ARCADA

Federated Backdoor Attacks against Speech Recognition

Khanh Trung Mai

Master's Thesis

Big Data Analytics

2024

Master’s Thesis

Khanh Trung Mai

Federated Backdoor Attacks against Speech Recognition.

Arcada University of Applied Sciences: Big Data Analytics, 2024.

Abstract:

This thesis explores and simulates federated learning, specifically focusing on applying adversarial attacks against input data to assess their impact on training performance. In the context of the rising importance of federated learning for preserving data privacy in machine learning, this research investigates scenarios where data and models remain decentralized, eliminating the need to transfer them to a central location and thereby safeguarding raw, sensitive data from exposure. The primary dataset under consideration is keyword-specific, involving the analysis and classification of sound waves recorded from real individuals. In addition to real-world data, synthetic data generated through cloud services is incorporated to augment the dataset, providing insights into its influence on the training phase. The study delves into various adversarial attack types, encompassing label manipulation and sound alteration, aiming to assess their impact on the federated learning model’s robustness.

Keywords: Federated Learning, Backdoor attacks, Speech Recognition

Contents

Figures

Tables

List of Algorithms

Abbreviations

AI	Artificial Intelligence
IoT	Internet of Things
FL	Federated Learning
ML	Machine Learning
AWS	Amazon Web Services
ASR	Automatic Speech Recognition
RMS	Root Mean Square
PBSM	Pitch Boosting Sound Masking

1. Introduction

Edge machine learning and federated learning share common goals and principles. However, federated learning takes the capabilities of edge machine learning a step further by allowing collaborative model training across multiple edge devices while maintaining data privacy and security(Ma et al. 2023a, Jeong and Chung 2022). Unlike traditional machine learning, where raw data is collected and aggregated into a centralized server, federated learning conducts model training directly on edge devices. This decentralized approach minimizes privacy concerns associated with sharing sensitive data and minimizes the need for extensive data transfer over the network. Federated learning is no longer limited to academic research. Its use has spread to various industries, such as healthcare, finance, and sales, where data privacy is crucial in protecting customers and legal compliance(Bharati et al. 2022).

On the other hand, cyber-attacks have become significant threats, often overlooked in favor of physical security concerns. These attacks provide an easy way to disseminate harmful content, collect unauthorized data, and manipulate human decision-making processes. Cybersecurity is paramount in federated learning because of the framework's distributed and collaborative nature.

This study examines the effect of various attacks on edge devices' data on the efficiency of centralized machine learning models that use federated learning, neural networks, and a range of adversarial attack tactics. The research uses a mix of real-world and artificial datasets to enhance training performance.

2. Related Work

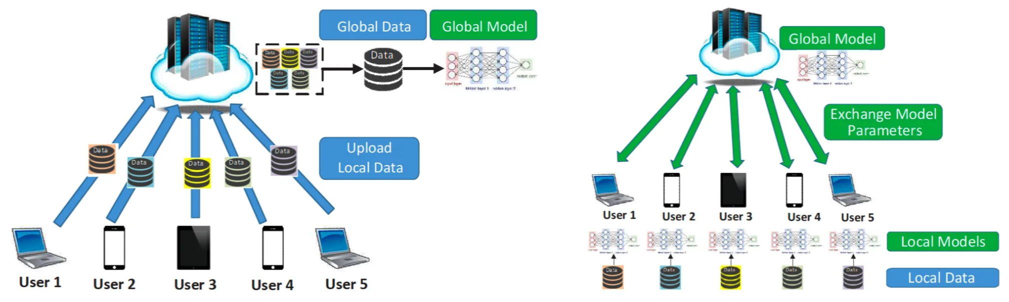
2.1 Federated Learning

Data privacy has become a sensitive topic due to immense cyber attacks on customer data (Mrini et al. 2024). Many machine learning applications are trained using centralized data, where customer information is collected on a server and then used for training. However, this method raises concerns about the security of sensitive data. To address this issue, decentralized AI has become a new trend and standard for handling and protecting sensitive data. Federated learning is one such approach, which processes and uses local data for training at the customer's end. This approach has resolved the issue of adversarial attempts, making it a promising solution for improving data privacy. Google first introduced federated learning in 2016 (Martineau 2022) when people were concerned about data breaches by raising multiple social network platforms, such as Facebook, where attention was focused on the potential threat of sharing personal information through the Internet. The main benefit of distributed learning is that the model is collaboratively trained on the customer device without sharing information with the central server, thus creating a trade-off.

Unlike conventional centralized models, federated learning distributes the model training process across these nodes, allowing them to compute updates on their respective datasets locally. These updates are aggregated at a central server, consolidating the collective knowledge without exposing raw data. This architecture preserves privacy by keeping sensitive information localized, enhancing scalability and efficiency. Federated learning holds promise in scenarios where data privacy is paramount, fostering collaborative model improvement across diverse datasets while minimizing the need for large-scale data transfers. As a cutting-edge framework, federated learning presents a compelling solution for addressing privacy concerns and accommodating the distributed nature of contemporary data sources.

2.1.1 Federated Averaging

A core tenet of federated learning revolves around the algorithmic orchestration of collaborative training for a global machine-learning model across decentralized devices. Several prominent algorithms have emerged in this domain, such as Federated Averaging



(a) centralized learning (can be outsourced learning). (b) distributed learning

Figure 1. Centralized learning vs Distributed learning

(FedAvg)(McMahan et al. 2016), Dynamic Regularization (FedDyn)(Acar et al. 2021), and Federated Stochastic Gradient Descent (FedSgd)(Yuan and Ma 2021), each with its set of advantages and limitations, contingent upon factors like complexity and specific use-cases (Shastri 2023). In the context of this thesis, the chosen algorithm is Federated Averaging (FedAvg), a decision made after carefully considering its suitability for the defined objectives and constraints of the research.

Federated Averaging builds upon the strengths of FedSGD (Polyak and Juditsky 1992) by incorporating a refined approach to parameter averaging. In this iterative process, client weights initialized from the same rounds undergo averaging, allowing for the fine-tuning of local parameters before their transmission to the central server for aggregation. This nuanced strategy addresses the challenges associated with variance in gradients from training rounds.

The FedAvg process unfolds through iterative rounds, where participating devices engage in local model training on their respective datasets. The locally trained models' parameters are then transmitted to a central server, orchestrating the aggregation process. Notably, instead of sending raw data, devices share model updates, specifically the gradients of the loss function concerning the model parameters. The central server adeptly aggregates these updates, computes the average, and redistributes the refined global model to the participating devices, thus optimizing the collaborative learning experience. As explained in (McMahan et al. 2016), the key factors that control the result are: C, the part of total clients that participate in each round; E, the number of local training rounds for each client with its local dataset; and B, the local batch size used for each client updates.

Algorithm 1 illustrates the mechanics of FedAvg: the server initializes the weights and randomly selects participants for each round. Subsequently, each chosen client undertakes local training with its data, adhering to predefined epochs and batch numbers. The resultant gradient descent values are then transmitted to the server and employed in the subsequent training round.

Algorithm 1 FedAvg with the K clients indexed by k, the B mini-batches indexed by b and E as the number of epochs

```
ServerSide:
    initialize $w_0$;
    for each round $t = 1, 2...$ do
        $m \leftarrow$ Choose number of participants
        $S_t \leftarrow$ Randomize $m$ clients
        for each client $k \in S_t$ in parallel do
            $w_{t+1}^k \leftarrow$ ClientSide$(k, w_t)$
        end for
        $m_t \leftarrow \Sigma_{k \in S_t} n_k$
        $w_{t+1} \leftarrow \Sigma_{k \in S_t} n_k / m_t * w_{t+1}^k$
    end for
ClientSide: Only run on selected client $k$
    $b \leftarrow$ (splits local dataset into batches size B)
    for each local epoch $i$ from $1$ to $E$ do
        for batch $b \in b$ do
            $w \leftarrow$ (train and update $w$)
        end for
        $w_{t+1}^k \leftarrow$ ClientSide$(k, w_t)$
    end for
    return $w$ to server
```

2.2 Adversarial Attacks Against Machine Learning

As discussed in previous research in early 2024 (Apostol Vassilev 2024), understanding the various types of attacks on ML systems is crucial for assessing their security and resilience against adversarial manipulation. In ML security, attacks can be broadly categorized into white-box, black-box, and gray-box attacks, each representing different levels of adversary knowledge and access to the ML system. These distinctions are pivotal for evaluating the robustness of ML models and developing effective defense mechanisms.

White-box attacks occur when the attacker fully knows the ML system, covering details like training data, model structure, and settings. Although these attacks rely on solid assumptions, they are vital for stress-testing systems against the most knowledgeable adversaries and devising effective defenses. This definition also includes adaptive attacks, where the attacker closely monitors any countermeasures applied to the model or the system. A study in 2023 (Ma et al. 2023b) dived into white-box attacks, including non-targeted and targeted ones, utilizing the fast gradient sign method (FGSM) and projected gradient descent (PGD). Real datasets collected from a LoRa testbed were used to demonstrate the efficacy of these adversarial examples against convolutional neural networks (CNNs), long short-term memory (LSTM) networks, and gated recurrent units (GRU).

In contrast, black-box attacks assume very little about the ML system. The attacker might only have access to query the model, lacking any insights into its training process. These attacks are seen as the most practical because they simulate situations where attackers have no prior knowledge of the AI system and rely solely on its standard interfaces. For example, An attach strategy (Papernot et al. 2017) aimed to train a local model to replace the target DNN by utilizing inputs generated synthetically by an adversary and labeled by the target DNN. They then utilized this local substitute to generate adversarial examples, which were observed to be misclassified by the targeted DNN.

Gray-box attacks lie in between, representing varying degrees of adversarial knowledge. For example, the attacker might know the model's structure but not its specific settings, or they might possess the model's settings but not details about the training data. These attacks often assume access to data similar to the training set and familiarity with the feature representation process. This latter assumption is particularly crucial in cybersecurity, finance, and healthcare, where feature extraction plays a significant role before training ML models. In a recent study (Lapid and Sipper 2023), an attack framework was introduced that created adversarial examples in image-to-text models, typically consisting of an image encoder and a transformer-based decoder. Unlike image classification tasks, where there's a finite set of class labels, generating visually similar adversarial examples for image-to-text tasks is more challenging due to the infinite space of possible captions.

2.3 Adversarial Attacks Against Decentralized Learning

As the popularity of Decentralized Learning rises, an increasing number of hackers are attempting to compromise either the model or the data. The trend towards Decentralized Learning has attracted heightened attention from malicious actors seeking to exploit vulnerabilities in the model or access sensitive data. The growing prevalence of Decentralized Learning has consequently led to a surge in hacking attempts, posing significant challenges to the security of both models and data.

In 2019, researchers discussed how to backdoor the federated learning models using the Model Replacement method (Bagdasaryan et al. 2019). The method was used as the baseline method for the experience, where the attackers changed the learning rate and the number of local epochs to direct the model for overfitting. As in federated learning, during the server model update, the training phase only selects a small number of clients; hence, this attack needs the backdoored participants to be chosen frequently; otherwise, the impact is minimal. The attacker intentionally attempts to send the server an adversarial model with updated gradients. This method described an attack in federated learning where an attacker aimed to replace the global model with a malicious one. The attacker iteratively adjusted their model to mimic the global model while incorporating a backdoor. They scaled up the backdoored model's weights to ensure its survival during the averaging process. Even without knowing specific parameters, the attacker could successfully attack by approximating the scaling factor. Additionally, the attacker could exploit vulnerabilities such as skipping the random masking of weights or selectively removing close-to-zero weights. This attack is effective, especially when the global model is nearing convergence.

As the setup for this experiment, 100 rounds were trained with a selected number of clients from each round participating in two different tasks: CIFAR and word prediction. After introducing the backdoor into CIFAR models, the backdoor accuracy initially dropped and gradually increased. This pattern occurred due to two main factors. First, the objective landscape was not convex, meaning it had multiple optimal points. Second, the attacker used a low learning rate to align their model with the current global model, making surrounding models devoid of the backdoor. As benign participants updated their models with higher learning rates, they moved away from the attacker's model, causing the global model's backdoor accuracy to drop. However, since the central model was nudged towards the backdoored model's direction, it was likely to converge to a backdoored version again. The attacker faced a situation: using a higher learning rate prevented the initial drop in backdoor accuracy but risked creating a notably different model from the global one. Conversely, in word-prediction models, the backdoor accuracy remained stable because most of the model's weights were word embeddings, rarely updated by participants. Consequently, even if the trigger sentence was uncommon, associated weights remained at the local extreme point set by the attacker.

Another study in 2022 (Cao and Gong 2022) demonstrated an adversarial attack against federated learning. This attack involved injecting fake clients into the system and creating a fake local training model to feed the server with malicious content. As the baseline attack, random Gaussian noises were used as the gradient descents for the malicious model. Then, based on that, the MPAF (Model Poisoning Attack based on Fake clients) method was invented to manipulate their local model updates to steer the global model toward a predetermined base model. In each round of FL, fake clients are generated and updated by subtracting current global model parameters from the base model's parameters and then scaling them up. The main challenge of this approach is the attacker's limited knowledge of the FL system, relying only on received global models. MPAF's strategy involves forcing the central model to mimic the base adversarial model.

2.4 Automatic Speech Recognition

Speech recognition, a subfield in AI and machine learning, involves the development of methodologies and applications that enable human-machine communication. Automatic Speech Recognition (ASR) technologies play a pivotal role in this domain, facilitating various industry applications. For instance, virtual assistants like Amazon's Alexa and Google Assistant leverage ASR to comprehend and respond to user voice commands, enhancing the interactive capabilities of smart devices. Moreover, in the realm of transcription services, ASR proves invaluable, converting spoken content from meetings, interviews, and lectures into written text. Customer service and call centers also benefit from ASR (Goyal et al. 2022), automating and improving the transcription of customer calls for real-time monitoring and analysis. Additionally, in the education sector, ASR is applied to transcribe lectures and generate subtitles, fostering accessibility in e-learning platforms. These examples underscore the broad spectrum of applications wherein speech recognition technologies significantly contribute to seamless human-machine interactions.

2.5 Research Questions and Objectives

Significantly, the speech recognition field has notably lacked studies and experiments concerning targeted attacks against input data. This research aims to address this gap by focusing on decentralized learning with classification tasks, particularly against harmful attacks. This approach enables us to explore and experiment with various methods to manipulate edge-device local datasets. We will also utilize cloud services to generate synthetic datasets, which can serve as valuable assets for attacks and defenses. In the subsequent chapters, we will analyze the framework, and based on the results, further inquiries can be unfolded:

- How can the performance of federated learning be affected by each attack?
- How will the model recover after each attack during the training round?
- Would synthetic data improve the training performance?

3. Research Methodology

3.1 Simulation Setup

For conducting simulations in this thesis, the Flower Framework is selected due to its capacity to simulate machine learning and federated learning without the necessity of handling an extensive array of physical devices. The Flower Simulation feature facilitates the configuration of clients and servers as code within a central node. This eliminates the need for a complex server setup and allows for the execution of simulations on a computational system to validate various scenarios, aligning seamlessly with the objectives of this thesis. The framework's flexibility in representing clients and servers through code simplifies the simulation process, providing an efficient and scalable solution for exploring diverse aspects of machine learning and federated learning.

In our computational framework, we leverage Puhti, a robust supercomputer facilitating scalable simulations for experimental purposes. Puhti offers an array of supercomputing nodes characterized by high memory capacities or storage, catering to diverse task requirements. Notably, Puhti supports CPU and GPU Linux partitions, catering to heavy-load tasks and providing versatility in computational capabilities. The system's scalability and support for parallel jobs make it especially well-suited for simulation workloads, enhancing the efficiency and effectiveness of our computational experiments.

To emulate the behavior of 40 clients undergoing 20 training rounds, our experimentation leverages the Hugemem Puhti partition job. This specialized job allocation offers an extensive computational resource pool, boasting 1496GB of CPU capacity and 3600GB of NVMe data storage. The allocated job has a time constraint of 3 days, and it facilitates the concurrent execution of up to 160 tasks. This configuration ensures the efficient simulation of a large-scale federated learning scenario, where the substantial computational resources and simultaneous task execution capabilities of the Hugemem Puhti partition contribute to the fidelity and scalability of our experiments.

Each client undertakes local training, utilizing 2 CPUs to process the provided input data. Given the intricacies of the deep learning model, a substantial resource allocation is imperative for a comprehensive simulation. To accommodate the complexity, we reserve a powerful 1465GB of memory, deploy 40 CPUs, and allocate 2000GB of NVMe storage for the simulation. These provisions are crucial to ensure efficient model training and robust performance. Notably, a timeout of 3 hours is imposed for each simulation scenario, striking a balance between computational efficiency and the practical constraints of timely completion.

3.2 Architecture Design

In this session, a comprehensive description of the employed architecture is presented. The process begins with segmenting the dataset into smaller subsets, each designated for consumption by individual clients. Following the dataset split, a targeted adversarial attack introduces poisoned data into the training phase. The training procedure commences on the local client machines, after which the refined models are transmitted to a centralized server. Notably, the training phase unfolds over 20 rounds, emphasizing the iterative nature of the learning process and the collaborative exchange between local clients and the central server to enhance the overall model performance.

3.3 Benchmark Dataset

We use the speech command (Warden 2018) hosted by Huggingface to support the state of the art. The dataset contains one-second WAV files that record commands by various confirmed speakers, enabling training models for automatic speech recognition tasks. Version two, released on April 11th, 2018, was used for the experimentation and contains around 100,000 audio files with 36 labels. Table 1 describes the dataset distribution and labels.

3.4 Synthetic Dataset

3.4.1 Introduction

AWS Polly (Barr 2016) provides text-to-speech services using deep learning technologies with various voices and accents that support testing the robustness of the simulation. AWS Lambda is a serverless computation service that allows us to generate large datasets on a large scale. AWS S3 is an object storage service that can host our dataset comfortably. One of the thesis goals is to use both natural and synthetic voices for training and simulation purposes. According to the Huggingface dataset, the synthetic dataset should

Table 1. Speech Command Dataset Distribution

Labels	
Name	Index
yes	0
no	1
up	2
down	3
left	4
right	5
on	6
off	7
stop	8
go	9
zero	10
one	11
two	12
three	13
four	14
five	15
six	16
seven	17
eight	18
nine	19
bed	20
bird	21
cat	22
dog	23
happy	24
house	25
marvin	26
sheila	27
tree	28
wow	29
backward	30
forward	31
follow	32
learn	33
visual	34
background noises	35

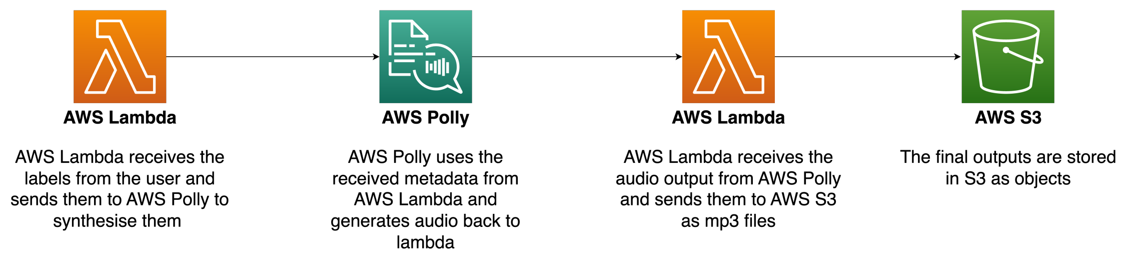

AWS Lambda
AWS Lambda receives the labels from the user and sends them to AWS Polly to synthesise them
AWS Polly
AWS Polly uses the received metadata from AWS Lambda and generates audio back to lambda
AWS Lambda
AWS Lambda receives the audio output from AWS Polly and sends them to AWS S3 as mp3 files
AWS S3
The final outputs are stored in S3 as objects

Figure 2. Architecture using AWS services to generate synthetic dataset

contain 35 labels for the classification task. As illustrated in Figure 2, we use the AWS Python SDK(AWS 2024) to trigger the Lambda function asynchronously. This generates the output for all labels concurrently, consuming only 5 minutes to synthesize around 70000 sound files for 35 labels. The output files are then stored in an S3 bucket and contained in the corresponding folder with labels as the name. For example, the synthetic data for the label 'zero' is stored in the folder /synthetic-data/zero/labelhash.mp3.

3.4.2 Output and cost

We collected 70,695 records from 35 different labels, with a combined size of 197MB. On average, each sample was about 2.5 KB. The total computation, synthesis, and storage cost was less than 2 USD, as shown in Figure 3. Our architecture was designed to be highly scalable, allowing for generating synthetic datasets for research at a low cost with efficient storage.

3.5 Data Pre-processing

We are provided with sound files that contain 16000 wavelengths as input. The signal is transformed using the Fast Fourier Transform(Wikipedia 2024) method into a spectrogram and component frequency information to train the model. This is achieved using a given frame length and step, as shown in Figure 4.

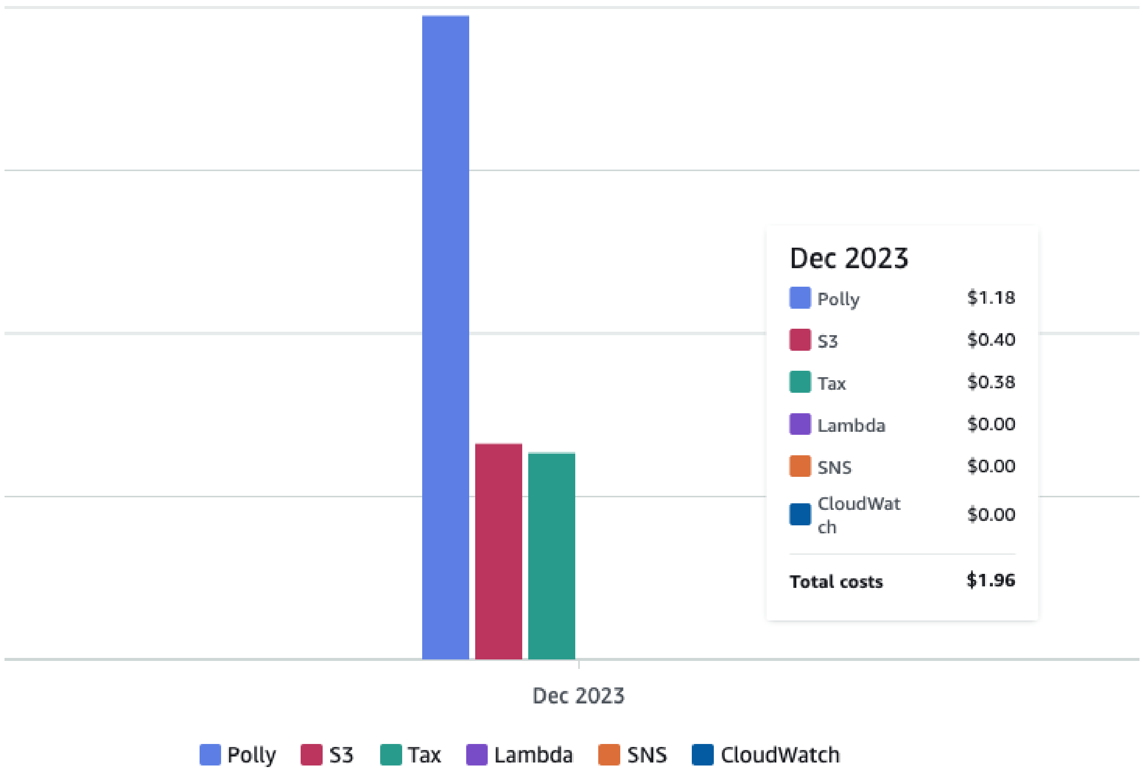


Figure 3. Total Cost for AWS services

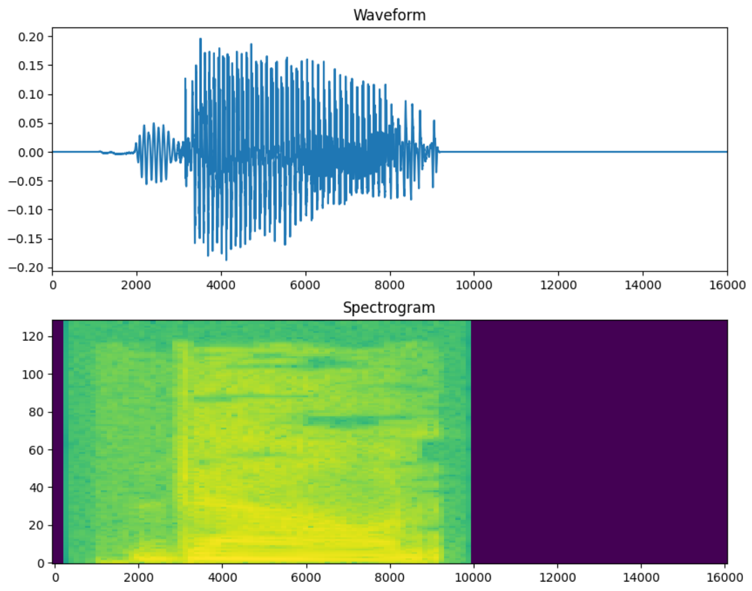


Figure 4. Waveform and Spectrogram

Table 2. Synthetic Dataset Distribution

Labels		
Name	Index	Number of files
yes	0	2000
no	1	2000
up	2	2000
down	3	2000
left	4	2000
right	5	2000
on	6	2000
off	7	2000
stop	8	2000
go	9	2000
zero	10	2000
one	11	2000
two	12	2000
three	13	2000
four	14	2000
five	15	2000
six	16	2000
seven	17	2000
eight	18	2000
nine	19	2000
bed	20	2398
bird	21	2000
cat	22	2000
dog	23	2000
happy	24	2000
house	25	2000
marvin	26	2000
sheila	27	2000
tree	28	2297
wow	29	2000
backward	30	2000
forward	31	2000
follow	32	2000
learn	33	2000
visual	34	2000
Total size = 197.4 MB		

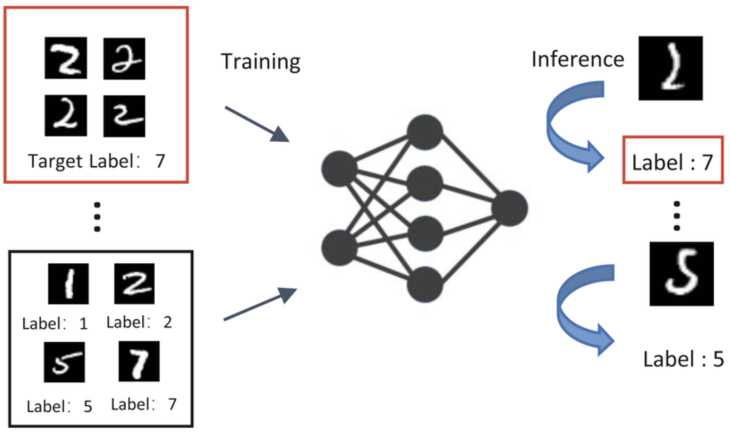


Figure 5. Label Flipping Attack

4. Adversarial Data Attacks

4.1 Flipping Labels

Flipping its labels(Jebreel et al. 2022) is one of the most straightforward attack methods in federated learning to manipulate a local dataset. By changing the labels of some examples from the partition, the attackers can quickly impact the learning performance. Using a label-flipping attack, the adversary poisons their local data by changing a source class to a target class without modifying the features, as demonstrated in Figure 5.

Algorithm 2 Label flipping

```
Input: Benign Data: $labels, percent, num_labels$
  $number\_targeted\_labels \leftarrow len(labels) * percent$ // Get the number of affected labels
  $targeted\_label\_indexes \leftarrow random(labels, number\_targeted\_labels)$ // Randomize targeted labels
  for $i$ in $targeted\_label\_indexes$ do
      $labels[i] \leftarrow random(num\_labels! = labels[i])$ // Flip the label of targeted signal
  end for
  $flipped\_labels \leftarrow labels$
Output: $flipped\_labels$
```

4.2 Adding Signal Noise

In constant to random noises that only influence the signal with a random distribution of values, colored noises are served in different fields based on unique characteristics. Those noises support engineering tasks like audio testing and calibration and help people with sleep, relaxation, and mental issues, as background sounds mask other disruptive noises.

In the scope of the thesis research, white noise and pink noise are chosen to analyze and apply with federated learning to see if they can affect the global training model locally and globally. The selected noises will mask the command sounds with a pre-configured ratio to ensure the poisoned command is still recognizable. Figure 6 shows the waveform and spectrum of the colored noises.

White noise(Jay Summer 2023) is a type of random noise that is dense across all frequencies, and each component is independent of the others, which has a similar sound as broadband noise, which can be observed from an untuned analog television. Similar to White noise, which also contains the sound components spreading across the whole spectrum, Pink noise has a lower pitch and value when compared to White noise. As being researched, it can be described as sounds of light rain, rivers, and winds. Algorithm 4 explains how we add noise to benign signals. First, we calculate the *RMS* of the targeted input and signal-to-noise (n) to get the RMS_n. With the noise as an adversarial signal, we modify the amplitude of the *noise* based on n to get the final malicious sample.

Algorithm 3 Add Real Noise with n as signal-to-noise ratio, noise as the adversarial signal

```
Input: Benign Data: $signal, noise, n$
Transformation:
    $RMS \leftarrow \sqrt{1/N\Sigma_{i=1}^{N} signal_i^2}$
    $RMS_n \leftarrow \sqrt{RMS^2/100^{(n/10)}}$
    $RMS_t \leftarrow \sqrt{\Sigma_{i=1}^{N} noise_i^2}$
    $noise \leftarrow noise * (RMS_n/RMS_t)$
Output: $signal + noise$
```

4.3 Stretching Sound

To enhance the representation of sound waves, we can manipulate the wavelength directly instead of introducing extraneous noise. A prevalent technique for achieving this is stretching the sound, which can influence the training model without distorting the essential spoken keywords. This approach ensures a more refined and controlled modification of sound characteristics, contributing to a more effective and targeted impact on

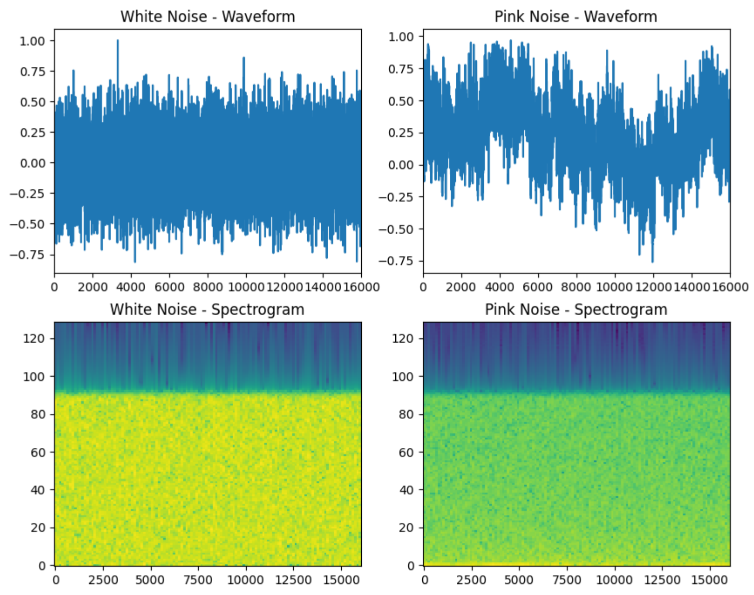


Figure 6. Colored Noise Waveform and Spectrum

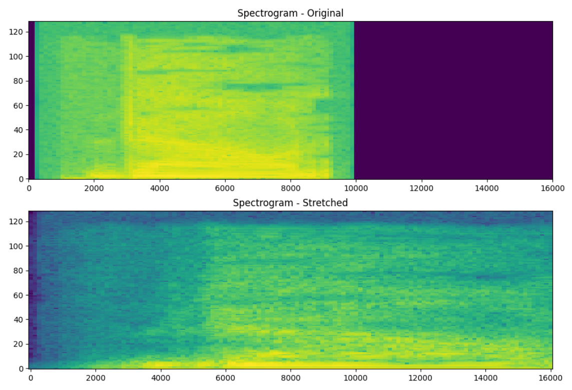


Figure 7. Stretching Sound by 50 percent

the overall audio quality. In Algorithm 4 and Figure 7, an example of 50 percent stretched sound is provided. We can use a benign signal (*signal*) and apply the phase vocoder technique (Prusa and Holighaus 2022) to stretch the input with a configured *sampleRate* to manipulate the speed of the sound.

Algorithm 4 Stretching Sound

```
Input: Benign Data: $signal, stretchRate, sampleRate$
Transformation:
    $advesarialSignal \leftarrow phase\_vocoder(signal, stretchRate, sampleRate)$
Output: $advesarialSignal$
```

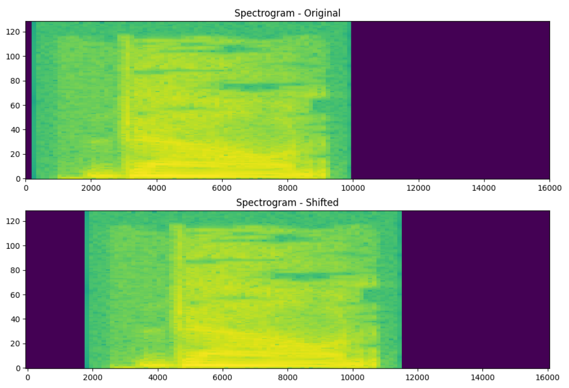


Figure 8. Shifting Spectrogram by 10 percent

4.4 Shifting Sound

In contrast to the method of stretching sounds, where modifications are made to both amplitude and frequency, an alternative approach involves subtly shifting the sound by small time durations to introduce a form of "poisoned" benign data. While these alterations regarding output sounds might go unnoticed by the human ear, their impact on the training model is direct and significant. The adjustments to the input for classification tasks create a nuanced layer of complexity, compelling the machine learning model to adapt to a broader range of temporal variations. Essentially, this strategic manipulation poisons the input data, imparting resilience and adaptability to the model, even in the face of imperceptible changes in the temporal domain. Using the benign data and a given $shiftRate$, an adversarial signal can be introduced by rolling the sample based on its $sampleRate$, as described in Algorithm 5 and shown in Figure 8.

Algorithm 5 Shifting Sound

```
Input: Benign Data: $signal, shift_rate, sample_rate$
Transformation:
    $shiftedSamples \leftarrow sampleRate/shiftRate$
    $advesarialSignal \leftarrow roll(signal, shiftedSamples)$
Output: $advesarialSignal$
```

4.5 Pitch Boosting Sound Masking

Pitch Boosting Sound Masking (Cai et al. 2022) is an adversarial attack method invented with the combination of boosting pitch and masking sounds to create negative samples that can poison training models.

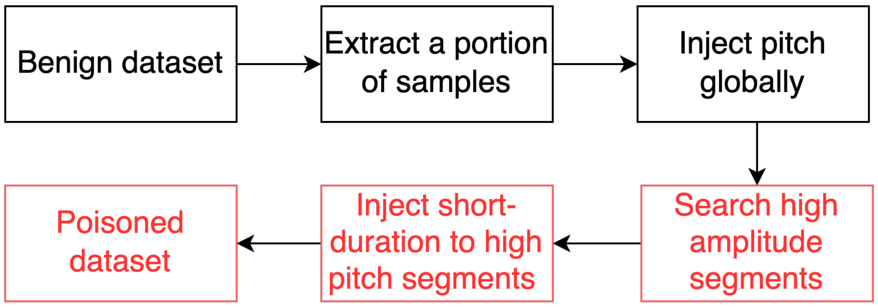

Benign dataset
Extract a portion of samples
Inject pitch globally
Search high amplitude segments
Inject short-duration to high pitch segments
Poisoned dataset

Figure 9. Pitch Boosting Sound Masking architecture

As explained in Algorithm 6 and Figure 9, a subtle pitch adjustment is applied globally after using the Fourier Transform to create the frequency domain of the benign signal ($signal\_boosted$). While imperceptible to the human ear during testing, this adjustment sets the stage for the subsequent manipulation. The next phase involves pinpointing specific high-pitched frequencies within the spectrum we intend to manipulate. Only a brief segment of the signal is selectively modified to maximize the efficacy of the attack. Upon identifying the high-amplitude signals, a controlled elevation in pitch is introduced to the targeted frequencies, ensuring a reasonable magnitude. In Figure 10, the modified high pitch was highlighted in red color to note the malicious trigger.

Algorithm 6 Pitch Boosting Sound Masking with hs as the high amplitude added to the signal

```
Input: Benign Data: $signal, hs$
  $signal\_boosted \leftarrow signal + 5$ //Add high pitch to the signal
  $max\_apm\_index \leftarrow getMaxAmpSegment(signal\_boosted)$ // Get the high amplitude segment index
  $signal\_boosted \leftarrow signal\_boosted[max\_apm\_index \pm 5] + hs$ // Insert a high-amplitude signal to the high segment
Output: $signal\_boosted$
```

Overall, the primary objective of this method is to specifically target high-frequency components within benign signals, generating a manipulated signal that proves challenging for human ears to discern. This approach aims to exploit nuances in audio perception, thereby enhancing the efficacy of the generated poisoned signal.

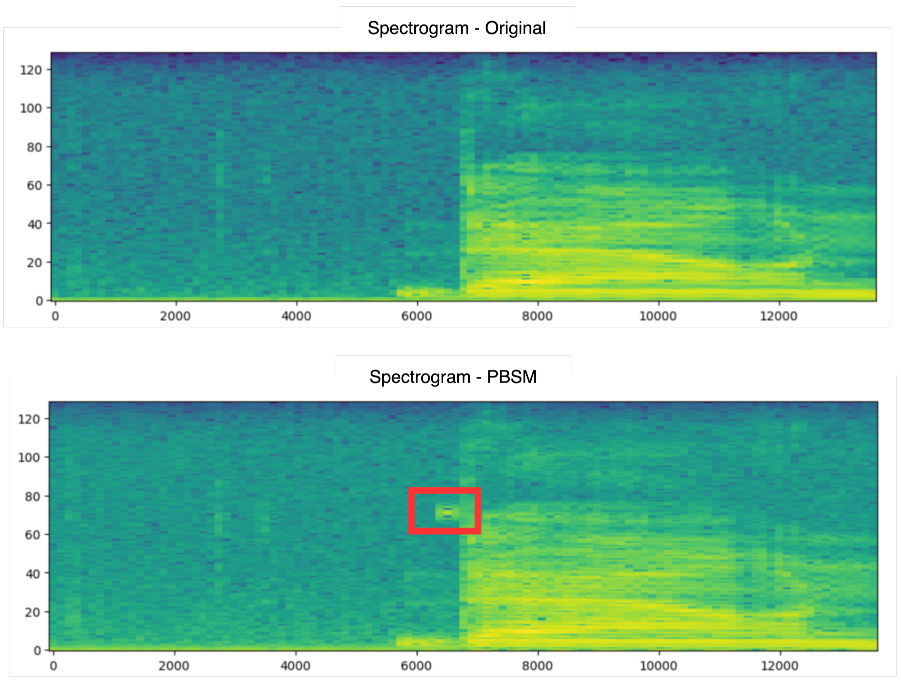


Figure 10. PBSM sample trigger (High pitch modified in red)

5. Results

Due to the insights acquired from our experimental endeavors, our analysis exclusively concentrates on the accuracy and loss scores of the centralized model. Our evaluation encompasses two key aspects: firstly, the utilization of the benchmark dataset in isolation, as well as in conjunction with synthetic data, allowing us to discern any consequential effects on performance. Secondly, we investigate the influence of introducing noise to the dataset on the training models. Our experimental framework is designed to examine carefully single and multiple adversarial attack types targeted at the training data, offering a comprehensive assessment of the robustness and resilience of the models under various conditions.

5.1 Benchmark Data vs Synthetic Data

With the combination of benchmark and synthetic data that we generated using AWS services to double the size of the training input, we expect to have a better performance from the clean data and a resilient centralized model against adversarial attacks. We continue the experiments with 40 clients and 20 rounds of federated learning to compare the centralized model between two different datasets.

Figure 11 visualizes the performances when using the benchmark dataset and combined with the synthetic dataset. As a result, the final accuracy score was improved from 0.85 to 0.87, while the loss was reduced from 0.58 to 0.52. The increase in the available input to each client explained the improvement in the final model. As we have many labels for the classification task, the input size plays a vital role in the distribution to each client when developing a federated learning simulation.

In the upcoming sessions, we will conduct various adversarial attacks on data, both with and without the mentioned noises—precisely, pink and white noises. In the subsequent experiments, additional attack methods were incorporated during the training phase outlined in the methodology section. This allowed us to assess and compare the effects of different attack types, helping us identify which one has the most significant impact on the task at hand.

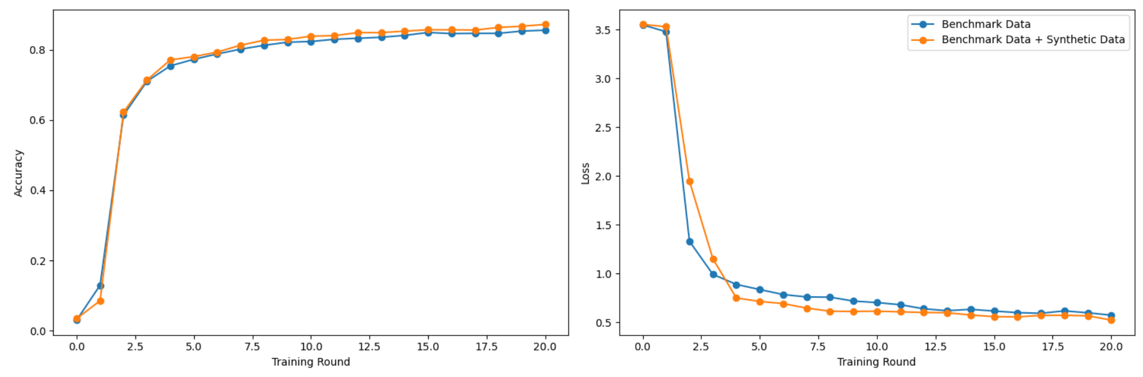


Figure 11. Benchmark vs Benchmark+Synthetic

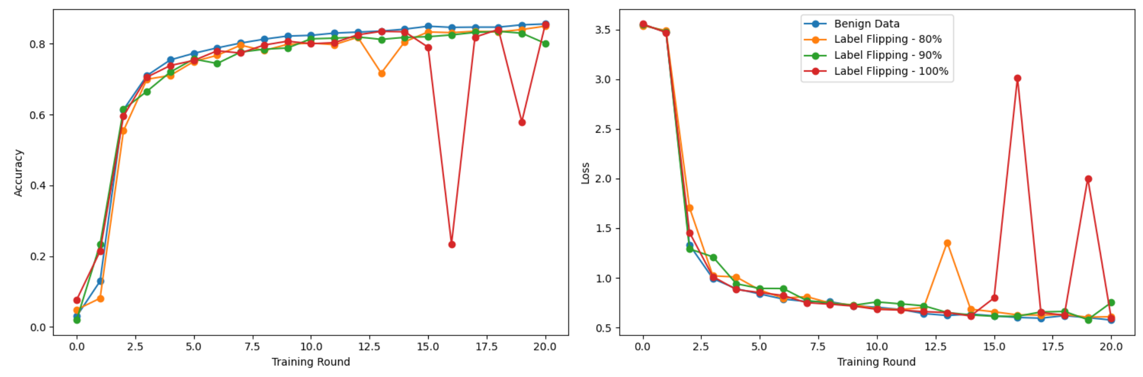


Figure 12. Label-flipping attack on benchmark data

5.2 Attack against benchmark dataset

We conducted two types of attacks on the benchmark dataset, namely PBSM and label-flipping.

5.2.1 Label-flipping

Firstly, Figure 12 visually presents the impact of label-flipping attacks on accuracy and loss; the percentage number indicates how the portion of data in an individual client is affected. Notably, this attack type exhibited substantial effects, causing a significant decline in accuracy during a specific training round when the targeted client was selected. For instance, the accuracy plummeted from 0.79 in the 15th round to less than 0.23 in the 16th round with a 100 percent label-flipping or from 0.81 in the 12th round to below 0.71 in the 13th round with an 80 percent label-flipping occurrence. With the 90 percent label-flipping, the performance from the last ten rounds was relatively stable, which can be understandable as the affected clients were not selected for training. This analysis suggests that label-flipping emerges as a potent attack, showcasing its effectiveness. However, the

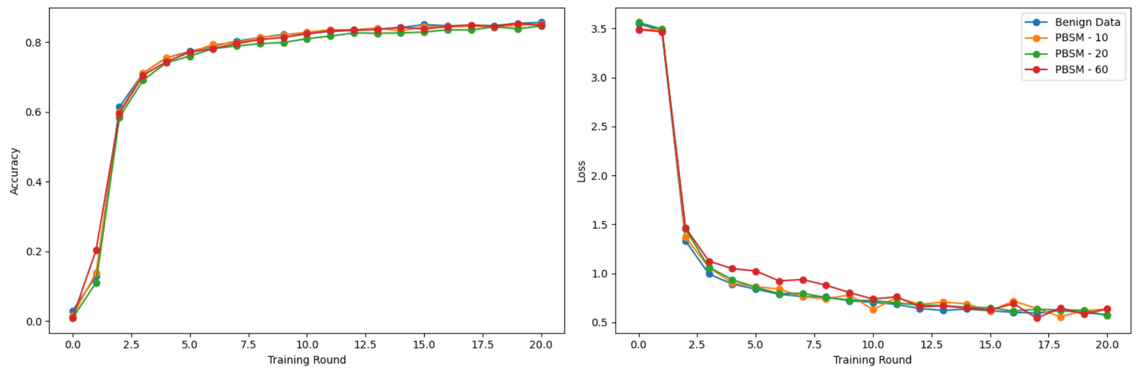


Figure 13. PBSM attack on benchmark data

discernible impact on the training model is undeniable, primarily attributable to the alteration of data labels. This alteration significantly influences the training model during the training round in which the targeted client is engaged. Consequently, this insight aids in the identification of affected clients, facilitating their exclusion from subsequent training rounds to mitigate the harmful effects on model performance.

5.2.2 PBSM

Secondly, Figure 13 presents the outcomes of the PBSM attack to show the impact of varying PBSM rates on model performance. Notably, PBSM-20 is the most influential variant, exerting a noticeable effect on the training model. Despite the participation of affected clients in the training process, the score exhibits a resilience that distinguishes the PBSM attack from a more overt label-flipping attack. As anticipated, the overall results of PBSM attacks demonstrate a reduction compared to benign data; however, the discernible impact remains relatively subtle and poses a more intricate challenge for detection than the previous attack type.

5.3 Attacks Against Combined Dataset

5.3.1 Label-flipping

After subjecting the researched attack types to the combined dataset in Figure 14, the label-flipping method notably impacted the centralized model. The model was trained with different percentages of affected labels under each malicious client. In the experiments, the accuracy during training dropped approximately from 80 percent to 70 percent. This decline aligns with the earlier findings, emphasizing the considerable fluctuation in performance when the label-flipping method was employed, particularly during the in-

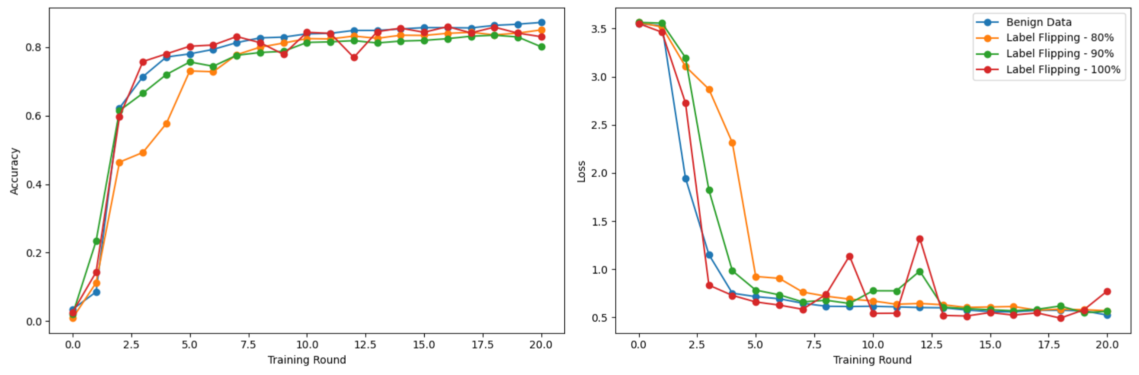


Figure 14. Label Flipping attack on benchmark data

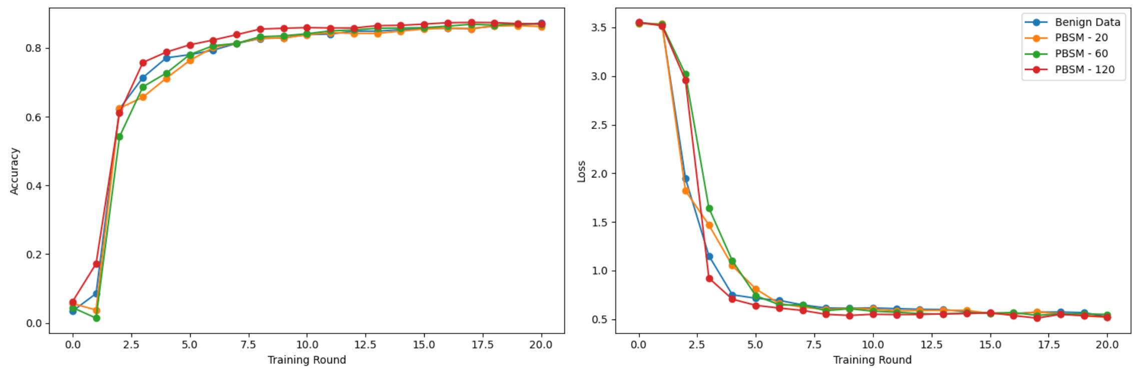


Figure 15. PBSM attack on combined data

volvement of the affected client in the training round. As shown in the result, the increased flipped labels affected the result immensely.

5.3.2 PBSM

Conducting various experiments with different configurations, it was found that the PBSM method had a negative effect on the central model's performance. However, the impact was less significant than that of the label-flipping method. The results showed that PBSM had a milder influence, reducing the accuracy by a maximum of two percent, as depicted in Figure 15.

5.3.3 Wave Stretching

The results of the wave stretching attack, as illustrated in Figure 16, have provided valuable insights into the behavior of the model during the training phase. It was observed that if an adversarial client participated in the training, there was a notable drop in the model's performance. The training accuracy was reduced from around 80 percent to 70 percent. This indicates that the presence of adversarial clients can significantly impact the

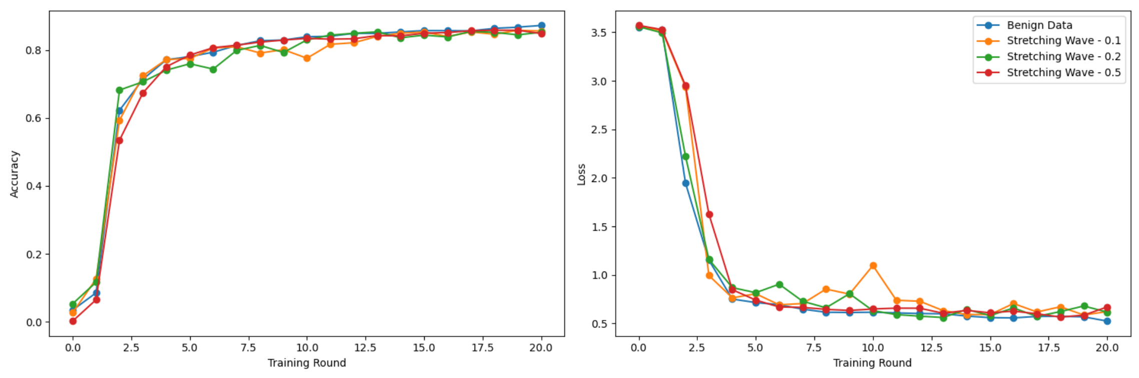


Figure 16. Wave Stretching attack on benchmark data

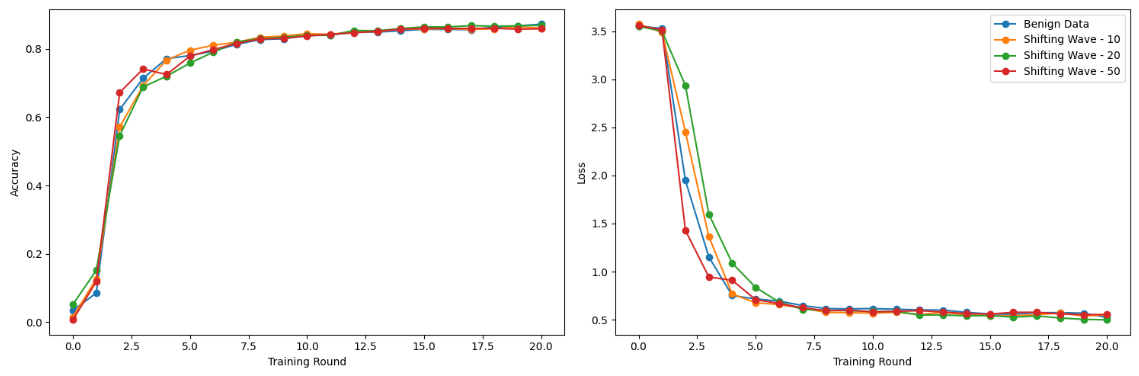


Figure 17. Wave Shifting attack on benchmark data

performance of the model during training.

5.3.4 Wave Shifting

During the course of a recent experiment, a wave-shifting attack was executed as per the graphical representation provided in Figure 17. Upon analyzing the results, it was observed that the affected client's performance experienced a substantial decline during the third round of training. The client's performance score dropped from approximately 70 percent to 65 percent, indicating a significant drop in performance accuracy. It is noteworthy that this reduction in performance was exclusively observed in the client that was subjected to the wave-shifting attack. None of the other clients participating in the same training sessions displayed any such decline in performance.

5.3.5 Real Noises Adding

The experiment conducted in Figures 18 and 19 involved adding white noise and pink noise to the client's input samples. The results showed that the strength of the added noise affected the central model. Specifically, as the noise increased in strength, the client model

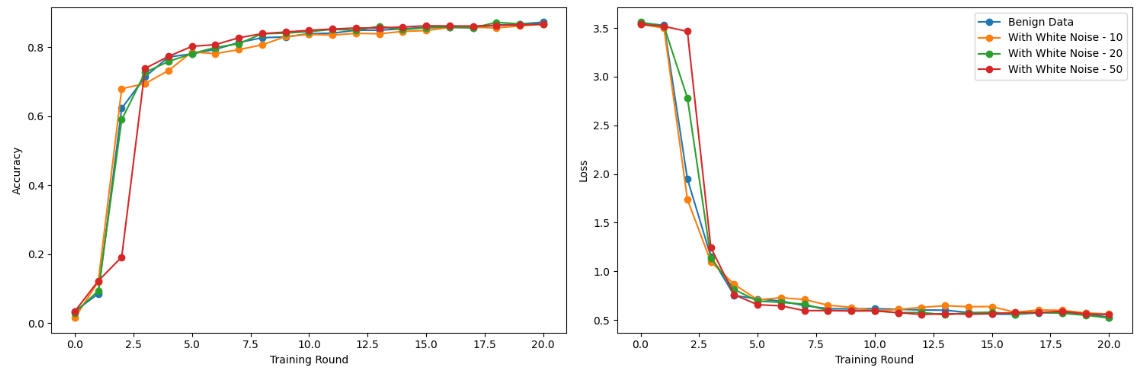


Figure 18. White Noise attack on benchmark data

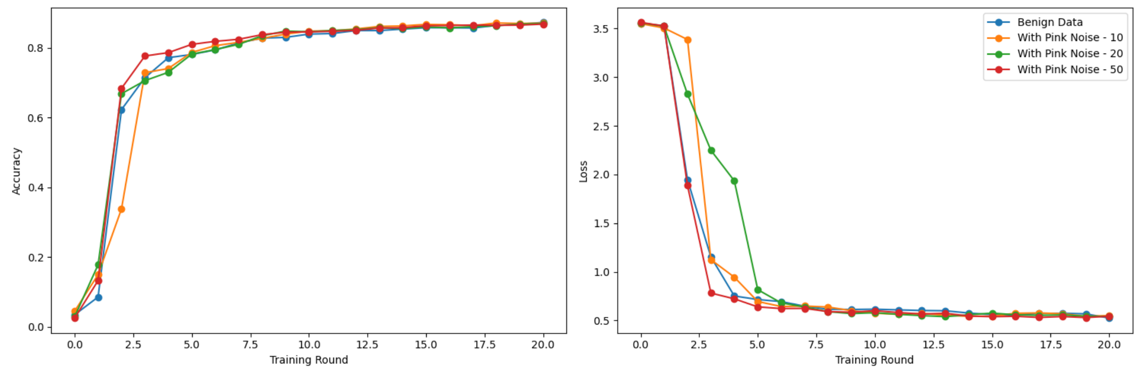


Figure 19. Pink Noise attack on benchmark data

had a greater impact on the central model. However, despite the added noise, the final model only experienced a 5 percent decrease in accuracy compared to the model trained with benign data. This suggests that the noise added to the input was relatively small and did not significantly impact the final model's accuracy. Overall, these findings provide valuable insights into the effects of noise on machine learning models and can inform future research in this area.d data. Applying these techniques resulted in the development of the central model, achieving a final accuracy of 86 percent for wave shifting and 85 percent for wave stretching.

6. Conclusions and discussion

6.1 Conclusions

Our experiment aimed to evaluate the impact of adversarial attacks on a federated learning model. We found that data manipulation can have a significant effect. We thoroughly analyzed each type of attack and compared their outcomes in detail. Our findings suggest that even minor modifications to the training data can significantly change the model's accuracy. Observing the results, we noticed that our federated learning framework can quickly recover from malicious attacks despite the drop in the accuracy caused by the adversarial clients.

In our experiment, we explored the possibility of creating synthetic data using public cloud services. The aim was to improve the performance of our machine learning model and enhance its resiliency against adversarial attacks. We found that by doubling the size of the dataset using synthetic data, we successfully enhanced the model's accuracy by ten percent. This is a significant improvement and has promising implications for the field of machine learning. By creating synthetic data, we can generate additional training data that is not based on real-world observations, which can help to mitigate overfitting and improve the robustness of the model. Furthermore, the use of public cloud services for synthetic data generation is a cost-effective and scalable solution that can be easily integrated into existing machine learning pipelines.

However, we acknowledge that our experiment was limited in terms of the number of participants, and we recommend conducting further research with a more significant number of clients and users. By doing so, we can achieve better simulations of real-world scenarios and better understand the potential impact of adversarial attacks on federated learning models.

6.2 Discussion

6.2.1 RQ1- How can the performance of federated learning be affected by each attack?

The system's performance was significantly impacted by adversarial attacks, which are malicious attempts to subvert the system's functioning. These attacks caused variations in the results obtained from the system across different attacks and configurations. In particular, the presence of malicious clients participating in the training round was found to have a significant impact on the performance of the system. This highlights the importance of ensuring the security and integrity of the training data and the overall system to prevent such adversarial attacks from compromising the accuracy and reliability of the system.

6.2.2 RQ2- How will the model recover after each attack during the training round?

In our analysis, we observed that the server model showed a high level of resilience in recovering from attacks, particularly when the number of clients chosen per round was limited. However, we also noted that the model's performance suffered significantly when subjected to more targeted manipulation, such as a label-flipping attack, which can result in incorrect classifications. We found that the model could only recover if the adversarial client were excluded from subsequent training rounds. This is because including the adversarial client in later training rounds would cause the model to learn from the erroneous labels and continue to make incorrect predictions. Overall, our findings highlight the importance of considering more targeted attacks when evaluating the robustness of machine learning models. Implementing appropriate defenses to mitigate these attacks is crucial, such as identifying and removing adversarial clients from the training process.

6.2.3 RQ3- Would synthetic data improve the training performance?

By leveraging synthetic data generated through cloud services, the training performance of the model witnessed a substantial improvement. This was primarily due to the fact that the synthetic data augmented the existing dataset and provided a more comprehensive and diverse set of data points for training the model. As a result, the model's resilience was significantly enhanced, making it more capable of handling complex real-world scenarios. This approach of using synthetic data is becoming increasingly popular in machine learning, as it allows for the creation of large and diverse datasets cost-effectively and efficiently.

6.3 Further Improvements

We recommend exploring this approach further, as it has been shown to yield promising results in recent studies. For instance, the paper by Zhang et al. (2022) (Zhang et al. 2022) demonstrates the efficacy of such attacks in improving the performance of a deep neural network used for image classification tasks.

By incorporating these advanced label-flipping attacks into the training process, we believe that models can achieve higher accuracy and robustness, even in the face of adversarial attacks. Therefore, we encourage researchers and practitioners to consider this approach in their future work to improve the performance of machine learning models.

In our research, we assert that the algorithm can be fine-tuned by modifying various presently defaulted and fixed hyperparameters. These hyperparameters, such as the learning rate, batch size, and number of epochs, are crucial in determining the algorithm's performance. By altering these parameters, we can optimize the algorithm to achieve better results and observe how attacks affect it in different configurations, as proposed in (Collins et al. 2022).

Furthermore, we argue that the synthetic dataset created in this thesis has significant potential beyond its current role as a benchmark. The dataset is generated using advanced techniques such as Generative Adversarial Networks (GANs) and can be used to enhance the robustness of the final model. Additionally, the dataset can be employed to generate adversarial data to threaten federated learning. By introducing adversarial data into the training process, we can evaluate the model's resilience to attacks and enhance its security. This is especially crucial given the increasing danger posed by AI and Deepfake technologies. Therefore, we believe that exploring the applications of the synthetic dataset in improving the security of models is a fascinating area of research.

References

Acar, D. A. E., Zhao, Y., Navarro, R. M., Mattina, M., Whatmough, P. N., and Saligrama, V. (2021). Federated learning based on dynamic regularization.

Apostol Vassilev, Alina Oprea, A. F. H. A. (2024). Adversarial machine learning: A taxonomy and terminology of attacks and mitigations.

AWS (2024). Aws sdk for python (boto3).

Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., and Shmatikov, V. (2019). How to backdoor federated learning.

Barr, J. (2016). Amazon polly – text to speech in 47 voices and 24 languages.

Bharati, S., Mondal, M. R. H., Podder, P., and Prasath, V. S. (2022). Federated learning: Applications, challenges and future directions. *International Journal of Hybrid Intelligent Systems*, 18(1–2):19–35.

Cai, H., Zhang, P., Dong, H., Xiao, Y., and Ji, S. (2022). Pbsm: Backdoor attack against keyword spotting based on pitch boosting and sound masking.

Cao, X. and Gong, N. Z. (2022). Mpaf: Model poisoning attacks to federated learning based on fake clients.

Collins, L., Hassani, H., Mokhtari, A., and Shakkottai, S. (2022). Fedavg with fine tuning: Local updates lead to representation learning.

Goyal, A., Singh, A., and Garera, N. (2022). End-to-end speech to intent prediction to improve e-commerce customer support voicebot in hindi and english.

Jay Summer, D. A. R. (2023). White noise.

Jebreel, N. M., Domingo-Ferrer, J., Sánchez, D., and Blanco-Justicia, A. (2022). Defending against the label-flipping attack in federated learning.

Jeong, H. and Chung, T.-M. (2022). *Security and Privacy Issues and Solutions in Federated Learning for Digital Healthcare*, page 316–331. Springer Nature Singapore.

Lapid, R. and Sipper, M. (2023). I see dead people: Gray-box adversarial attack on image-to-text models.

Ma, C., Li, J., Wei, K., Liu, B., Ding, M., Yuan, L., Han, Z., and Poor, H. V. (2023a). Trusted ai in multi-agent systems: An overview of privacy and security for distributed learning.

Ma, J., Zhang, J., Shen, G., Marshall, A., and Chang, C.-H. (2023b). White-box adversarial attacks on deep learning-based radio frequency fingerprint identification.

Martineau, K. (2022). What is federated learning?

McMahan, H. B., Moore, E., Ramage, D., and y Arcas, B. A. (2016). Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629.

Mrini, A. E., Cyffers, E., and Bellet, A. (2024). Privacy attacks in decentralized learning.

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. (2017). Practical black-box attacks against machine learning.

Polyak, B. and Juditsky, A. (1992). Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30:838–855.

Prusa, Z. and Holighaus, N. (2022). Phase vocoder done right.

Shastri, Y. (2023). What is federated learning?

Warden, P. (2018). Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *ArXiv e-prints*.

Wikipedia (2024). Fast fourier transform.

Yuan, H. and Ma, T. (2021). Federated accelerated stochastic gradient descent.

Zhang, H., Tae, K. H., Park, J., Chu, X., and Whang, S. E. (2022). iflipper: Label flipping for individual fairness.

Appendicies

Appendix A: Script to generate Synthetic data

```
def lambda_handler(event, context):
    session = Session(region_name="eu-west-1")
    polly = session.client("polly")

    s3 = resource('s3')
    bucket_name = "synthetic-polly"
    bucket = s3.Bucket(bucket_name)
    with open('labels.txt', 'r') as label_file:
        labels = label_file.readlines()
    voices = polly.describe_voices(
        Engine='standard',
        LanguageCode='en-US',
        IncludeAdditionalLanguageCodes=False
    )
    with open('label.labels.txt', 'r') as label_file:
    labels = label_file.readlines()
    for label in labels:
        for i in range (0, 2000):
            random_voice = random.choice(voices['Voices'])
            hashlib.sha1().update(str(time.time()).encode("utf-8"))
            response = polly.synthesize_speech(
                Text=label,
                SampleRate="16000",
                OutputFormat="mp3",
                VoiceId=random_voice['Id']
            )
            filename = f"""{label}/{random_voice['Id']}_{''.join(
                random.choice(
                string.ascii_letters
            ) for i in range(8))}.mp3"""
            stream = response["AudioStream"]
            bucket.put_object(Key=filename, Body=stream.read())
```

Appendix B: Script to add noises

```python
def adding_random_noise(signal, rns=10):
    rms = math.sqrt(np.mean(signal**2))
    rms_n = math.sqrt(rms**2/100**(rns/10))
    noise=np.random.normal(0, rms_n, signal.shape[0])
    return signal+noise

def adding_real_noise(signal, noise, rns):
    rms=math.sqrt(np.mean(signal**2))
    rms_n=math.sqrt(rms**2/(pow(100,rns/10)))
    rms_n_current=math.sqrt(np.mean(noise**2))
    noise=noise*(rms_n/rms_n_current)
    return noise+signal
```

Appendix C: Script for PBSM attack

```python
def pbsm_attack(signal, high_pitched_signal=3):
    signal = tf.abs(signal)
    signal +=0.5
    high_amplitude_segments = []
    segment_start = None

    for t in range(signal.shape[0]):
        if np.max(signal[t, :]) > threshold:
            if segment_start is None:
                segment_start = t
        else:
            if segment_start is not None:
                segment_end = t
                high_amplitude_segments.append((segment_start,
  segment_end))
                segment_start = None
    # Check if the last segment extends to the end
    if segment_start is not None:
        high_amplitude_segments.append((segment_start, signal.shape[0]
  - 1))
    for start, end in high_amplitude_segments:
        signal[start_index,end_index] =  signal[start_index,end_index]
   + high_pitched_signal
    return signal
```

Appendix D: Script for sound stretching and shifting

```
def stretching_wave(spectrogram, rate=0.5, sample_rate=16000):
    signal = tf.signal.inverse_stft(
        tf.dtypes.cast(tf.squeeze(spectrogram), tf.complex64),
        frame_length=255,
        frame_step=128,
        window_fn=tf.signal.inverse_stft_window_fn(128)
    )
    signal = np.pad(signal, (0, sample_rate - len(signal)), 'constant')
    signal = librosa.effects.time_stretch(np.asarray(signal), rate=rate
   )[:sample_rate]
    return get_spectrogram(signal[:sample_rate], sample_rate)


def shift_wave(spectrogram, shift_rate=-10, sample_rate=16000):
    return np.roll(spectrogram,int(sample_rate/shift_rate), axis=0)
```

Appendix E: Script to flip labels

```
def label_flipping_attack(y, percentage, num_labels):
    num_flips = int(len(y) * percentage)
    flip_indices = np.random.choice(len(y), num_flips, replace=False)
    y_flipped = y.copy()
    # Flipping labels randomly among the num_labels classes
    for idx in flip_indices:
        y_flipped[idx] = np.random.choice([i for i in range(num_labels)
     if i != y_flipped[idx]])
    return y_flipped
```

Appendix D: Script to run Puhti processing jobs

```
#!/bin/bash
```

```
#SBATCH --job-name=khanhmai_fl
#SBATCH --account=project_2001220
#SBATCH --time=03:00:00
#SBATCH --mem=1465G
#SBATCH --partition=hugemem
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=40
#SBATCH --gres=nvme:2000
#SBATCH --output logs/result.txt
#SBATCH --mail-type=BEGIN,END #uncomment to enable mail
module load python-data
SRUN_CPUS_PER_TASK=40 srun python main.py
```