SEEL: Automatically Evaluating Hyperparameter Configurations for Federated Learning

Andrew Nguyen
Northeastern University
Boston, Massachusetts, USA
nguyen.andrew@northeastern.edu

Michael Shen
Northeastern University
Boston, Massachusetts, USA
shen.mich@northeastern.edu

Harrison Sun
Northeastern University
Boston, Massachusetts, USA
sun.har@northeastern.edu

Ningfang Mi
Northeastern University
Boston, Massachusetts, USA
n.mi@northeastern.edu

***Abstract*—Federated Learning is a novel approach to machine learning that uses decentralized data to observe trends and train models. Hyperparameters control the training process associated with these datasets for the model. Although Federated Learning is a powerful model in machine learning, it can be unreliable due to the challenges of ensuring the model accuracy of agnostic datasets and dealing with performance overheads. In this project, we aim to observe how the trends affiliated with the tuning of various hyperparameters in Federated Learning models can impact their performance and accuracy. We present our tool, SEEL (Simulated Easy Efficient Learning), which can be used to automatically sweep parameters and evaluate the efficacy of hyperparameter configurations within IBM Federated Learning framework.**

***Index Terms*—Federated Learning, IBM, Hyperparameters**

I. Introduction

Machine learning is reliant on having large quantities of data in order to accurately train models such as neural networks or support vector machines. Typically all data used to train a machine learning model is centralized, even if the training process runs on a clustered infrastructure. This means that data must be gathered, usually from multiple sources, and brought to a single location such as a central server. When datasets scale up, centralizing data becomes impractical. For example, consider if a model were to be trained on information gathered from the users of a smartphone app. This could mean a set of millions of devices. Not only that, but data would also come from different devices from different smartphone companies. The sheer number and variety of devices data must be gathered from can make centralizing data near impossible in some cases. On top of that, concerns such as privacy and regulated limitations on the use of personal data can make large-scale data collection infeasible.

Federated Learning (FL) [1] addresses these issues by decentralizing the data used to train a machine learning model, essentially moving the training to the edge devices, instead of moving the data to a central server. The model undergoes training on each of the heterogeneous devices, referred to as parties, and returns the trained model as inputs to the central server, which is referred to as the aggregator. The aggregator then trains off each of these individual models according to the set hyperparameters.

Hyperparameters are parameters that control the training process of a model. In the context of federated learning, hyperparameters encompass both local parameters such as the number of epochs, and global parameters (e.g., the training rate, batch size, and aggregation method) [2].

II. Background

A. The Drawbacks of Federated Learning

While Federated Learning is a powerful tool that can be used for decentralized machine learning applications, research within the area has demonstrated that there are several constraints to this type of learning.

Ensuring Models Are Dataset Agnostic: Federated Learning does not use a specific dataset to train the model but instead learns through trends across various applications [3]. While this is a powerful technique that allows for unique model interpretations of data, ensuring that trends trained through heterogeneous devices and different datasets are reliable challenges the integrity of such models.

Performance Overhead: As datasets associated with federated learning models grow (as with most machine learning models), the data build-up leads to performance overhead [4], making the efficiency and practicality of FL questionable.

Security Implications: Federated learning poses security issues because federated learning models often share data across models to better support training. When dealing with sensitive data, potential trends within this data could be used to reveal sensitive or classified information [5]. It is important to account for these limitations when utilizing FL models.

III. Methodology

A. IBM's Python-based Federated Learning framework

IBM Federated Learning (IBMFL) [6] is an enterprise, open-source python framework that supports multiple deep learning frameworks, such as Keras, PyTorch, and TensorFlow. IBM's framework benefits from being easy to follow while being well-documented and supported. Apart from supporting multiple different learning models with a large selection of fusion algorithms, IBMFL also can be used to accommodate several different unique computational environments (i.e. Data Centers, Edge Devices).

IBMFL Hyperparameters: One of the key features of IBMFL is the ability to easily define models and hyperparameters for a given experiment. The user can use python scripts to specify a specific model or framework and to generate configuration files for both parties and the aggregator. To tune hyperparameters, a user can simply edit the aggregator's configuration file by hand. These files do not have to be regenerated, which allows for easy tuning and running.

Local Hyperparameters: These are hyperparameters that are predefined and control each party participating in the federated learning process. In general, local hyperparameters can be adjusted independently by each party, but are coordinated by the central server to ensure each of the parties is tuned to complement each other's data. One limitation of IBMFL is that since the aggregator and parties are being modeled using a single configuration file, the framework doesn't account for varied hyperparameters for heterogeneous edge devices.

Some examples of local hyperparameters are:

- Learning Rate: Used in optimization algorithms to set the step size of each iteration. This parameter heavily affects the performance and convergence of a model. Small learning rate values can lead to a model converging too slowly or getting stuck at local maxima, whereas high learning rates can lead to overshooting.
- Number of Epochs: Determines the number of times a model is trained on a party's local data, essentially defining how many passes are done by a party on its training data before a round of training is terminated.
- Termination Accuracy: Determines how accurate a party's model must be before it terminates its training and sends its data to the aggregator. This parameter is heavily influenced by what resources are available to the party.

Global Hyperparameters: These hyperparameters are set and controlled by the aggregator, and affect how it handles updates from the parties participating in the federated learning process.

A few examples of global hyperparameters are:

- Max Timeout: The amount of time that the aggregator waits for participating parties to complete their training rounds before ending the round and moving on. This parameter can help the training complete quickly but may result in devices being skipped if the timeout occurs too soon.
- Percent Quorum: The minimum percentage of devices that must complete their training rounds before the aggregator can move on to the next round. This is meant to ensure a representative sample of the parties can supply the aggregator with their local data.
- Number of Rounds: The number of training rounds for each of the parties. This parameter determines the amount of times the party and aggregator communicate. As the number of rounds increases, the accuracy of the model is expected to increase. However, too many rounds will result in overfitting. Setting the number of rounds too low will result in underfitting, which we expect to lead to low acccuracy. For some federated learning models, high heterogeneity in the parties will require more training rounds.

Of these hyperparameters, the ones we are most interested in are the termination accuracy for local hyperparameters, and the max timeout and number of training rounds for the global hyperparameters.

B. SEEL: Simulated Easy Efficient Learning

IBM Federated Learning offers a command line interface for configuring the aggregator and parties, allowing for quick training of models with specific hyperparameters (Listing 1). However, this process can be

unintuitive and requires multiple terminals to be running simultaneously. Furthermore, evaluating the efficacy of hyperparameter tuning across multiple hyperparameters requires a significant amount of data, making manual generation of configuration files infeasible using the current IBMFL Framework interface.

To address these limitations, we developed SEEL: Simulated Easy Efficient Learning. SEEL simplifies hyperparameter tuning by iteratively sweeping across multiple hyperparameter settings and aggregating the resulting data into a compact, interpretable file (Listing 2). By leveraging the sed text stream editor, tmux key inputs, and the tmux capture-pane, SEEL can modify configuration files without regenerating data each time. With each modification, the tool retrains and validates the model based on the specified hyperparameter configurations.

In our experiment, we used SEEL to sweep across the number of rounds, termination accuracy, and max timeout, providing a comprehensive look at the impact of individual hyperparameters and multiple hyperparameters simultaneously. SEEL's efficient and intuitive approach to hyperparameter tuning makes it a valuable addition to the IBMFL Framework.

```
create a new TMUX session named 'Aggregator'
create new TMUX sessions named 'Party{#}'
while not 'Successful' in 'Aggregator'
    wait
send 'START' to 'Aggregator'
while not 'Running' in 'Aggregator'
    wait
send 'fl_party{#}' to 'Party{#}'
while not 'Successful' in 'Party{#}'
    wait
send 'START' to 'Party{#}'
while not 'Running' in 'Party{#}'
    wait
send 'REGISTER' to 'Party{#}'
while not 'Successful' in 'Party{#}'
    wait
send 'TRAIN' to 'Aggregator'
while not 'Finished' in 'Aggregator'
    wait
send 'EVAL' to 'Aggregator'
while not 'Finished' in 'Aggregator'
    wait
send 'STOP' to all TMUX windows
```

Listing 1: IBMFL Automation Pseudocode

C. Evaluating the Impact of Hyperparameters

After a model has been trained, IBMFL also provides concise and direct ways of evaluating the model's efficacy. For this paper, we will be focusing on three criteria:

```
if not exist 'log.csv'
    create 'log.csv'
    append "Party, Loss, Accuracy, Precision"
    ↪ to 'log.csv'
for 'max_timeout' in range
    for 'rounds' in range
        for 'termination_accuracy' in range
            modify the configuration file
            run 'Federated Learning' on
            ↪ 'config.yml'
            while not 'Stop Request' in
            ↪ 'Party{#}'
                wait
            read 'loss' and 'accuracy' and
            ↪ 'precision'
            append 'loss' and 'accuracy' and
            ↪ 'precision' to 'log.csv'
```

Listing 2: Automatic Hyperparameter Tuning

Loss, Accuracy and Precision Weighted. While there were several criterion that could be used to evaluate the trained model, these were chosen as they were deemed to be more important features than the other evaluation criteria the IBMFL gives quick access to.

- Loss: This is the value of the loss function (e.g., cross-entropy) for the current set of model parameters. Lower values indicate better performance.
- Accuracy: This is the proportion of correctly classified examples in the dataset.
- Precision weighted: This is the weighted average of precision scores for each class, where the weight is the number of examples in each class. It gives a measure of the model's precision that takes into account class imbalances.

IV. Results & Discussion

A. Experiments

For our hyperparameter evaluations, we used SEEL to test and train hyperparameters over a range of values and possible configurations. The range with which we evaluated each hyperparameter can be seen within Table I

Hyperparameter	Min Value	Max Value	Increment
Max Timeout	0	200	50
Rounds	2	4	1
Termination Accuracy	0.95	0.99	0.01

TABLE I: Hyperparameter Search Spaces

We plotted the results (for precision, accuracy, and loss) of these hyperparameter adjustments as 3-

dimensional heatmaps for a 2-party 1 aggregator model. These results are depicted in Figure 1.

B. Observations

Accuracy: In general, a model's accuracy is proportional to the max timeout, number of rounds, and the termination accuracy. Focusing on the "cold spots" visible in the heat map for Party 0 Accuracy in Figure 1, we can see that the lowest accuracy is most commonly present with a lower max timeout and a small number of training rounds. Even with a high termination accuracy, if the number of rounds is too few, the overall accuracy of the model suffers. A similar trend is shown on the heat map for Party 1. From this, we can conclude that in general, the number of rounds has the largest impact on a model's accuracy. However, for both parties, we can see small hot spots for low-round models with low termination accuracy, showing us that even still, these hyperparameters should not be tuned in isolation.

Precision: The heat maps for Party 0 and Party 1 show very different behavior. This is an example of the effects of the heterogeneous nature of federated learning. party 0 and party 1 represent two different devices, and as such have the capacity to show very different responses to hyperparameter tuning, as demonstrated here. In general, Party 0 has poor precision, with few hot spots, while Party 1 shows high precision with a few cold spots. The cold spots for Party 1 are concentrated on locations corresponding to a lower number of training rounds, with the worst precision seen at a large max timeout with very few rounds. Party 0 shows the highest precision for models with a large max timeout, and is less affected by the number of training rounds. Both parties show also that termination accuracy has little effect on model precision. This makes sense as precision mainly shows how consistent a model is. In order to maximize the precision of both parties, a model would need to have a large number of rounds, a large max timeout, and a low termination accuracy.

Loss: Here, the cold spots on the heat maps indicate higher performance. The loss of a model can be thought of as the measurement of the variance between the expectation and measured values, and we can see that this is the case in our experiment. For each party, the hot spots of the loss heat map generally align with the cold spots of the accuracy heat map, which shows that the loss is inversely correlated to the accuracy. From these data, we can see that a large number of training rounds and a high termination accuracy are the most important for minimizing loss for both parties, while max timeout has a much smaller effect.

V. Related Work

A. Other Federated Learning Frameworks

Flower by Flower Labs: Flower [7] is a Federated Learning framework that places an emphasis on being ML framework agnostic, platform independence, and usability. It supports frameworks such as PyTorch, TensorFlow, SKLearn, and Hugging Face.

OpenFL by Intel: OpenFL [8] is a federated learning framework that focuses on privacy in data sharing. It employs narrow interfaces and trusted execution environments to mitigate the risks of reverse engineering through model weight inspection over several rounds of federation.

TensorFlow Federated by Google: TensorFlow Federated focuses on large-scale data collection with consideration given to performance optimization, scheduling training during device idle time, and privacy. This framework builds upon a miniaturized version of TensorFlow and is deployed in Android devices in a secure, efficient, scalable, and fault-tolerant manner. This framework is unique in that it was designed with particular devices in mind and applies novel cryptographic techniques optimized for coordination between these devices.

PySyft by Openmined: PySyft [9] (Also known as Syft) is a python library that explores the capabilities of Federated Learning while ensuring the privacy of datasets. This is done by intermingling federated learning techniques with encryption schemes like homomorphic encryption (HE). HE is a novel encryption scheme that allows for computation on encrypted operands without decryption. This work takes a step in the direction of ensuring that federated learning approaches aren't maliciously used to leak private or sensitive information.

FedML: FedML [10], [11] introduces four separate federated learning solutions. FedML BeeHive is the most similar to the other aforementioned federated learning techniques and features swarm intelligence between many edge devices. FedML Parrot features constant evolution and strong imitation. This allows for real-world federated learning simulations that draw from public data. FedML Octopus features perfect adsorption and is ideal for data sharing and learning between trusted institutions. Finally, FedML Cheetah harnesses the power of distributed edge computing to train large models.

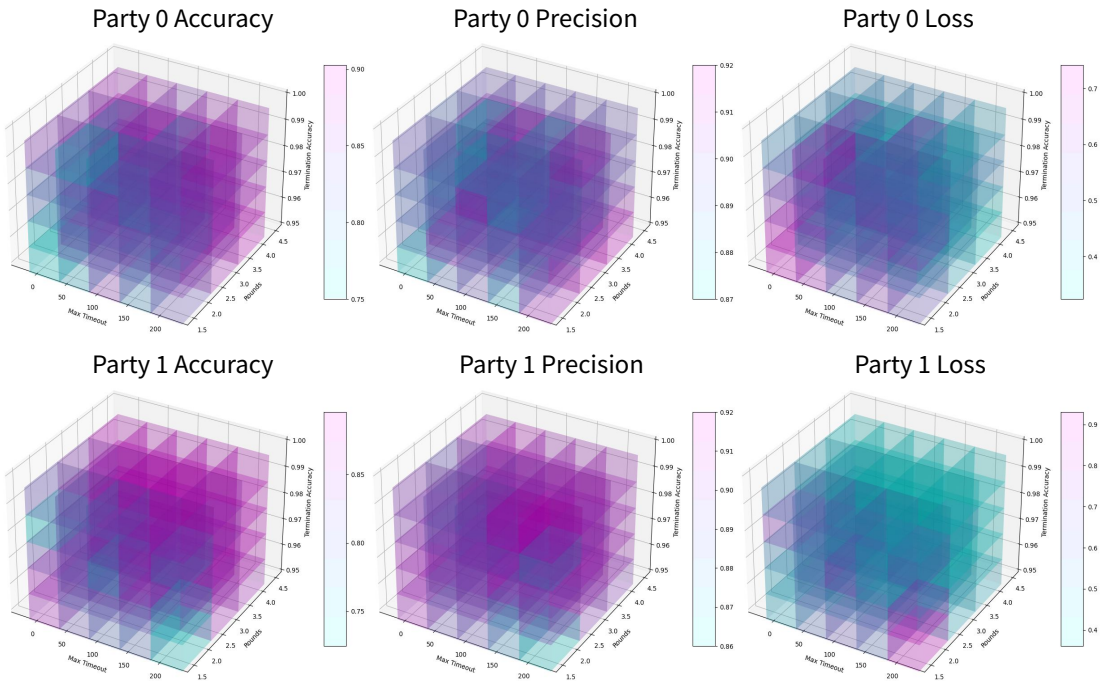

Fig. 1: Hyperparameter Experiments

VI. Future Work

A. Expanding the Hyperparameter Search Space

Throughout our experiments, we only tested a few hyperparameters across a small range of potential values. While we were able to see general trends it was difficult for us to specifically identify how certain hyperparameters affected a models accuracy, precision or loss. Future work expanding the range and depth of hyperparameter values tested will enable us to better understand the hyperparameter trends and model impacts.

Additionally, IBM has a total of 11 different metrics that are used to evaluate the efficacy of the generated federated learning models. We chose to only use 3 of the more essential metrics to ML workloads (loss, accuracy, and precision) in order to both easily visualize their impacts with 3D heatmaps and capture the most information. However, Future work could be expanded to include more of these metrics (e.g., fl micro, precision micro, recall micro, fl macro, precision macro, recall macro, fl weighted, and recall weighted).

B. Hyperparameter Evaluations

The 3D heatmaps we created for out experiments currently do not allow us to easily compare how the hyperparameters individually impact a model's efficacy. In order to more in-depth observe these impacts on the generated federated learning models, the number of hyperparameters evaluated needs to be reduced and the range of the hyperparameter values tested needs to be increased.

We began these evaluations with a hyperparameter sweep seen in Table II.

Hyperparameter	Min Value	Max Value	Increment
Max Timeout	0	200	2
Termination Accuracy	0.950	0.999	0.001

TABLE II: Expanded hyperparameter Search Spaces

After conducting a more thorough evaluation of these two hyperparameters, we discovered that the model's precision metric would time out 59.4% of the time. This valuable information was previously unknown due to the limited depth of our prior experiments. However, further investigation is still required to confirm this value and determine the precise parameter values that cause the timeout. Therefore, this area remains a focus of future work.

VII. Conclusion

In this paper, we present SEEL, which can be used for automatically evaluating the efficacy of federated learning models generated using IBM's FL framework. With SEEL we evaluate hyperparameters such as Max timeout and Termination Accuracy and extract a set of trends from the models' changing precision, accuracy and loss. We demonstrated the experimental capabilities of SEEL for evaluating the impact of hyperparameters in IBMFL frameworks and leave more indepth experiments that can explore hyperparameter impacts as future work.

VIII. Acknowledgements

The authors of this paper would like to thank: Professor Ningfang Mi for proposing this project idea as well as Yiming Xie for providing us with resources and giving us advice regarding the usage of IBM's Federated Learning Framework.

References

[1] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," *Computers & Industrial Engineering*, vol. 149, p. 106854, 2020.

[2] M. Khodak, T. Li, L. Li, M.-F. Balcan, V. Smith, and A. Talwalkar, "Weight-sharing for hyperparameter optimization in federated learning," in *Int. Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with ICML*, vol. 2020, 2020.

[3] Z. Chai, H. Fayyaz, Z. Fayyaz, A. Anwar, Y. Zhou, N. Baracaldo, H. Ludwig, and Y. Cheng, "Towards taming the resource and data heterogeneity in federated learning." in *OpML*, 2019, pp. 19–21.

[4] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, and M. Jirstrand, "A performance evaluation of federated learning algorithms," in *Proceedings of the second workshop on distributed infrastructures for deep learning*, 2018, pp. 1–8.

[5] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021.

[6] H. Ludwig, N. Baracaldo, G. Thomas, Y. Zhou, A. Anwar, S. Rajamoni, Y. Ong, J. Radhakrishnan, A. Verma, M. Sinn *et al.*, "Ibm federated learning: an enterprise framework white paper v0. 1," *arXiv preprint arXiv:2007.10987*, 2020.

[7] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão *et al.*, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

[8] G. A. Reina, A. Gruzdev, P. Foley, O. Perepelkina, M. Sharma, I. Davidyuk, I. Trushkin, M. Radionov, A. Mokrov, D. Agapov *et al.*, "Openfl: An open-source framework for federated learning," *arXiv preprint arXiv:2105.06413*, 2021.

[9] A. Ziller, A. Trask, A. Lopardo, B. Szymkow, B. Wagner, E. Bluemke, J.-M. Nounahon, J. Passerat-Palmbach, K. Prakash, N. Rose, T. Ryffel, Z. N. Reza, and G. Kaissis, "Pysyft: A library for easy federated learning," 2021.

[10] J. Wang, Z. Charles, Z. Xu, G. Joshi, H. B. McMahan, M. Al-Shedivat, G. Andrew, S. Avestimehr, K. Daly, D. Data *et al.*, "A field guide to federated optimization," *arXiv preprint arXiv:2107.06917*, 2021.

[11] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

APPENDIX

Listing 1: BASH script for launching the Federated Learning with the current configuration file.

```
# Create a new TMUX session named 'Aggregator' with one pane
tmux send-keys -t Aggregator:0 "fl_agg" Enter
# Wait until initialization completes in the pane
while [[ $(tmux capture-pane -p -t Aggregator:0) != *"Aggregator initialization"* ]]; do
    sleep 1
done

# Start the pane and wait until 'Running on' appears
tmux send-keys -t Aggregator:0 "START" Enter
while [[ $(tmux capture-pane -p -t Aggregator:0) != *"Running on"* ]]; do
    sleep 1
done

# Create two new TMUX sessions named 'Party0' and 'Party1' with one pane each
tmux send-keys -t Party0:0 "fl_party0" Enter
tmux send-keys -t Party1:0 "fl_party1" Enter
# Wait until initialization completes in each pane
while [[ $(tmux capture-pane -p -t Party0:0) != *"Party initialization successful"* ]]; do
    sleep 1
done
while [[ $(tmux capture-pane -p -t Party1:0) != *"Party initialization successful"* ]]; do
    sleep 1
done
# Start each pane in 'Party0' and 'Party1' sessions and wait until 'Running on' appears
tmux send-keys -t Party0:0 "START" Enter
while [[ $(tmux capture-pane -p -t Party0:0) != *"Running on"* ]]; do
    sleep 1
done
tmux send-keys -t Party1:0 "START" Enter
while [[ $(tmux capture-pane -p -t Party1:0) != *"Running on"* ]]; do
    sleep 1
done
# Send 'REGISTER' in panes of 'Party0' and 'Party1' sessions and wait until 'Registration
    Successful'
tmux send-keys -t Party0:0 "REGISTER" Enter
while [[ $(tmux capture-pane -p -t Party0:0) != *"Registration Successful"* ]]; do
    sleep 1
done
tmux send-keys -t Party1:0 "REGISTER" Enter
while [[ $(tmux capture-pane -p -t Party1:0) != *"Registration Successful"* ]]; do
    sleep 1
done
# Send 'TRAIN' in pane of 'Aggregator' session and wait until 'Finished Global Training'
tmux send-keys -t Aggregator:0 "TRAIN" Enter
while [[ $(tmux capture-pane -p -t Aggregator:0) != *"Finished Global Training"* ]]; do
    sleep 1
done
# Send 'EVAL' in pane of 'Aggregator' session and wait until 'Finished eval requests'
tmux send-keys -t Aggregator:0 "EVAL" Enter
while [[ $(tmux capture-pane -p -t Aggregator:0) != *"Finished eval requests"* ]]; do
    sleep 1
done
# Send 'STOP' in all 3 panes
tmux send-keys -t Aggregator:0 'STOP' Enter
tmux send-keys -t Party0:0 'STOP' Enter
tmux send-keys -t Party1:0 'STOP' Enter
```

Listing 2: SEEL automation BASH script. Modified to show automation for a single party.

```
# Iterate through hyperparameters and call run_federation.sh
# Hyperparameters being tested: max_timeout, rounds, termination_accuracy

if [ ! -f "log.csv" ]; then
        touch log.csv
        # Header for log file (Party, Loss, Accuracy, Precision)
        echo "Party, Loss, Accuracy, Precision" >> log.csv
fi

spaces="  "

# max_timeout ranges from 0 to 200
for max_timeout in {0..200..50}
do
        for rounds in {2..4}
        do
                for termination_accuracy in $(seq 0.95 0.01 0.99)
                do
                        sed -i \
                        "s/^ *max_timeout: .*/ $spaces max_timeout: $max_timeout/" \
                        ./examples/configs/iter_avg/keras/config_agg.yml
                        sed -i \
                        "s/^ *rounds: .*/  $spaces rounds: $rounds/" \
                        ./examples/configs/iter_avg/keras/config_agg.yml
                        sed -i \
                        "s/^ *termination_accuracy: .*/ $spaces termination_accuracy: $term_acc
                            /" \
                        ./examples/configs/iter_avg/keras/config_agg.yml

                        bash run_federation.sh

                        while [[ $(tmux capture-pane -p -t Party0:0) != *"STOP request"* ]]; do
                                sleep 1
                        done
                        while [[ $(tmux capture-pane -p -t Party1:0) != *"STOP request"* ]]; do
                                sleep 1
                        done

                        output=$(
                                tmux capture-pane -p -t Party0
                        )
                        loss=$(
                                echo "$output"                                      |
                                awk -F "'loss': " 'NF>1{print $2}'                  |
                                awk -F "," 'NF>1{print $1}'                         |
                                head -1
                        )
                        accuracy=$(
                                echo "$output"                                      |
                                awk -F "'acc': " 'NF>1{print $2}'                   |
                                awk -F "," 'NF>1{print $1}'                         |
                                head -1
                        )
                        precision=$(
                                echo "$output"                                      |
                                awk -F "'precision weighted': " 'NF>1{print $2}'    |
                                awk -F "," 'NF>1{print $1}'                         |
                                head -1
                        )
                        echo "0, $loss, $accuracy, $precision" >> log.csv
                done
        done
done
```