UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL DE CALCULATOARE

PROIECT DE DIPLOMĂ

Învățare Federată în Scenarii de Mobilitate Umană la Exterior

Vlad-Alexandru Proteasa

Coordonator științific:

Conf. dr. ing. Radu-Ioan Ciobanu

BUCUREȘTI
2022

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

DIPLOMA PROJECT

Federated Learning in Human Mobility Scenarios

Vlad-Alexandru Proteasa

Thesis advisor:

Conf. dr. ing. Radu-Ioan Ciobanu

BUCHAREST
2022

CONTENTS

SINOPSIS

Învățarea federată este o variație a învățării automate care antrenează algoritmul pe mai multe dispozitive descentralizate sau servere prin colectarea în mod exclusiv a parametrilor necesari actualizării modelului și nu prin colectarea efectivă a datelor. Această abordare crește nivelul de confidențialitate și securitate deoarece consumatorii nu sunt nevoiți să împartă datele datele cu nimeni altcineva și toată procesarea poate să fie efectuată pe dispozitivul lor mobil. În acesta cercetare o să analizăm cum mai multe tipuri de rețele neuronale pot fi folosite pentru a prezice mobilitatea umană folosind algoritmi de învățare automată convenționali cum ar fi RNN, LSTM sau GRU, după care vom extinde această implementare pentru a include învățarea federată. Rezultatele pentru ambele abordări vor fi evaluate pentru a confirma implementarea, după care vom determina cum a fost impactată acuratețea rețelelor și daca este un compromis justificat pentru a spori nivelul de securitate și confidențialitate a datelor.

ABSTRACT

Federated learning is a machine learning technique that trains an algorithm across numerous decentralized edge devices or servers by gathering just the parameters required to update the model and not the actual data. This increases data privacy and security since consumers do not need to share their personal data with anyone and all necessary processing may be done on their edge device. In this study, we will look at how various neural networks may be used to predict human mobility using convectional machine learning algorithms like RNN, LSTM, or GRU, and then we will expand the implementation to include a federated learning framework. The results of both approaches will be reviewed in order to confirm the implementation and then to determine how the accuracy of the networks was altered and whether it is a justifiable trade off for the increased benefits of data security and data privacy.

1 INTRODUCTION

1.1 Context

Federated learning is a machine learning approach that trains an algorithm across several decentralized edge devices or servers. This strategy differs from standard centralized machine learning techniques, in which all local data-sets are uploaded to a single server, as well as more traditional decentralized alternatives, which frequently presume that local data samples are dispersed uniformly.

1.2 Problem

The main issue that federated learning projects face is obtaining comparable results with traditional machine learning techniques. This is due to the fact that the developers or data scientists that try to implement such a model do not have as much control over the data. So far there are multiple papers that explored how to implement machine learning models for next location predictions using conventional machine learning. There are also papers that explore how to adapt such models using federated learning frameworks but, from our research we did not find one that integrates federated learning into this specific scenario.

1.3 Objective

The main purpose of this project is to attempt to implement a valid federated learning model that is able to make next location predictions using a federated learning framework that simulates a real life environment in which data is randomly collected and fed into the model. This project does not aim to optimize data collection policies neither does it hope to create a better solution for traditional machine learning next-location predictions. The main focus is to obtain comparable results using a traditional models and than analyze how they perform in federated learning environments and how they could be optimized to reach a result closer to the standard method.

1.4 Proposed Solution

The approach for this task was to implement traditional models and than try to adapt them as much as possible to fit the federated learning methodology. From the research that was performed the best approach for next position predictions seems to be using supervised learning methods. The models that best fit this problem seem to be: Recurrent neural network, Long short-term memory and Gated recurrent unit. Another improvement that had good results in our testing on the selected data-set was to split the check-ins of the users using the K-means unsupervised model and train a model for each cluster.

After completing the implementation of the classic machine learning models and obtaining an acceptable error rate the main challenge will be to re-adapt the solution to fit the the chosen federated learning framework. Currently there are three main frameworks available: TensorFlow Federated Learning, PySyft and Flower.

1.5 Results

The research results demonstrate that there is a minor decline in the overall accuracy of the selected networks while migrating to federated learning. This was to be expected, and we were glad to see that the overall impact was not as severe as it could have been due to model over-fitting. The overall accuracy falls between acceptable levels, indicating that this shift may be made in order to design a safer machine learning system that prioritizes privacy and security when it comes to consumer's personal data.

1.6 Structure of the Paper

The second part, Motivation, will provide a high-level summary of what we are attempting to accomplish in this project with the resources we have available and the contexts in which this project may be utilised. The third section, State of the Art, will explain some of the work that has been done thus far, technologies available, as well as reasons why they should be considered, their drawbacks, and how these tools compare to each other. This section will also go through the data set briefly and also provide information about the data that we will be working with.

The fourth part, Proposed Solution, will show the conceptual implementation and workflows that we want to implement using the tools described. In the fifth part, Implementation, we will go through each component of the previously outlined project and how we implemented them.

The sixth part, Evaluation, will compare the two methodologies developed (machine learning and federated learning) with other proposed solutions in order to determine whether the assumptions and implementations are accurate. The final section, Conclusion, will address whether or not we obtained satisfying outcomes and how they may be improved or expanded in the future.

2 MOTIVATION

At the time of starting this project a big issue that the whole planet was facing was the global pandemic caused by the spread of the COVID-19 virus. This caused many countries around the world to introduce restrictions that imposed limits on the daily movements of each individual. People also started to take notice of the danger at hand and even before the restrictions started to avoid crowded areas and many places that increased the risk of coming into contact with someone infected. This presented a great opportunity to study how machine learning models can be used predict human mobility and how they could help in this particular situation [17].

One aspect that was concerning regarding this project was how would this scale long term and what would be the incentive that would determine people to join in and contribute with their data if this model made it to production as part of another application that made use of this next location prediction model. This is a very sensitive subjects as the data needed is composed of the actual GPS location of the user and the exact timestamp of the check-in. As people are more and more concerned with what happens with their personal data, how is it getting stored and what companies are using that data for they are getting reluctant in sharing personal information and trying to limit what information is getting shared.

The best solution at the moment is to integrate a federated learning model. This way the users get assured that their data does not get shared with anyone because it never leaves their device. The data from the device gets processed locally on the edge device and what goes on the main server are only processed values, the weights needed to update the global model.

From the research done before starting this project there are multiple works that successfully implemented ways of predicting human mobility with conventional machine learning models [15] [20] and slightly altered versions that introduce new metrics such as social connections or personality traits [3] [9]. As data privacy becomes more and more important it seems as a good opportunity to study how or if federated learning is the best way of solving this problem.

In terms of what we may expect to achieve in terms of predictions, we have three key alternatives for moving forward with this project: next-location prediction, crowd flow prediction, and trajectory creation [11]. Crowd flow prediction does not appear to match the attitude of constructing a federated learning model since it negates the objective of not needing the user to give their data and providing a forecast of where other people would be. In the long term, trajectory creation does not appear to be very beneficial. People currently utilize programs that predetermine their route to their destination in such a manner that their time spent on the road is optimized. Creating a trajectory generating model seems pointless because it would only be useful in determining whether individuals will really take the suggested path.

Finally, next-location prediction appears to have the most applications in real-life settings for the average user in our opinion. Many applications, in our opinion, might successfully use this as a means of improving the user experience.

One such program is "Moovit," which we routinely use. This program creates a route to the given place using solely public transit. A next-location prediction algorithm might be used in this case to better propose places by guessing where the individual could be headed in the next couple of hours.

Another example would be travel-related programs such as "TripAdvisor," which assess and recommend tourist attractions. The recommendations may be enhanced if the app knew where the users were heading because they would be more likely to try things in their path rather than taking a detour. They may find these recommendations more valuable and use the application more frequently, generating more data to improve the model and increase traffic to the platform.

Even services like "Google Maps" might benefit from this by improving how map data is loaded. The program may be able to download information more effectively when connected to a WI-FI network, which may subsequently be utilized for offline navigation. This might result in a more seamless user experience since the user would not have to remember to download the map in advance because the service would do so automatically. Considering roaming data costs and personal irritation, such a setup might result in a better overall user experience.

3 STATE OF THE ART

One of the first stages in preparation for this project was to analyze prior work, especially what models were used, how they were altered to match the criteria, and how they may fit in our solution. In this section, we will examine different models and determine their benefits and downsides.

3.1 Model Selection

The two main categories of machine learning [5] models available are supervised and unsupervised learning models.

Supervised learning [5] involves learning a function to map an input to an output based on example input-output pairs. For our use case, the input would be a representation of someone's location at a point in time and the output would try to predict the next location.

Unsupervised learning [12] is used to draw inferences and find patterns from input data without references to labeled outcomes. Two main methods used in unsupervised learning include clustering and dimensionality reduction.

The most popular supervised learning models include Markov Chain, Recurrent Neural Network, Gated Recurrent Units, Long Short-Term Memory, and Random Forest.

Markov Chain is a stochastic model that is used to predict the outcome of an event given the previous state and its actions. The state is represented by the context of the subject. Markov chains are memory-less, they do not take into consideration prior actions nor the states in which the subject has been previously. Prediction depends solely on the current state and the action performed while in this state.

Because traditional neural networks treat each input and output pair as a distinct entity, they do not fit well on sequential data, such as a person's location check-ins throughout the day. This is where RNNs (Recurrent Neural Networks) [6] come in to try to improve. They attempt to forecast using an internal state (memory). When making a choice in this technique, the model considers the current input as well as what it has learnt from prior iterations. The calculation is made up of a sequence of processes, each of which produces an output that is fed into the following phase, resulting in a feedback loop. Based on the input-output relationship we can have multiple types of RNNs: one-to-one, one-to-many, many-to-one and many-to-many. Managing gradient changes, also known as the vanishing gradient problem [14] and the exploding gradient problem [18], is a typical challenge when developing a model like this. These problems are common for models based on back-propagation. Because of the chosen activation function and learning rate, the loss function approaches zero too early for the vanishing gradient, and the training process is greatly slowed or even halted. The loss function values for the exploding gradient are too large, and the network changes are too abrupt, therefore it will take a long time for the model to converge to an optimum state.

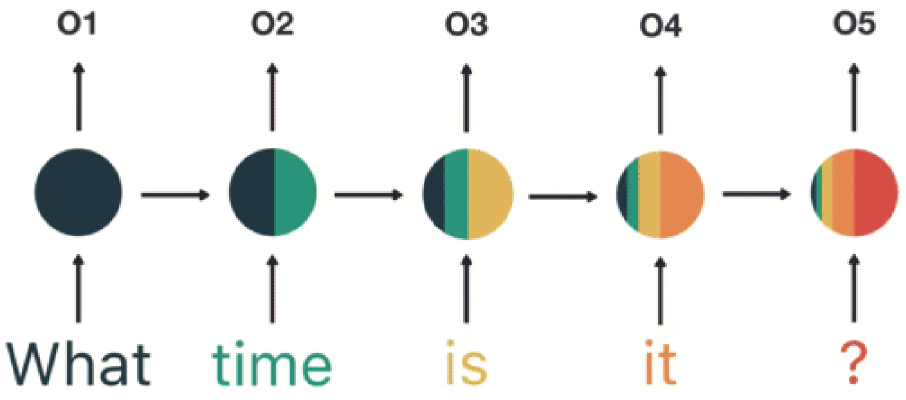

Figure 1: RNN hidden states[1]

The main downside of RNNs is that they can't handle large data samples well. The earlier inputs tend to have less of an impact on the final decision, while the later inputs have a much larger impact. We can use other models to overcome the vanishing/exploding gradient problem, such as GRU and LSTM.

Gated Recurrent Unit (GRU) [4] is an improved version of a traditional recurrent neural network that aims to solve the vanishing gradient problem [14]. The enhancement consists of including two "gates" into the implementation: an update gate and a reset gate. These so-called gates are made up of two vectors that define which information should be used to produce the final output. The "gates" work together to provide the model with more memory, which prevents previous model changes from being neglected in later phases and can help improve the model by discarding redundant data.

Long Short Term Memory (LSTM) [7] is similar to Gated Recurrent Unit (GRU) [4] in that it is a Recurrent Neural Network adaption that use "gates" to overcome the vanishing gradient problem. In LSTM, three gates are used: forget, input, and output.

- The Forget Gate is in charge of deciding what knowledge is vital and should be saved and what should be forgotten.
- The Input Gate is used to add data to neurons. It determines which values should be added to the cell using an activation function such as the sigmoid.
- The Output Gate is in charge of picking crucial information from the current cell and displaying it as output.

[1]https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

Random forest is a machine learning approach that is most commonly used for classification issues but may also be used for regression. The key element on which this technique is based is ensemble learning, which combines numerous classifiers to solve a difficult issue. As previously stated, the random forest model is made up of numerous classifiers known as decision trees. The training procedure is carried out by aggregating the findings and enhancing the "forest's" overall accuracy. The model's ultimate outcome or conclusion is generated by taking the average of all the trees. Increasing the number of trees is a frequent approach to improve accuracy, and unlike classic decision tree algorithms, this does not lead the model to overfit as quickly.

3.2 Data Privacy

There are many concerns in the public space regarding the personal data that is getting collected by big tech companies and what they are doing with it [10]. In recent years this got amplified by events such as Cambridge Analytica and the US presidential election or whistleblowers from companies like Facebook or state agencies from abroad. It is easy to be get scared by these events and try to block all data that is going out from your phone or laptop.

To present how easy it is to extract personal information we will use a photo as the example. By using multiple tools that are available to anyone we can extract the exact GPS coordinates from where the photo was taken, date and time, the device which was used to take the photo, if it was edited, what software was used to edit the photo and much more. Even the lack of information can tell us a lot about a photo, applications like WhatsApp wipe out certain metadata from EXIF files [2].

People are getting more and more cautious with what data they are sharing and they go out of their way to try and stop this data from getting in the wrong hands. Even companies such as Apple listened to these concerns and have given to users to block apps from monitoring them when they are not using the app.

This is a big issue for machine learning because without data we can not train the models to perform the tasks that we want them to. Using groups that opt-in to data collecting programs can impact the results and using already collected data could mean that the model will not update to the current context when changes occur.

Federated learning presents itself as a way of overcoming this issue. This methods does not need the data to be all in the same place on a database managed by the developers. In this approach the data never leaves the user's device and thus it has two main advantages to the traditional methods: the data never gets shared with anyone besides besides the one that generated it and because it never gets shared it eliminates the risk of someone intercepting the packages that contain the data when it is getting shared. This is achieved by running the model on the user's device and updating something called local model, after this step periodically the device shares with the main server the updated weights. When the server considers that it has collected enough information it aggregates the collected weights and updates the global model which is than shared with everyone.

Even though this is a great improvement in data privacy and security it is not fault free. Someone can still get hold of packages that are being shared and could in theory reverse engineer how the weights were generated and extract the user's data.

Another shortcoming of this method is that the developers are not aware of how the real data looks so many assumptions have to be made during development of the model. This can be improved by having some sample data collected from volunteers to have a starting point.

3.3 Frameworks

Currently there are many machine learning frameworks that make it easier to develop from scratch a fully fledged product without having to develop the model from scratch. These can be really useful in cutting down development time because we do not need to implement again well knows and documented models and functions that have already been available for a while, tested to cover all the possible bugs and optimized to run efficiently on CPUs and GPUs. While searching for which framework to use it looked like there were three main options available: TensorFlow, PyTorch and Scikit-learn.

Even thought Scikit-learn seems to provide a great set of functions and models to work with it does not seem like it provides enough customization. This is great for prototyping but it did not seem like it has all the tools required to finish this project. Possibly it could of been a good tool for the initial part when multiple models were being tested but when switching to federated learning these models would of had to be re-implemented in the new framework and tested again. It is up to debate whether or not it would of been the better approach.

Started by a group of developers from the Google Brain team and than released in 2015 to the public, TensorFlow [2] is an open source framework that enables faster development process for machine learning applications and research. This library is available and fully supported for both Python and C++ on which it exposes an API which has a vast array of implementations of mathematical function and deep learning models. Regarding the platforms that this library can run on we have all the we know and established operating systems like Linux, macOS and Windows and lately it also supports IoT device being able to run on Android Raspberry Pi.

The last framework that we want to mention is PyTorch, another open source project has much in common with TensorFlow. It is a reliable library well documented and with a large community to back it up. Just like TensorFlow it is available for Linux, macOS and Windows and can be used in projects based on Python, C++ or Java.

We will then offer a more in-depth comparison of PyTorch and TensorFlow, focusing on the

[2] `https://github.com/tensorflow/tensorflow`

differences between these two frameworks and why someone would choose one over the other when designing a machine learning project.

The first comparison will be in terms of performance, or the raw throughput of a model built using these frameworks. This is demonstrated by how quickly they can handle massive amounts of data. PyTorch offloads much of its work to the same versions of the cuDNN and cuBLAS libraries, which might explain the following results in Table 1. The results of the benchmark testing [16] show that PyTorch outperforms its equivalent in terms of overall performance. Throughput for the AlexNet, VGG-19, ResNet-50, and MobileNet models is measured in pictures per second, tokens per second for the GNMTv2 model, and samples per second for the NCF model.

Table 1: Throughput: higher is better [16]

Framework	**AlexNet**	**VGG-19**	**ResNet-50**	**MobileNet**	**GNMTv2**	**NCF**
TensorFlow	1422 ± 27	66 ± 2	200 ± 1	216 ± 15	$9631\pm1.3\%$	4.8e6$\pm2.9\%$
PyTorch	1547 ± 316	119 ± 1	212 ± 2	463 ± 17	$15512\pm4.8\%$	5.4e6$\pm3.4\%$

Training time and accuracy may be one of the most tempting parameters for any framework in the machine learning environment since we want to test it as quickly as possible while obtaining the most accurate results possible. Once again, PyTorch appears to be the clear victor in this area. As seen in Figure. 2 and Figure. 3 and discussed in [19], PyTorch outperforms TensorFlow by about 3.5156 seconds, or 31.4%. This is a significant improvement in terms of performance, especially when the accuracy of each model is considered. When we examine the accuracy for each implementation, we can observe that the average varies little from one to the next. In conclusion, both were capable of successfully implementing and testing the model, however PyTorch won due to the shorter learning time required.

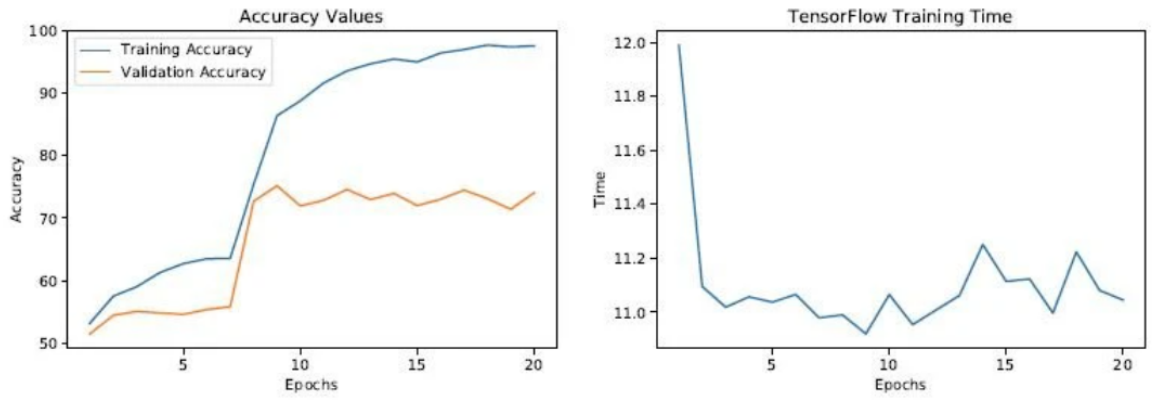

Figure 2: TensorFlow Accuracy and Training Time [19]

There have recently been several public debates on the environmental impact of computers. This was due to the growing popularity of blockchain-based technologies, particularly Bitcoin and Etherium [8]. These systems demand a lot of computing power, which means a lot of

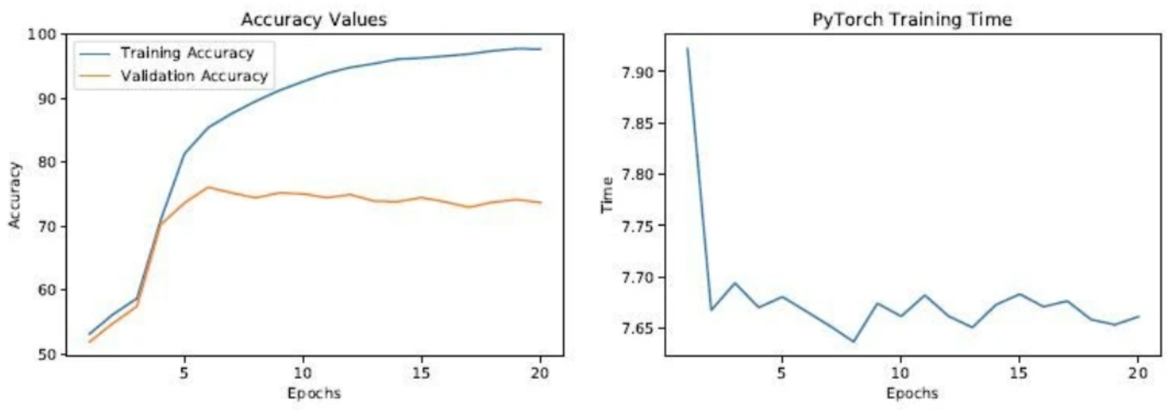

Figure 3: PyTorch Accuracy and Training Time [19]

energy, which is quite comparable to machine learning processing. Many businesses have migrated to cloud computing and are relying on cloud providers such as Amazon Web Services, Microsoft Azure, and Google Cloud to run their applications rather than building a dedicated data center that they must manage, which has many advantages such as cost savings, enhanced security backups, scalability, and many others, as detailed by Hukam Chand Saini in [1]. Steven Gonzalez Monserrate examined the influence of cloud systems on our environment in [13], focusing on carbon emissions, water cooling systems, noise pollution, and waste.

This also makes us wonder how efficient these frameworks are, how efficiently they utilise our computers not only for cost savings but also for environmental protection. According to Chance Simmons' and Mark A. Holliday's research [19], TensorFlow consumes less memory when operating, yet this is not the most important issue. The true metric is the calculation time (time necessary to train the model). Despite the fact that TensorFlow uses less memory, PyTorch needs us to operate our systems for shorter periods of time.

To conclude the topic of comparing the two frameworks, considering the better overall performance and use cases in which each tool is used, PyTorch appears to be the best approach considering the current context for the scope of this project in which the model will only be evaluated to see how well it would perform and is not meant to be used in a real world application just yet.

3.4 Data Set

3.4.1 Overview

There were many uncertainties while searching for a dataset, such as what to look for and how to evaluate the datasets that we discovered. Given that the goal of this project is to adapt a machine learning model to work on federated learning, one of the primary requirements was to select a well-established dataset that has previously been used successfully in machine learning projects relating to human mobility.

We eventually came upon a dataset compiled using Gowalla's API and made public by Stanford University that is part of the Stanford Network Analysis Platform (SNAP) Collection [3]. SNAP [4] is a general purpose, high performance system for analysis and manipulation of large networks. Graphs consists of nodes and directed/undirected/multiple edges between the graph nodes. Networks are graphs with data on nodes and/or edges of the network.

Gowalla was a location-based social networking website that was founded in 2007 but shut down in 2012. Members on this platform could share their check-ins via the smartphone application or the mobile website. This dataset has been successfully used in studies aimed at performing next-location predictions, which corresponds exactly to our end goal.

3.4.2 Statistics

The data-set is split into two parts:

- [*Social Graph*] A representation of the social connections between the users whose data has been collected. This consists of a total of 196,591 nodes and 950,327 edges, where each node represents a user and the existence of an edge the existence of a direct social relationship on the platform.
- [*Check-ins*] A total of 6,442,890 check-ins that were registered from Feb. 2009 to Oct 2010. Each entry consists of the id of the user, check-in time, latitude, longitude and location id.

Table 2: Sample data entries in Gowalla data-set for check-ins

[user id]	[check-in time]	[latitude]	[longitude]	[location id]
196514	2010-07-24T13:45:06Z	53.3648119	-2.2723465833	145064
196514	2010-07-24T13:44:58Z	53.360511233	-2.276369017	1275991
196514	2010-07-24T13:44:46Z	53.3653895945	-2.2754087046	376497
196514	2010-07-24T13:44:38Z	53.3663709833	-2.2700764333	98503

3.5 Technologies Used

In this part, we will detail the technologies that were utilized in the final implementation, including what they do, why they are popular in the machine learning community, and what their benefits and drawbacks are. This will not go into detail about how each of them was utilized or what characteristics made the project possible, but it will support our final

[3]`http://snap.stanford.edu/data/loc-Gowalla.html`

[4]`http://snap.stanford.edu/index.html`

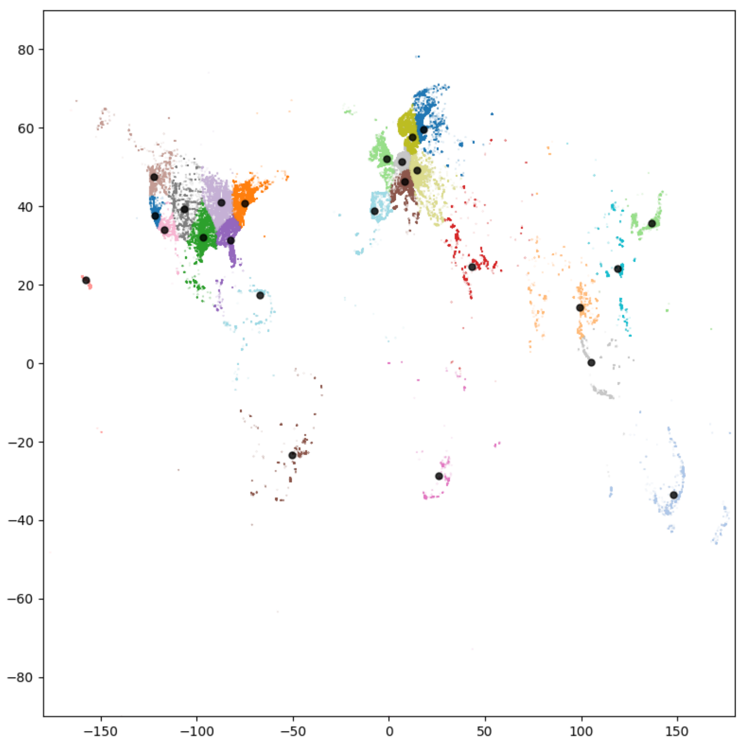

Figure 4: K-Means Clustering applied on the data set

selection and should give the reader a better idea of why someone could choose this stack of technologies.

3.5.1 Python

Python is a high-level, interpreted programming language that prioritizes readability and enables for fast prototyping and development. This language provides a lot of versatility at its core, as well as through utilizing the vast array of open source libraries that are accessible. When compared to other programming languages such as C/C++ or Rust, Python offers numerous features that offload the work that a developer must do: dynamic typing, garbage collector, portability, and embedding.

Python provides code that is concise and readable. While machine learning and AI rely on complicated algorithms and varied workflows, Python's simplicity allows developers to create robust systems. Instead of focusing on the technical subtleties of the language, developers may commit their entire attention to solving machine learning problems.

3.5.2 PyTorch

PyTorch [5] is one of the many machine learning frameworks accessible to Python programmers, whose core is written in C++, which is one of its key advantages. Because it does not impose additional overhead like other frameworks established in other programming languages, this framework is particularly efficient in processing massive amounts of data. As a result, Python is one of the most often utilized tools for constructing machine learning models.

Another advantage of this framework is how simple it is to understand and begin constructing models due to its numerous similarities to vanilla Python. It also provides a user-friendly runtime environment that makes debugging simple. The way models are implemented in the framework takes this a step further. PyTorch is built on "dynamic computational graphs," which means it does not consider neural networks as static objects and allows changes to be made while the application is running. This allows and encourages developers to experiment as they construct and test the models they are working on.

PyTorch was designed for parallelism and portability. This architecture allows programs to efficiently use the machine's resources to process massive amounts of data. This is accomplished by employing a single shared thread pool for inter-op parallelism, which is shared by all inference processes split inside the application process. PyTorch may use several threads within the ops in addition to inter-op parallelism (intra-op parallelism). This can be beneficial in a variety of situations, such as element-wise operations on big tensors, convolutions, GEMMs, embedding lookups, and so on. Furthermore, the developer may quickly modify the framework to leverage the machine's CPU cores or GPU to best suit the demands of a certain use case. This also implies that a GPU is not required by default, making it more accessible to a wider audience.

Despite the benefits discussed above, there are certain drawbacks that should be considered when determining how well this framework matches the solution and the overall purpose of the project. Even though we indicated that this framework allows for dynamic model adjustments in the runtime environment, which is wonderful for constructing models, it has a significant influence on where we may utilize it. This works well in development and research environments, but not in production, where you don't want anybody other than the program to change the model.

Another disadvantage of this framework is the lack of any built-in monitoring tools. Other third parties, such as data visualization libraries or external platforms like TensorBoard, should be utilized to monitor and visualize what is occurring inside the application.

Finally, this is not intended for the complete creation of machine learning tools for user applications. If somebody wants to take this a step further and create an actual application that uses the models built with this, they should move the model to another framework with deployment capabilities.

[5] `https://github.com/pytorch/pytorch`

Finally, this technology is incredibly configurable, has a strong community behind it, and fits nicely with our final aim of just exploring how well we can adapt a machine learning model to perform in a federated learning environment. We will just try to mimic such an environment and will not seek to undertake an actual deployment and test it in real life because this is outside the scope of the project.

3.5.3 Flower

Flower [6] is an open-source framework for constructing federated learning systems that allows for a seamless transition from traditional machine learning frameworks to federated learning. It is adaptable and enables for the quick creation of a simulated local training environment that can be quickly expanded and modified to real-life settings. Because it is language and framework agnostic, it is one of the most adaptable frameworks in the market. Any project may use this to shift to federated learning.

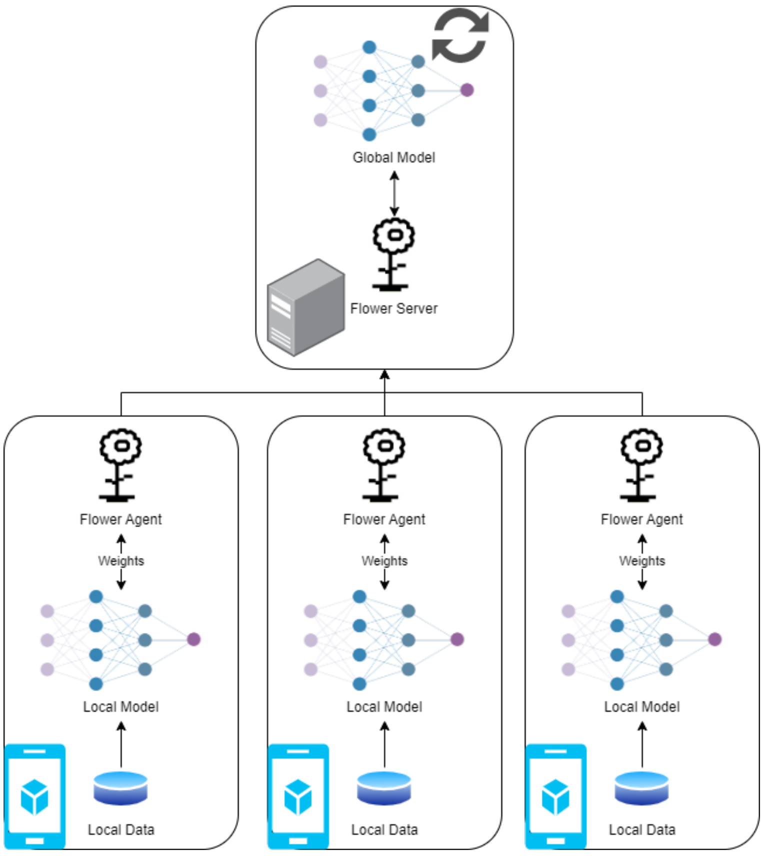

Figure 5: Flower workflow

Remote Procedure Call (RPC) is the foundation of the Flower architecture. A RPC occurs in distributed computing when a device, the client, instructs another device, the server, to run a procedure (function/subroutine) in another address space accessible over the network interface. The server functions as a library, exposing an API that the developer may use

[6]`https://flower.dev/`

without doing any additional tasks. The client only has access to the function signatures, not the actual implementation.

An RPC's flow is as follows:

- The client invokes the client stub, and the parameters are placed into the stack in the same way that a normal function call is.
- The client stub generates a package with the arguments and then calls the system to transmit the message. This is also known as marshaling.
- The message is sent from the client to the distant server by the client's local operating system.
- Incoming packets are sent to the server stub by the server's operating system.
- The server stub unpacks the message's arguments (unmarshalling).
- When the server process completes, the server stub marshals the return values.
- The message is subsequently sent to the transport layer via the server stub.
- The transport layer returns the resultant message to the client transport layer, which passes it on to the client stub.
- The caller's execution is resumed when the client stub unmarshalls the return value.

In this system, the client obtains data that is then fed into the training pipeline, which creates the updated local model. A Flower client on the client device receives the new weights and transfers them to the RPC client, which sends them to the server. The data is received by the RPC server, which then generates a new and enhanced global model using the given strategies. The server sends the revised global model to its clients through RPC, which will replace the prior local model. This is a high-level overview of how Flower ties into the machine learning workflow.

4 PROPOSED SOLUTION

4.1 Machine Learning

This will present the theoretical approach of the traditional machine learning implementation and the reasoning behind certain decisions.

4.1.1 Data

Before developing the models, an analysis of the obtained data is required to find probable trends and better comprehend the data set. This will be done with Python scripts and libraries such as math, numpy, and torch data loader. The goal of this stage is to gain a better knowledge of how to process the data, shape it into a valid input and output pair, divide it (train, validation, and test), and include in the main program.

Each record in the Gowalla data set, as described above, consists of a user id, latitude, longitude, time, and point of interest id. They are organized mostly by user id, from the most recent to the least recent check in. Given the project's goal of building a next-location prediction model, we will need to make certain changes to generate the inputs that we will use. Given that human mobility is impacted not only by the individual's present position but also by positions prior to the last check-in, the approach will be comparable to a time-series prediction algorithm. Given that the input will be a sequence of check-ins, data cleaning and preprocessing will examine not just the current item being assessed, but also the entries before and following it.

The data loader will perform the following functions:

- Read the raw input data from the Gowalla data set and transform it to a format that can be handled by the machine learning framework's functionalities.
- Remove outliers and invalid check-ins from the data. Many entries were eliminated for the following reasons: time between check-ins exceeded the maximum period permitted due to inaccurate timestamps or human "error," and check-in locations were invalid due to faulty coordinates or distance between spots that did not satisfy the requirements. Given that material was gathered through the API of a social network app rather than a formal research, we will attribute the lack of activity on the platform to human error in most cases.
- Preprocess the data and extract the necessary information in the form of an input-output pair. The input will be a sequence of N check ins from one user, with the suggested

features being the check in's delta time, latitude delta, longitude delta, and time till the next check in. The result will be delta latitude delta longitude values that have been normalized by scaling them to the range in which they reside.

- The results are divided into three categories: training data used to improve the model's accuracy (about 70% of the sequences collected), validation data used to visualize and debug how the model adapts to new data during training (about 15%), and test data used to evaluate the final generated model (the remaining 15% of the sequences).

```python
def loadData():
    data = read("Gowalla.txt")

    for checkin in data:
        if checkin is valid:
            input, output = extractFeatures(checkin)

            userCheckins.append(pair(input, output))

        if userCheckins.length == REQUIRED_CHECKINS:
            processedData.splitAndAdd(userCheckins)
            userCheckins.clear()

    return processedData.split()
```

Listing 4.1: Python example

4.1.2 Models

RNN, LSTM, and GRU were chosen from the models described in the "State of the art" section. These models are quite similar in their core, but by making little changes, there should be a visible variation in each model's behavior and performance.

RNN will be used as a reference for comparing and analyzing the outcomes because it is the simplest of the three. This model will be the focus of the typical machine learning technique throughout deployment. The LSTM and GRU will be evaluated using the parameters that were selected for this model to check if the results match the expectations. After attaining satisfactory results with the RNN implementation, LSTM and GRU will be used to replace the RNN network in the implementation.

There is a lot of trial and error in the early phases of implementation. Overfitting and underfitting issues, data management, and how much data is needed to update the model are all issues. Changes were made to the following parameters: learning rate, dropout probability, network size, number of data samples, and sequence length.

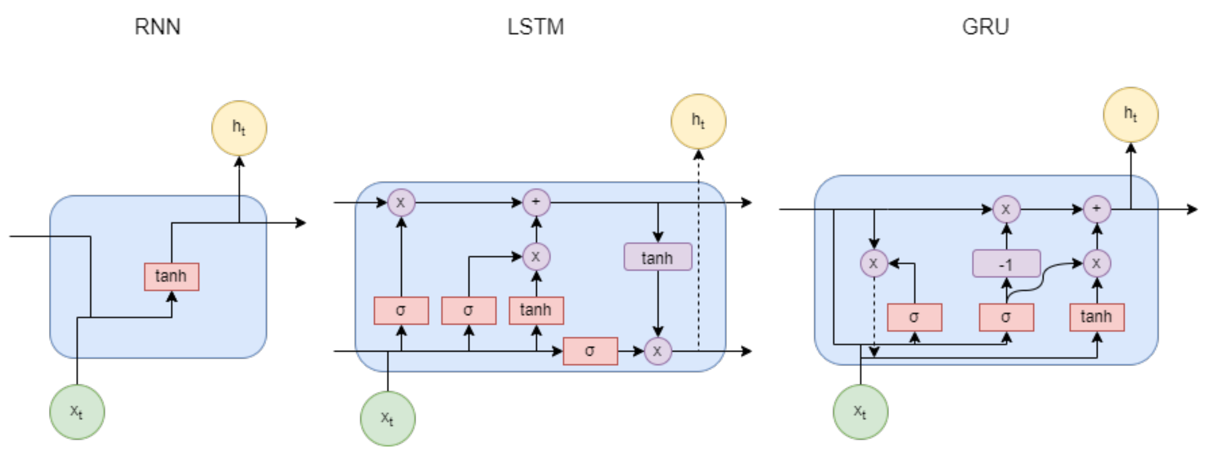

Figure 6: RNN, LSTM and GRU internal cells

4.1.3 Loss

In this project, the loss value will be a direct reflection of the accuracy of the performance of the model. Given how the features will be scaled and the overall strategy, the mean loss will be proportional to the delta difference between the individual's anticipated movement and the actual values recovered from the data set.

Given the nature of this project, the majority of development effort will be spent experimenting with alternative loss functions, optimizers, learning rates, dropout, network size and feature scaling in attempt to decrease this measure:

- The loss function is a means of determining how effectively a particular algorithm represents the provided data. If forecasts differ too much from actual outcomes, the loss function generates large values, causing considerable changes to the model's weights.
- After the loss has been estimated, optimizers such as SGD and Adam are used to update the real model.
- Dropout probability is a variable that aids overfitting models. Using this option directs the model to disregard some of the nodes at random during training. This may have the unintended consequence of generating a lower validation loss in comparison to the training loss in the initial rounds.

The key to developing an accurate machine learning model is to experiment with numerous ways based on the acquired findings and discover a combination of accessible functions and their parameters to reduce the total prediction error.

4.1.4 Workflow

The project's final workflow will consist of four major steps: data loading, training, validation, and testing.

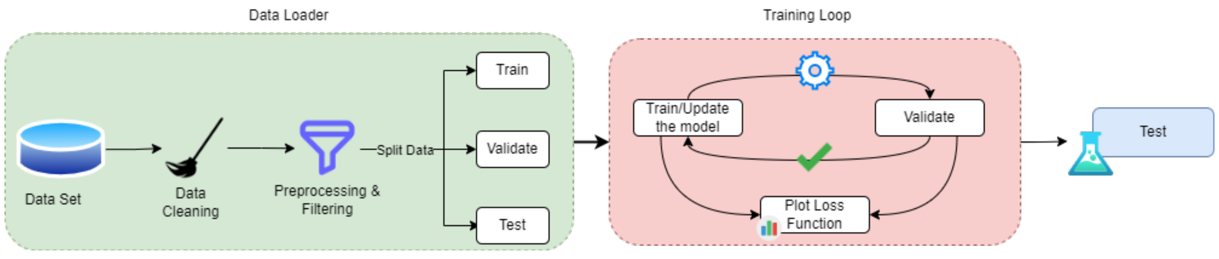

Figure 7: Workflow Overview

The main function will first setup the necessary parameters before calling the data loader to parse the data and extract the parameters. Following that, the Python script will begin by gathering the data, cleaning it, prepping it, and eventually passing it to the main function, as stated in the Data Loader paragraph.

When this is finished, the model, loss function, and optimizer are all initialized. If these procedures are successful, the training loop is initiated, which consists of two major processes: updating the model based on the training data and creating the loss value for the validation data.

After N number of epochs have passed and the training loop is complete, the model will get to run on the test data and provide us with the actual accuracy of the generated model.

4.2 Federated Learning

Changes applied to the machine learning architecture to fit the federated learning framework. These changes should be minimal so that they do not influence the accuracy and performance of the model more than what is required.

4.2.1 Preparing the Data for the Clients

There are two main components of federated learning that are getting added to the initial implementation: client and server. The client is very similar to a wrapper over the original implementation but with some slight alterations regarding data collection and updating the model.

Because this is a simulated environment, the clients are not need to be completely isolated. This provides two options for implementing the section that deals with providing data to each client. The first would be to perform the preprocessing algorithm and then store the data into multiple files that can then be passed as a command line argument for every client to read and gather the information. Another option is to maintain the present data loader implementation and produce additional clients from the code itself.

4.2.2 Aggregation and Global Model Update

Although selecting techniques and rules for planned data collection and model updating is not within the scope of this project, it is necessary to have a basic grasp of how this is performed and what consequences it may have.

This is specified mostly on the server or simulator side. On the server, these parameters are configured and how weights are gathered and the model is updated is determined by using the FedAvg function or by implementing the Flower framework's strategy interface, which provides greater overall control.

The server will begin to execute several rounds after the clients have been generated, either by running multiple instances of the client script or by utilizing other mechanisms, and the strategy for collecting weights and updating has been established. The server starts up and waits for N clients to connect and initialize their data in order to contribute to the update process. When the required amount of clients have enrolled, the server will begin the first round. Clients are chosen at random and begin updating their local model and exchange weights with the server. The weights are aggregated using FedAvg, and a new global model is created. This model will be broadcasted to the clients and utilized from now on.

5 IMPLEMENTATION

The technology stack that was used in the end for developing this projects is as follows: Python as the main programming language, PyTorch for implementing the machine learning models and Flower for adapting the implementation to federated learning. Other libraries used: math, numpy, geopy.distance, matplotlib.

5.1 Machine Learning

In this section, we will detail how we used the selected technology stack to implement the components outlined in the previous chapter. This will cover the standard machine learning approach, which will be enhanced later using the Flower federated framework.

5.1.1 Data Loader

The data loader is the project's basis and has gone through several iterations throughout development. The initial objective of this module was to read the input data, categorize it by users, and divide it into the three categories that would be utilized for the model (train, validation and test). After this was completed successfully, further details had to be gathered in order to obtain the information needed to generate and scale the input features.

Information such as the range of values in each field, mean values, and standard deviation. Observing these raw statistics output numbers, it was clear that a data cleaning process required to be implemented. As a result, values that did not fall within the typical spectrum, such as latitude outside of the range [-90, 90], longitude outside of the range [-180, 180], or timestamps outside of the time-frame in which the data was acquired.

After additional testing and methods to data processing, the following input characteristics for the model were chosen: latitude, longitude, delta since the last check in for latitude, longitude, time, and delta till the next check in. Delta latitude and delta longitude are the output characteristics.

Both min-max normalization and conventional scaling with z-score were used to normalize the data, but in the end, min-max normalization provided better overall results.

Considering the high variance between check in times and distances another two filters were introduced to exclude/split the sequences: time between check ins should not be higher than 48 hours and distance should not exceed 5 kilometers. As mentioned before the entries are not directly removed from the usable data but instead the script marks this sequence as completed and checks if the minimum number of 40 check ins has been registered for this sequence.

The data loader's final change before providing the information is to randomize the order of the sequences. This has the effect of decreasing selection bias caused by the sequence in which the check ins were gathered. This is also done on the training data throughout the training loop, although it does not totally deal with selection bias and simply normalizes how the weights are updated during training.

5.1.2 Models

The models were built with the PyTorch framework, which allows a large number of options by extending the torch.nn.Module class. RNN, LSTM, and GRU models have been implemented, as detailed in earlier sections.

There are two methods that must be created in order to implement the model class for each of them: __init__ (the constructor) and forward (computes output Tensors).

To generate a functioning model, three essential components must be defined in the constructor method:

- Initialization of the parameters, in this instance the size and number of layers of the hidden layer.
- Defining the model-specific layer responsible for forward propagation.
- Defining the completely linked layer as well as any further layers. To make these models as basic as feasible, only one completely linked layer was developed in this scenario.

```
1  self.hidden_layers = hidden_layers
2  self.layer_dim = layer_dim
3
4  self.rnn = nn.RNN(
5      input_dim,
6      hidden_layers,
7      layer_dim,
8      batch_first=True,
9      dropout=dropout_prob
10 )
11
12 self.fc = nn.Linear(hidden_layers, output_dim)
```

Listing 5.1: Python RNN initialization using PyTorch

The code snippet in Listing 5.1 is applicable to all three models, with the exception of line 4, which should be replaced with the model-specific function.

In Listing 5.2 is presented the code for the forward method of the RNN class in which are performed the following computations:

- Initializing hidden state for first input with zeros
- Forward propagation by passing in the input and hidden state into the model
- Reshaping the outputs in the shape of (batch_size, seq_length, hidden_size) so that it can fit into the fully connected layer
- Convert the final state to our desired output shape (batch_size, output_dim)

```
h0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_layers).
    requires_grad_().cuda()

out, h0 = self.rnn(x, h0.detach())

out = out[:, -1, :]

out = self.fc(out)
return out
```

Listing 5.2: RNN Forward Function

For the LSTM model the forward function also initializes the initial cell state for each element in the input sequence:

```
c0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_layers).
    requires_grad_().cuda()
```

Listing 5.3: LSTM Forward Adjustment

5.1.3 Training Loop

Despite its simplicity, the training loop is an essential component of our project. This is a common approach with two major components: training and validation as presented in the code snippet from Listing 5.4: Training Loop.

```
for _ in range(epochs):
    model.train()
    for x, y in trainDataSet:
        optimizer.zero_grad()
        output = model(x)
        loss = loss_fn(y, output)
        loss.backward()
        optimizer.step()

    model.eval()
    with torch.no_grad():
        for x, y in validationDataSet:
            output = model(x)
            loss = loss_fn(y, output)
```

Listing 5.4: Training Loop

The training stage iterates over all potential sequences in the training data set, attempting to predict the most likely outcome using each given item. After generating the output, which consists of the predicted delta latitude and delta longitude, the loss function is invoked and computes how far the model's output is from the actual values. Using this value, the chosen optimizer function, stochastic gradient descent (SGD), performs a backward propagation step to update the model's weights.

The validation loss is supposed to be generated in the second stage. The model computes certain predictions for all provided sequences using another segment of data, and the loss function is utilized to determine the error between the output and the reference values. In comparison to the previous stage, there are no model improvements here because the backward propagation function is not called. The loss numbers obtained here will be utilized subsequently to diagnose possible problems with the model's training.

5.1.4 Main

The program's main function is where all of the previously stated components (data loader, model initialization, training loop, and evaluation) come together to build the script that will generate the model for next-location prediction.

Given the path to the data set's file, the data loader is setup and begins processing the data, yielding one tensor for each feature in each category.

If no errors were found in the previous stage, the meta parameters are set to define the environment in which the model will be trained. Some of the factors that impact the training process are: the number of epochs, network size parameters (input, sequence size, layer size, number of layers, output size), and optimizer parameters (learning rate, momentum, dropout probability, weight decay).

The main function is also in charge of gathering and presenting information in an understandable manner when the training and evaluation loops are running. Loss values for validation and training are gathered for each epoch and recorded to standard output for future reference. After the training cycle is completed, these results are displayed, providing a useful reference for visualizing what unfolded. Some of the flaws that may be seen with this figure are: overfitting, underfitting, vanishing gradient [14] or exploding gradient [18].

Finally, when all of the training has been completed and the training process has been analyzed, the assessment process begins. The test data is used to assess the accuracy of the resulting model, and the program outputs the prediction results for this segment of the data set.

5.2 Federated Learning

In this section, we'll go through how we implemented the necessary changes to migrate the implementation to the Flower framework. The main components on which we will emphasize are: creating data partitions, the server, and the clients.

5.2.1 Generating Data Partitions

As previously stated, each client receives a random number of sequences. These data set partitions are generated by a new component introduced in the project. This partition generating script invokes the data loader to retrieve all valid values from the data collection. Following the completion of the preprocessing stage, a random quantity of data is chosen to be stored in each client's file, which will be identifiable by the client's unique id.

We discovered that by creating totally randomized volumes of data at this step, we were occasionally starving some of the clients since some received the majority of the data samples while others received nearly none. This example was producing very poor and inconsistent results because clients with little to no data were being sampled excessively for training and were unable to provide any helpful input to the server.

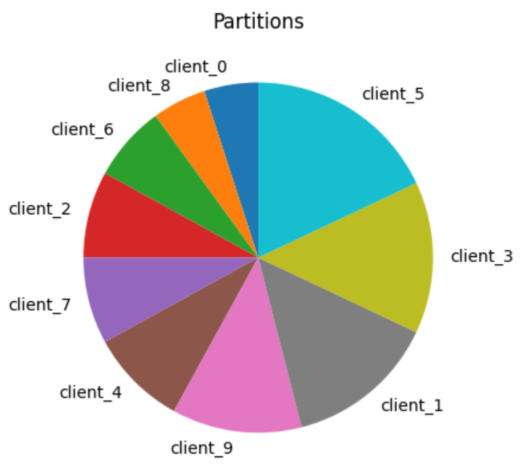

Figure 8: Example of generated data partition

There is also the case of giving equal amounts of data to each client. This method produced very good results but it was not a real representation of how this system would work if it would of been deployed in a real world test environment.

Because we have limited control over how customers are chosen, we had to impose some constraints in order to standardize the quantity of data that each client received. The numbers that best matched our use case while still introducing a degree of randomness to ensure a realistic test environment were a minimum of 5% and no more than 20% of the total available data.

5.2.2 Server

The initial step in transitioning to the federated learning paradigm utilizing Flower was to create the server. Using this framework, the procedure was simple and free of obstacles. We defined the port on which the server would operate, the number of training rounds, and the strategy as presented in Listing 5.5 and done so by following the available documentation.

The server will listen on the provided port for any clients that choose to participate in the training process. The framework has the possibility to build a secure connection using both authentication mechanisms and SSL certificates, however this was not required for this use-case.

In the conventional machine learning approach, the number of rounds is comparable to the number of epochs. This informs the server to execute N rounds of randomly sampling clients who entered the pool and updating the global model based from the findings.

```python
strategy = fl.server.strategy.FedAvg(
        fraction_fit=0.5,
        fraction_eval=0.2,
        min_fit_clients=4,
        min_eval_clients=1,
        min_available_clients=int(NUM_CLIENTS),
)

def start_server():
    fl.server.start_server(
        server_address="localhost:8080",
        config={"num_rounds": 100},
        strategy=strategy,
    )
```

Listing 5.5: Flower federated learning server

There are several implementations available for designing the approach, such as FedAvg, FedAdaGrad, or FedAdam, each with its own set of parameters that may be tuned to enhance the learning process. Regardless of how these strategies improve the model's weights, they all have one thing in common: the sampling policy. In this strategy, parameters specifying how many clients to sample for training and testing, the minimum number of clients for each operation, and the minimum number of clients in the pool may be provided.

5.2.3 Client

The client implementation required additional changes to the original source code, both in the training and data loading procedures. The same data loader as before will be used to partition the data according to each federated learning client, but the resulting features will not be delivered directly. Instead, each client will receive an randomly selected amount of sequences. This was accomplished by writing another script with the goal of equitably separating the data and saving it into multiple text files that were indexed.

In an ideal test environment, we would be able to spin up one client for each user in the data set, but due mainly to hardware constraints, we will have to settle for 10 unique instances of the client program that utilize a random amounts of the data set. This will not provide a a one-to-one comparison to a real-world environment, but it should provide a decent starting point for assessing how the model will behave.

```
for id in range(NUM_CLIENTS):
    client_process = Process(target=start_client, args=(id, ))
    client_process.start()
    processes.append(client_process)
```

Listing 5.6: Starting Flower clients

Listing 5.6 shows how the main program uses the multiprocessing Python library to launch multiple instances of the client application. Each instance is assigned an id, which the client will use to identify the files containing the associated partition of the data set, and which should be used when the servers inform the client that it has been picked for a training or test round.

6 EVALUATION

6.1 Machine Learning

This section will compare the performance of the standard machine learning algorithm to establish a reference for evaluating the transition to the federated learning approach. The evolution of the loss values during training and the overall accuracy of the final model will be the key topics of discussion.

6.1.1 Loss

During testing we were able to observe the impact that dropout probability has on the way the model updates. This meta parameter is primarily used to prevent overfitting of the model but sometimes it can have the side effect of causing the validation or training loss to be lower than the actual training loss. This phenomenon can be observed in Figure 8, which presents

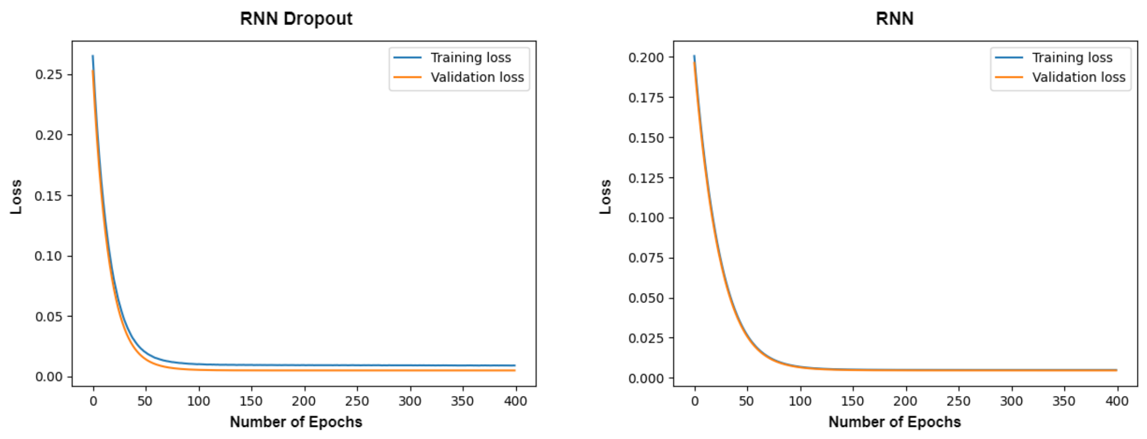

Figure 9: RNN Comparison

the evolution of loss function during training of the RNN model. Here we can observe that when introducing a dropout probability of 20% the model's validation loss has a value of 0.0051 whereas the actual training loss is 0.0091 which is 78.43% higher. Even though in both cases we can see that after 100 epoch we are faced with diminishing returns in regards to improving loss values, the final result when evaluating the model against training data is about 10.32% higher for the model that uses dropout probability because it slows down the learning too much.

The progress of the LSTM training process is seen in the following data from Figure 9. Again, a dropout probability of 20% is used, but the difference between training and validation is almost non-existent in this circumstance. Because of the supplementary gates (forget, input, and output) introduced by this neural network, the model updates at a slower pace than its RNN counterpart. Even after 100 epochs, we can find considerable differences in loss values when compared to the RNN model. On top of this, the final model does not converge to a final state despite running for 400 epochs.

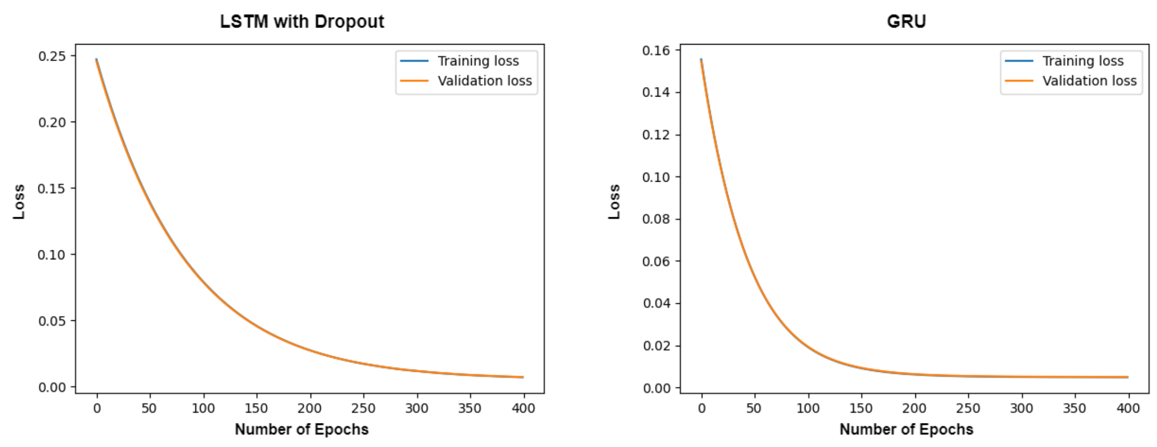

Figure 10: LSTM and GRU Loss

This approach yields a stronger correlation between train and validation but has the disadvantage of slowing down the learning rate. This may be improved by increasing the actual value of the learning rate, but the purpose of this comparison is to evaluate the models using essentially the same meta parameters so that the environment in which they run is not changed.

Figure 9 also depicts the final model that was tested, GRU. Given the results achieved thus far, this appears to be a good middle ground between the satisfactory results produced by the RNN network and the enhanced correlation of the LSTM network. In Table 3 we can see that when introducing the dropout probability the best performing network is GRU.

Table 3: Loss: lower is better

	0% dropout			**20% dropout**		
Data segment	**RNN**	**LSTM**	**GRU**	**RNN**	**LSTM**	**GRU**
Train	0.0049	0.0082	0.0048	0.0091	0.0069	0.0053
Validation	0.0046	0.0083	0.005	0.0051	0.0071	0.0049
Test	0.0048	0.008	0.0049	0.0053	0.0068	0.0052

From Table 3 we can observe that the best performing network in terms of loss function raw values is the RNN network with no dropout probability. In the following section we will analyze how this metric translates into the actual precision of the models and the overall accuracy. Dropout did not present any improvements in terms of the final values of the loss but it will be a good meta parameter that can be used for training the local federated learning models that tend to be more inclined to suffer from over-fitting.

6.1.2 Accuracy

We will examine the actual accuracy of each model implemented in this section using two metrics: distance error and Top N accuracy. The mean value of the errors between the actual next position and the predicted values obtained during the testing stage will be used to determine the distance error. Top N accuracy is a metric frequently used in classification algorithms that is defined as the likelihood of the actual value being predicted in the top N results.

Table 4: Evaluation metrics

	0% dropout			**20% dropout**		
Metric	**RNN**	**LSTM**	**GRU**	**RNN**	**LSTM**	**GRU**
Distance error	0.5377	1.0339	0.5497	0.6002	0.9031	0.5639
ACC@1	0.0529	0.0126	0.0479	0.04	0.0174	0.0468
ACC@5	0.2478	0.0536	0.227	0.1832	0.0735	0.1992
ACC@10	0.3685	0.0964	0.3514	0.2886	0.1319	0.3192

According to the findings collected and reported in Table 4, the Recurrent Neural Network achieves the best overall accuracy with a mean distance error of 0.5377 kilometers and top 10 accuracy of 36.85%. In comparison to other articles that examined similar implementations to ours, such as [21], the final findings demonstrate that we were able to create a competitive implementation of classic machine learning techniques. The accuracy of the top 1 results is 1.5% lower than that of the conventional networks, but the top 10 accuracy outperforms even the Flashback RNN implementation provided in [21] [11] for the RNN implementation by 2.06%.

Regarding the underperforming LSTM network, this was an expected result in terms of model accuracy given the loss function values acquired and examined in the preceding section. This is also the only network that has an increase in accuracy after applying the dropout probability which resulted in an mean distance error that decreased by 12.65% and a top 10 accuracy increase of 3.55%.

After evaluating the evolution of the loss function for all implementations, the rankings remain unchanged for both versions with and without dropout probability enabled. When adopting a dropout probability of 20%, GRU outperforms RNN and LSTM, with the latter failing to converge to a final state after 400 epochs. For the version with no dropout added, RNN outperforms all models, followed by GRU and LSTM, for the same reason as previously stated.

6.2 Federated Learning

In this section we will compare the overall accuracy of the models after re-adapting them using the Flower framework to simulate a federated learning environment. Regarding the testing environment that has been set up we have a total of 100 training rounds in which 5 out of the 10 available clients are randomly selected and run only once to train the local model. For validation the same principle is applied as in the training round but in this case 2 clients are selected from the pool.

6.2.1 Loss

First, the loss function values will be examined for the federated learning strategy, just as they were for the machine learning technique previously described.

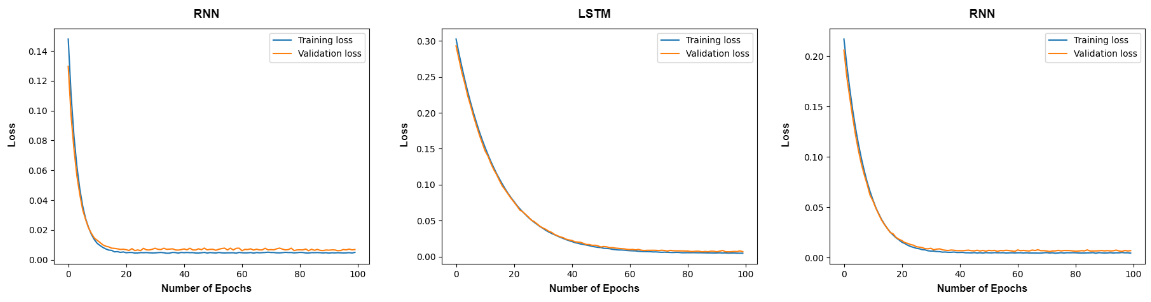

Figure 11: RNN, LSTM and GRU Loss

We can see a very similar trend, in which the RNN quickly converges to a final state and then stops updating. This is reinforced by the overfitting that may occur in each client's local model due to the limited data available for training.

LSTM optimizes at a slower rate, but compared to prior findings, it has reached the point of diminishing returns in terms of enhancing the model's accuracy. This result also shows how over-fitting affects the final model's training, but this time in a beneficial way because LSTM is more competitive when the final loss values generated after testing are considered.

Table 5: Loss: lower is better

	0% dropout			**20% dropout**		
Data segment	**RNN**	**LSTM**	**GRU**	**RNN**	**LSTM**	**GRU**
Train	0.0049	0.0046	0.0044	0.0047	0.0051	0.0055
Validation	0.0068	0.0069	0.0069	0.007	0.007	0.0069
Test	0.0068	0.007	0.0068	0.0069	0.007	0.0068

Finally we have GRU, which this time manages to slightly outperform the RNN during training. All things considered the final values are very similar and we can not make a clear decision on which of these models performs better by only taking into consideration the loss values.

Table 5 shows that this time dropout probability has little to no effect on the loss function for the testing values and produces somewhat worse outcomes during training. Even though this parameter should aid reduce overfitting and build a more generic model, it appears that even at a low value of 20%, it causes more harm than good.

6.2.2 Accuracy

We will evaluate the overall accuracy and precision of the models in the federated learning environment using the same metrics previously suggested. The metrics chosen are mean distance error and Top N accuracy.

Because of the way weights are handled on the server and the data is not concentrated on a single computer, somewhat lower results are predicted during this evaluation. Each model does a local optimization of the local model before being sent to the server. FedAvg is used by the servers to produce the new weights, which are then shared with all clients.

By evaluating the mean distance error from Table 6, we can see that it is doubled for RNN and GRU but nearly unaltered for LSTM. As with the loss values, RNN and GRU appear to suffer the most from the transition to federated learning. Given that the distance between check-ins in the gathered inputs is around 4.71 kilometers, an error of 1.17 kilometers is a 25% difference between the actual values. This is not ideal, but we must study and see how it affects the main metric, which is Top N accuracy.

Table 6: Evaluation metrics

	0% dropout			**20% dropout**		
Metric	**RNN**	**LSTM**	**GRU**	**RNN**	**LSTM**	**GRU**
Distance error	1.1618	1.1707	1.1585	1.1664	1.1745	1.1609
ACC@1	00368.	0.0245	0.0349	0.0289	0.02629	0.0342
ACC@5	0.2109	0.1398	0.2315	0.1735	0.1359	0.2079
ACC@10	0.3235	0.2362	0.3404	0.2816	0.233	0.3223

LSTM continues to have the lowest accuracy when compared to the other models, but it is the only model that improves over the conventional implementation. This demonstrates that federated learning may be used to improve the performance of sluggish implementations that require several rounds to attain acceptable loss levels.

Out of all the implementations tested, GRU with 0% dropout probability performs the best. It has an accuracy of 34.04%, which is just 1.1% lower than the typical machine learning GRU accuracy and 2.81% lower than the highest performing model, RNN. GRU's enhanced complexity aided it in maintaining a high level of accuracy when compared to earlier findings.

6.2.3 Client sampling

Splitting the data and customer sampling are two aspects that must be examined in order to provide a comprehensive picture of how we arrived at these conclusions. We discussed how data is maintained in earlier sections, so we will just address it quickly here because these two elements are connected and determine the rate at which the models update, and how we produced results very similar to the initial implementation in the federated learning implementation.

When creating the files that contain the data sequences, we extract a random percentage ranging from 5 to 20 percent of the overall data. We created these margins after observing that when this procedure is entirely randomized, the data is not distributed in an equitable manner. The initial batch of clients received far more data than the final ones to join the pool, with some obtaining as little as 0% of the available data-set. The best results were obtained during testing when we had an equal split across all clients, however this had to be discarded because it was not relevant in real-world scenarios.

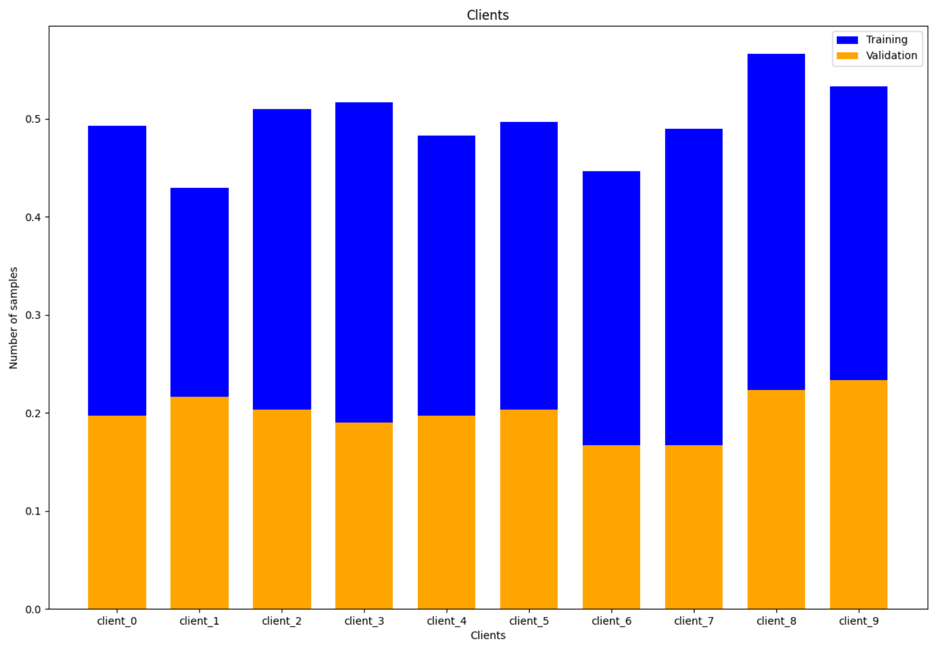

Figure 12: Clients sampling rates

Figure 11 depicts the sample rate for each unique client following many training and testing sessions. As we can see, over several iterations, all of the customers' sample rates tend to reach 50% for training and 20% for testing rounds. These are the same values as those stated in the server's strategy. In the case of a single iteration of the script, because we are only running 100 rounds for training and testing, certain clients may receive a greater sampling rate, which might have a detrimental influence on the model depending on how much data they were provided. Using the data in Figure 11, for example, if the client with id 8, with the highest sampling rate of 56.66%, received just 2% of the data, the results for that iteration would be substantially worse than the ones studied earlier.

Because we do not have that much control over how the clients are getting selected during training we have to introduce some boundaries regarding how much data does one client receive so that we do not starve the others.

7 CONCLUSIONS

After analyzing the results from the machine learning and federated learning implementations, we can infer that the ultimate accuracy of the models that perform best in a typical test environment has decreased. Given the added benefit of improved data privacy that federated learning provides, a drop in error that ranges between 0.7 and 4.5 percent is a reasonable compromise.

Trading performance for other factors such as enhanced stability, memory efficiency, or security is a typical dilemma in computer science engineering. Most of the time, finding an implementation that checks all the boxes is difficult or impossible task, and some sacrifices must be made in order to create the solution that best suits the request or problem at hand.

The results also demonstrated some amazing outcomes in terms of enhancing one of the implementations following the transition to federated learning. This was rather unexpected, and it may provide another topic of discussion in terms of scenarios in which federated learning might be utilized to improve typical machine learning solutions.

There is definitely potential for expansion and improvement: additional networks evaluated, assessing how this solution performs when utilizing other data sets, and even altering the test environment. As previously stated in this research, we encountered hardware limitations during the federated learning testing and were only able to test this with only 10 distinct clients. We cannot say with certainty that this would function in the real world environment, but it is a good starting point for anyone who may wish to explore this topic.

Other enhancements that might be done in the future in this configuration for our implementation would include the inclusion of the social graph provided by the Gowalla data collection. Other articles in the field of human mobility research have employed metrics relating to the phycology and social interactions of the individuals who participated in these trials [3] [9]. This component of the data set might be used to extract metrics such as the people with whom the user has engaged, the number of social connections the user has, and places visited by other people in the user's social network.

BIBLIOGRAPHY

[1] *Benefits of Cloud Computing for Business Enterprises: A Review*. SSRN, 2019. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3463631#.

[2] Ed Baker. Exif custom: Automatic image metadata extraction for scratchpads and drupal. *Biodiversity Data Journal*, 1:e973, 2013.

[3] Mariano Beiró, Andre Panisson, Michele Tizzoni, and Ciro Cattuto. Predicting human mobility through the assimilation of social media traces into mobility models. *2193-1127*, 5, 01 2016.

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[5] Rodrigo Fernandes de Mello and Moacir Antonelli Ponti. *Machine Learning. A Practical Approach on the Statistical Learning Theory*. Springer, 2018.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

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

[8] INSTICC. *Blockchain-based Traceability of Carbon Footprint: A Solidity Smart Contract for Ethereum*. SciTePress, 2020.

[9] Dong Yup Kim and Ha Yoon Song. Method of predicting human mobility patterns using deep learning. *Neurocomputing*, 280:56–64, 2018. Applications of Neural Modeling in the new era for data and IT.

[10] P. Leonard. Customer data analytics: privacy settings for 'big data' business. *International Data Privacy Law*, 4(1):53–68, 2013.

[11] Massimiliano Luca, Gianni Barlacchi, Bruno Lepri, and Luca Pappalardo. A survey on deep learning for human mobility. *ACM Comput. Surv.*, 55(1), nov 2021.

[12] Kemal Aydin (eds.) M. Emre Celebi. *Unsupervised Learning Algorithms*. Springer, 2016.

[13] Steven Gonzalez Monserrate. The Cloud Is Material: On the Environmental Impacts of Computation and Data Storage. *MIT Case Studies in Social and Ethical Responsibilities*

of Computing, (Winter 2022), jan 27 2022. https://mit-serc.pubpub.org/pub/the-cloud-is-material.

[14] Seol-Hyun Noh. Analysis of gradient vanishing of rnns and performance comparison. *Information*, 12(11):442, 2021.

[15] Xi Ouyang, Chaoyun Zhang, Pan Zhou, Hao Jiang, and Shimin Gong. Deepspace: An online deep learning framework for mobile big data to understand human mobility patterns, 2016.

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[17] Md. Mokhlesur Rahman, Kamal Chandra Paul, Amjad Hossain, G. G. Md. Nawaz Ali, M. Rahman, and Jean-Claude Thill. Machine learning on the covid-19 pandemic, human mobility, and air quality: A review, 03 2021.

[18] Alexander Rehmer and Andreas Kroll. On the vanishing and exploding gradient problem in gated recurrent units. *IFAC-PapersOnLine*, 53(2):1243–1248, 2020.

[19] Chance Simmons and Mark A. Holliday. A comparison of two popular machine learning frameworks. *The Journal of Computing Sciences in Colleges. Papers of the 33rd Annual CCSC Southeastern Conference*, 35(4):20–25, 2019. `http://www.ccsc.org/publications/journals/SE2019.pdf#page=20`.

[20] Eran Toch, Boaz Lerner, Eyal Ben Zion, and Irad Ben-Gal. Analyzing large-scale human mobility data: a survey of machine learning methods and applications. *Knowledge and Information Systems*, 58, 03 2019.

[21] Dingqi Yang, Benjamin Fankhauser, Paolo Rosso, and Philippe Cudre-Mauroux. Location prediction over sparse user mobility traces using rnns: Flashback in hidden states! IJCAI'20, 2021.