabulary. Softmax activation function is used to convert the output vector into a vector of probabilities normalized over the vocabulary.

The loss function of the LSTM model is computed as the cross entropy between the predicted target labels and the actual output labels. Detailed summary of the model is shown in Figure A.1 (Appendix).

Hardware Specifications

The experiments are conducted using Windows 10 laptop powered by Intel(R) Core(TM) i5-1035G1 CPU@1.00GHz and 64-bit processor which is used as the server. Two Raspberry Pi 4 Model B devices with 4 GB RAM and 16 GB external SD card are used as clients for training the next word prediction model using federated learning. The Raspberry Pi has been preinstalled with Raspbian Operating system, which is a version of the linux-based Debian operating system. Both the Raspberry Pi devices use wireless connections and were configured over the WiFi network.

3.5 Integration with FLOWER Framework

The training of the model begins with the random initialization of the model by the server. This randomly initialised model is shared with both clients to start the federated training. The Keras LSTM model is specified both in the server.py and client.py as shown in Figure 3.11 below. The model is compiled using the Adam optimizer which uses the stochastic gradient descent (SGD) method. This method helps in optimizing the error gradient for the current model state computed using the training data and updates the model weights using backpropagation. The amount at which the weights are updated is specified by the learning rate. It is currently set to 0.001 which is the default configuration of Keras Adam optimizer class. The loss function which needs to be minimized is specified as categorical cross entropy. This function computes the cross entropy loss between the actual target sequences and the output of the model.

```python
def main() -> None:
    # Load and compile model for

    model = keras.Sequential(...)
    # Add an Embedding layer expecting input vocab of size 10000, and
    # output embedding dimension of size 16.
    model.add(layers.Embedding(input_dim=10000, output_dim=16))

    # Add a LSTM layer with 128 internal units.
    model.add(layers.LSTM(128))

    # Add a Dense layer with units equal to vocab size.
    model.add(layers.Dense(units=10000, activation='softmax'))
    print(model.summary(...))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 3.11: Keras LSTM model used in implementation

The strategy is configured to use the FederatedAveraging (FedAvg) algorithm and takes in two clients for the parameters: *min_fit_clients* and *min_eval_clients* which initialises the minimum number of clients required during training and evaluation. These values can be changed when the system is scaled to use more clients. The parameter *min_available_clients* is also set to 2 as the current federated system is configured to use only 2 clients. The fraction of clients to be used in training and evaluation is specified using the *fraction_fit* and *fraction_eval* parameters, both are set to 1 to use both clients. The code snippet [4] of the strategy configuration in the flower server file is shown in figure 3.12.

```python
# Create strategy
strategy = fl.server.strategy.FedAvg(
    fraction_fit=1,
    fraction_eval=1,
    min_fit_clients=2,
    min_eval_clients=2,
    min_available_clients=2,
    on_fit_config_fn=fit_config,
    on_evaluate_config_fn=evaluate_config,
    initial_parameters=model.get_weights(),
)
```

Figure 3.12: Code for Configuration of the Federated Averaging strategy in Flower server.

The connection to both the Flower Raspberry Pi clients is established via SSH using

MobaXterm tool. A small subset of data for two users is extracted from Reddit dataset and saved on each of the Raspberry Pi Clients. This data is already split into two separate train and test JSON files. The setup of the virtual environment on each of the Raspberry Pi clients is done using Poetry which is a package management tool in Python. The pyproject.toml file contains all the packages and their versions that need to be installed to run the client code on the Raspberry Pi. This file is similar to the requirements.txt file which is generally used for running Python projects. Figure 3.13 shows the MobaXterm GUI used to connect to one of the Raspberry Pi named raspfed1.

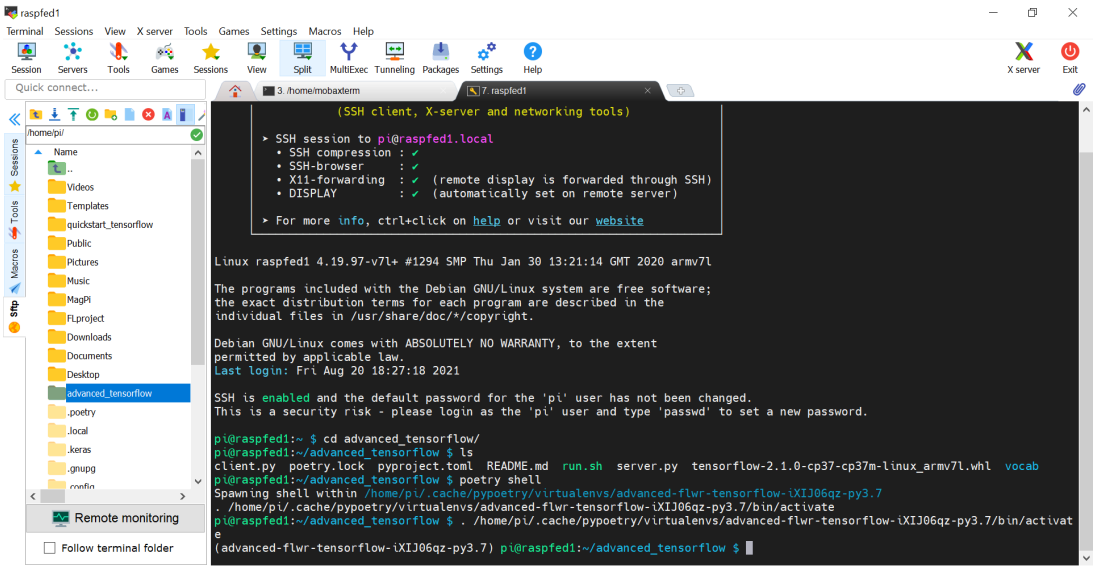

Figure 3.13: MobaXterm GUI to connect with Raspberry Pi Clients

The server.py is executed on the local machine which serves as the server while the client.py is run on both the Raspberry Pi clients connected via SSH as shown above. The server waits until both clients are connected and once the connection is established, the federated training process is started. The number of training rounds is specified by the server. Currently, the system is set up to run 50 rounds of training with 10 epochs in each round for both clients.

3.6 Challenges encountered

The following challenges were encountered as part of the design and implementation of this dissertation:

- Implementing next-word prediction model on an embedded device such as a Raspberry Pi using federated learning has been a challenging task owing to the novelty of the idea as the same has only been achieved on Android devices.

- Federated learning has been around since 2016, the research field is relatively new, and the available frameworks and technologies supporting FL are under ongoing development, which made it extremely difficult to develop a working prototype for a real-world FL scenario. The available code base is continuously evolving with many issues still unresolved.

- For next-word prediction task on a mobile keyboard, it requires learning the most frequently typed words by the user. Due to the privacy constraints, there was no access to real data. Initially, the dataset from the popular question and answer website, Stackoverflow, was explored but due to the lack of enough data preprocessing information required for using it for LSTM model led to the change in the dataset.

- The Reddit dataset was sourced from LEAF [10]. It included comments from thousands of users and getting data corresponding to only two users involved processing large files of size 1.6GB, which resulted in out-of-memory errors on the local machine. This was resolved by using lab machines which had 16GB RAM.

- Choosing a framework for federated learning was also a bottleneck as most of the popular frameworks like Tensorflow Federated (TFF), PySyft and LEAF did not support the implementation of federated learning on real devices but only provided simulations.

- Using the Flower framework and setting it up on the Raspberry Pi was the most difficult task as it required installing a lot of dependencies and packages involving debugging compatibility issues and finding the right versions of different packages to work with. After consistent troubleshooting, Python dependency management tool, Poetry was installed to handle version mismatches, which led to the successful setup of the Flower framework example code in Tensorflow.

- Finally, since there is no working example of implementing an LSTM in the flower framework, integrating the model in the framework to produce a working solution was a challenging task.

3.7 Summary

This chapter discussed the design and implementation details of the next word prediction LSTM model and training the same on resource-constrained Raspberry Pi devices acting as federated clients. This involved data gathering from the LEAF Reddit dataset and preparing it to mimic the local data cache of the Raspberry Pi clients. Next, it explained

the LSTM model design followed by its integration into the flower framework and summarising the challenges faced during the process. The next chapter will discuss the results and evaluate the hardware performance metrics on the Raspberry Pi devices.

Chapter 4

Evaluation

This chapter discusses the analysis of the federated training process and presents the results of the model implemented in the previous chapter. It also measures the performance metrics on the Raspberry Pi devices.

4.1 Experimental Setup

As described in the previous chapter, the experimental setup for the next word prediction task in a federated setting comprises the laptop as the server and two Raspberry Pi devices as the federated clients. The two Raspberry Pi client devices are named raspfed1 and raspfed2. The data of two users extracted from Reddit dataset described in Section 3.3 of the previous chapter, is stored on each of the two Raspberry Pi devices. It is important to note that due to the practical limitations of storage and computational capacity on the Raspberry Pi, only a subset of data is used, divided into train and test samples as shown in Table 4.1. The data in raspfed1 consist of 1485 training samples and 576 test samples. Similarly, the data in raspfed2 consist of 1097 train samples and 292 test samples.

Federated Clients	**Train Data**	**Test Data**
raspfed1	1485	576
raspfed2	1097	292

Table 4.1: The composition of local data on Raspberry Pi clients

4.2 Evaluation Results of the Federated Training

The federated training process involved training an LSTM model for next word prediction on Raspberry Pi devices. Both the Raspberry Pi clients participated in the training and evaluation round. The Reddit data was partitioned into clients in non-IID fashion. The data was non-IID as the size of the data was different and it was unique in the sense that it belonged to different users. The LSTM model comprised a single layer of 128 units followed by a fully connected dense layer with units equal to the size of the vocabulary. The federated training process starts with the random initialization of the model weights by the server which are sent to both clients.

4.2.1 Understanding the loss function

The loss function used in the Keras LSTM model is the categorical cross entropy. For each input sequence of 10 tokens, the output is a single token which is the next predicted word. The next word predicted by the model is a single vector having a size equal to the vocabulary size and has not been normalised. The softmax activation ensures that the raw predictions are converted to probabilities normalised over the vocabulary. The model is compiled using categorical cross entropy loss as this problem becomes a multiclass classification problem where the number of classes are equal to the number of tokens in the vocabulary of the training corpus.

The cross entropy (CE) loss is calculated as the difference between the output probabilities predicted by the model and the actual output (ground truth). The objective is to adjust the model weights during the training process such that the cross entropy loss is minimum.

4.2.2 Federated Evaluation at the Clients

The model is evaluated based on the loss and accuracy metrics. Client side evaluation is performed and is controlled by the server. The server configures the model hyperparameters: local batch size and the number of training epochs at the clients. At each client, the training data is divided into train and validation splits in the ratio of 9:1. The cross entropy loss and accuracy were calculated during the training on both train and validation data and illustrated in Figure 4.1. From the plot, it can be seen that for the first client, raspfed1, the initial loss reported on the training set is 7.8726 in the first epoch, which gradually declines to 3.9518 by the 10th epoch of the first round of federated training. However, the loss on the validation set is still higher with a value of 5.1543 by the end of the first round of federated training. Similar observations were reported by the second

client, raspfed2, where the loss on train data falls from 8.5485 to 4.1182 by the end of the 10th epoch in the first round of training. However, the final loss on validation data is 5.8556. As the training continues, the model starts to overfit the training data on both clients. This behaviour can be attributed to the small amount of training data available at the clients. The results obtained by the federated model are not as promising as the ones presented in the State-of-the-art FL simulations in [12][17] as the local data is limited to a few hundred training samples as well as the number of clients are limited to 2.

.

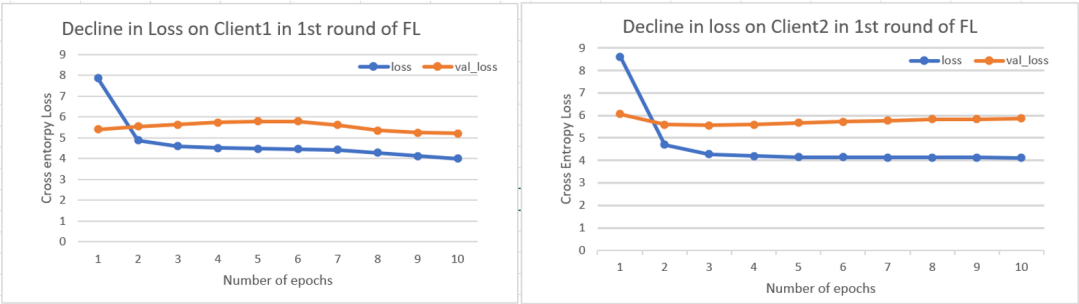

Figure 4.1: Plots showing the decline in CE loss during the first round of federated training on both clients.

At the end of each round of federated training, the local model at each client gets updated with the new global model weights. The loss and accuracy of this global model is then evaluated by both clients on their local test data and the results are published. The final evaluation results obtained on both clients at the end of the 50 rounds of training are CE loss and accuracy of (3.9669, 23.12%) and (6.1696, 39.37%) respectively. High value of CE loss on test data clearly points towards overfitting in the LSTM model. Reducing the complexity of the model can help in resolving the issue of overfitting. However, since the model comprises a single hidden layer of LSTM with 128 units only, increasing the training data would be a better option.

Varying the number of local epochs per round of federated training, test accuracy of the model does not improve for either of the two clients. However, it leads to increased training time of the whole system which in turn results in higher utilization of the computational resources on the Raspberry Pi devices. The results of choosing multiple number of epochs per round of training for 50 rounds are shown in Table 4.2.

It is evident that to achieve better results from the federated training of next word prediction LSTM model, there is a need to increase the number of participating clients. The experiments performed above clearly demonstrate the feasibility of training a next word prediction language model on real devices like the Raspberry Pi. However, to commercially exploit the capabilities of FL as done by the mobile keyboard application,

No. of epochs	Training Time (min)	Test Accuracy	
		Client1	Client2
1	15.59	24.37%	40.62%
5	35.29	23.74%	38.74%
10	55.93	23.12%	39.37%

Table 4.2: Test accuracy and training time reported on varying the number of local epochs on Raspberry Pi clients.

Gboard, there is a need to scale this prototype by increasing the number of Raspberry Pi clients to 50 or 100.

4.3 Measurement of Performance Metrics on Raspberry Pi

This section shows an evaluation of the measurements of time taken, memory consumed, and temperature recorded while running FL on the Raspberry Pi device as shown in [15] [20].

- Memory usage was measured by running free -h command on Raspberry Pi devices during the training process. Table 4.3 shows the memory used in FL training process on each of the client devices. Used memory also includes the memory occupied by the Raspbian operating system. It can be clearly observed that it is possible to train an LSTM with a single hidden layer of 128 units on a Raspberry Pi. However, increasing the complexity of the LSTM model further would mean training more parameters on a larger dataset. This would result in higher memory utilization.

Clients	Memory		
	Used(with OS)	% Utilization	Total
raspfed1	876MB	21%	3.8GB
raspfed2	658MB	17%	3.8GB

Table 4.3: Memory consumption on Raspberry Pi clients during federated training.

- The temperature of the Raspberry Pi devices was monitored during the training using the command: vcgencmd measure_temp. Since performing FL on device involves training LSTM model, it significantly increases the temperature of the device as shown in Table 4.4. The acceptable limit of temperature for the device to func-

tion properly as recommended by the Raspberry Pi Foundation is 85°C. Beyond this range, the device needs to be cooled down using an external cooling fan.

Clients	Temperature	
	Idle	Training
raspfed1	48°C	64°C
raspfed2	45°C	63°C

Table 4.4: Temperature recorded on the Raspberry Pi clients during federated training versus when they are left idle.

- The training time was recorded by the flower framework at the time of completion of FL on the Raspberry Pi devices. As the number of local epochs were increased, the training time increased significantly without much improvement in the accuracy of the model. It took 15.59 minutes to run 50 rounds of training with the number of local epochs set to 1. On increasing the number of epochs to 10, the training time increased to 55.93 minutes.

4.4 Summary

The LSTM model used for prediction of the next word given a sequence of 10 tokens has been implemented and trained on the resource constrained device, Raspberry Pi. Two Raspberry Pi devices are used as federated clients in a real deployment scenario. The final global model obtained after the last round of training resulted in a prediction accuracy of 23% and 39% on test data at client1 and client2, respectively. The performance metrics captured while executing training on the Raspberry Pi were analyzed. It included the training time, memory usage, and temperature measurement on the devices. The experimental results show the practical feasibility of implementing a language model on resource constrained devices in a federated setting. However, to achieve viable accuracy, the system needs to be scaled to include more than two devices, with each device having access to more training samples compared to the local data currently available.

Chapter 5

Conclusions & Future Work

5.1 Conclusion

This dissertation assessed the feasibility of implementing a next word prediction LSTM model on resource constrained devices such as a Raspberry Pi. It was inspired from similar work done on Android devices [17]. Addressing the initial research objectives, it clearly demonstrated the practicality of the novel idea of implementing next word prediction on a Raspberry Pi. The LSTM model implemented and trained on the device contains a single hidden layer of 128 units, which is well within the resource bounds of the Raspberry Pi Model B device with 4 GB RAM. However, the Raspberry Pi devices need to be connected to a constant source of power supply until the training is completed. The use of the flower framework helped in implementing this novel solution in a practical setting using real Raspberry Pi devices compared to other frameworks which only supported FL simulations. Lastly, it can be concluded that it is computationally possible to train language models on devices like Raspberry Pi using federated learning. However, to exploit the technology in a commercial setting, the current prototype needs to be scaled to use more clients required to achieve the desired accuracy of the LSTM models.

5.2 Limitations and Future Work

The limitation of this work, as highlighted by the experiments conducted, is the usability of the trained model to predict the next word given the low accuracy achieved. This can be attributed to the current implementation solution which deploys federated learning on only two Raspberry Pi devices. The next most exciting step to build upon this dissertation would be to deploy this solution on a testbed of 50 to 100 Raspberry Pi devices to relate more closely to real-world situations and uncover possible issues. Moreover, this

dissertation needed data which could be used to mimic the user-typed data on mobile keyboards. Due to privacy concerns associated with sharing personal data by the users, the data for two users with the maximum size of training samples was extracted from the Reddit dataset. However, even that data was limited to just a few hundred training samples available per user. It would be worth exploring another dataset which can provide larger text data per user, where each user's data will correspond to local data stored on a particular Raspberry Pi device. Getting real user device data as used in [17] can lead to a more accurate model which maximizes the real-world usability of this solution.

Once there are more client devices available to train on, it would be interesting to optimize the model by stacking more hidden layers of LSTM. Having more devices with sufficient data will be required to train the model parameters for a complex LSTM neural network. This could lead to better predictions and reduced cross entropy loss. Fine tuning hyperparameters like local batch size, number of epochs, and fraction of the client population used in training and validation steps are also possible directions which can be explored to improve the performance of the overall system.

Bibliography

[1] Rami Al-Rfou, Marc Pickett, Javier Snaider, Yun-hsuan Sung, Brian Strope, and Ray Kurzweil. Conversational contextual cues: The case of personalization and history for response ranking. *arXiv preprint arXiv:1606.00372*, 2016.

[2] Jason Baumgartner, Savvas Zannettou, Brian Keegan, Megan Squire, and Jeremy Blackburn. The pushshift reddit dataset. In *Proceedings of the international AAAI conference on web and social media*, volume 14, pages 830–839, 2020.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The journal of machine learning research*, 3:1137–1155, 2003.

[4] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[5] Google BigQuery. Reddit comments dataset. `https://bigquery.cloud.google.com/dataset/fh-bigquery`, 2016.

[6] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 441–459, 2017.

[7] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[8] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. page arXiv:1602.05629, 2016.

[9] J Dean Brock, Rebecca F Bruce, and Marietta E Cameron. Changing the world with a raspberry pi. *Journal of Computing Sciences in Colleges*, 29(2):151–153, 2013.

[10] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[11] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 267–284, 2019.

[12] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635*, 2019.

[13] Valerie Chen, Valerio Pastro, and Mariana Raykova. Secure computation for machine learning with spdz. *arXiv preprint arXiv:1901.00329*, 2019.

[14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[15] Yansong Gao, Minki Kim, Sharif Abuadbba, Yeonjae Kim, Chandra Thapa, Kyuyeon Kim, Seyit A Camtepe, Hyoungshick Kim, and Surya Nepal. End-to-end evaluation of federated learning and split learning for internet of things. *arXiv preprint arXiv:2003.13376*, 2020.

[16] Slawomir Goryczka and Li Xiong. A comprehensive comparison of multiparty secure additions with differential privacy. *IEEE transactions on dependable and secure computing*, 14(5):463–477, 2015.

[17] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Ahmed Imteaj, Urmish Thakker, Shiqiang Wang, Jian Li, and M Hadi Amini. Federated learning for resource-constrained iot devices: Panoramas and state-of-the-art. *arXiv preprint arXiv:2002.10610*, 2020.

[20] Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. *arXiv preprint arXiv:1909.12326*, 2019.

[21] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, and Rachel Cummings. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[22] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[23] Jakub Konečný, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[24] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[25] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[26] Akhil Mathur, Daniel J Beutel, Pedro Porto Buarque de Gusmão, Javier Fernandez-Marques, Taner Topal, Xinchi Qiu, Titouan Parcollet, Yan Gao, and Nicholas D Lane. On-device federated learning with flower. *arXiv preprint arXiv:2104.03042*, 2021.

[27] H Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data. `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`, 2017.

[28] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*, 2017.

[29] Christopher Olah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015.

[30] Tom Ouyang, David Rybach, Françoise Beaufays, and Michael Riley. Mobile keyboard input decoding with finite-state transducers. *arXiv preprint arXiv:1704.03987*, 2017.

[31] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *European conference on computer vision*, pages 213–226. Springer, 2010.

[32] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[33] Raphael Tang and Jimmy Lin. Adaptive pruning of neural language models for mobile devices. *arXiv preprint arXiv:1809.10282*, 2018.

[34] Tensorflow. Tensorflow federated: Machine learning on decentralized data. `https://www.tensorflow.org/federated`, 2015.

[35] Kangkang Wang, Rajiv Mathews, Chloé Kiddon, Hubert Eichner, Françoise Beaufays, and Daniel Ramage. Federated evaluation of on-device personalization. *arXiv preprint arXiv:1910.10252*, 2019.

[36] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[37] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[38] Seunghak Yu, Nilesh Kulkarni, Haejun Lee, and Jihie Kim. On-device neural language model based word prediction. In *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, pages 128–131, 2018.

[39] Jiawei Yuan and Shucheng Yu. Privacy preserving back-propagation neural network learning made practical with cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):212–221, 2013.

Appendix A

Model specifications

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 16)          160000
_________________________________________________________________
lstm (LSTM)                  (None, 128)               74240
_________________________________________________________________
dense (Dense)                (None, 10000)             1290000
=================================================================
Total params: 1,524,240
Trainable params: 1,524,240
Non-trainable params: 0
```

Figure A.1: Model Summary

...