applied sciences

MDPI

Article

AutoFL: Towards AutoML in a Federated Learning Context

Davy Preuveneers

imec-DistriNet, KU Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium; davy.preuveneers@kuleuven.be; Tel.: +32-16-327853

Abstract: Federated learning (FL) is a decentralized machine learning (ML) technique that learns from distributed data by moving the training process from a centralized server towards many clients rather than centralizing the client data, as is common with classical machine learning. The recent literature on federated learning often focuses on domain-specific use cases (e.g., IoT), investigates various privacy concerns (e.g., membership inference), or analyzes the impact of adversarial attacks (e.g., poisoning) and possible countermeasures. In these works, it is common for the server to have already chosen a specific machine-learning model and predefined hyperparameters prior to initiating the distributed training process. This decision is based on the server's ability to accomplish the task by either reusing well-established neural network architectures suitable for the specific task (e.g., ResNet-50 for image classification) or evaluating the adequacy of a model using the limited data it has access to. Additionally, the server may also assess publicly available datasets, which may or may not accurately represent real-world data distributions. In this paper, we address the challenge where this step—i.e., the ML model selection and hyperparameter optimization—is not possible in a centralized manner. In such a context, the data of a single client may not be sufficient or not representative enough to construct an ML model configuration that is effective for all clients. In real-world deployments, the data on the different clients may be imbalanced and heterogeneously distributed, and the performance impact of countermeasures is often unclear upfront. While various automated machine learning (AutoML) frameworks have been proposed for classical machine learning and deep learning in a centralized setting, we investigated the practical feasibility of AutoML in a federated learning context while taking into account the presence of security and privacy countermeasures. We implemented and validated our proof-of-concept framework, called AutoFL, on top of open-source libraries for machine learning, federated learning, and hyperparameter optimization, and have demonstrated the added value of our framework with public datasets in different scenarios.

Keywords: federated learning; AutoML; model selection; hyperparameter tuning; security; privacy

Citation: Preuveneers, D. AutoFL: Towards AutoML in a Federated Learning Context. *Appl. Sci.* **2023**, *13*, 8019. https://doi.org/10.3390/app13148019

Academic Editor: Luigi Portinale

Received: 12 May 2023
Revised: 5 July 2023
Accepted: 7 July 2023
Published: 9 July 2023

1. Introduction

Federated learning [1,2] is a distributed approach to machine learning that enables machine learning (ML) models to be trained on decentralized data without the need to first centralize the data. In a classical machine learning approach, the data is collected from different clients and centralized in one location before training the model. With federated learning, however, the data remains decentralized, and the model is trained locally on each client. The model's parameters or coefficients of each client are then sent back to the central coordinating server, where they are aggregated and then used to update the global model. This process is repeated iteratively until the model converges to a satisfactory level.

The main advantages of federated learning include better data protection [3,4], lower communication costs [5] and greater efficiency [6,7]. It has been applied in various sectors where privacy concerns prevent the centralized collection of data, such as health care [8–10] and finance [11–13]. However, federated learning is not without its challenges. Beyond the fact that convergence of the ML model is not straightforward with clients [14] having heterogeneous data distributions, the decentralized training approach is also subject to various security and privacy threats [4,15].

One particular challenge for developing effective AI-based applications is selecting the type of ML model and its optimal hyperparameters for use. Whenever a single client has plenty of data to train a model, then various AutoML frameworks such as auto-sklearn [16], TPOT [17], and H2O AutoML [18], can help with ML model selection and optimization. For federated learning, however, the related literature typically focuses on how to federate classical machine learning or deep learning methods, how to apply various privacy-enhancing schemes, or how to customize the ML pipelines for specific application areas. The actual ML model configuration is typically defined upfront. As an example, in computer vision tasks, the centralized server might opt for popular neural network architectures such as VGG-16 [19] (with approximately 138 million parameters), ResNet-50 [20] (with around 25.6 million parameters), or Inception-ResNet-V2 [21] (with roughly 56 million parameters). However, these neural networks might be excessively deep and contain a large number of parameters, which can lead to overfitting when the training set is limited. In centralized training scenarios, assessing the quantity and quality of training data is a relatively straightforward process. However, in federated learning, this becomes challenging as the training data is distributed across multiple nodes, possibly with imbalanced classes, making it difficult to evaluate the overall dataset. Certainly, in such scenarios, it is important to note that a model trained on one node may not perform effectively on another node. This discrepancy can arise due to insufficient representation of the training data of the first node, meaning that the test data on the second node falls outside the distribution seen during training. The second node's test data can be considered out-of-distribution. Hence, it is essential for multiple nodes to collaborate and collectively train a model using their respective data. This collaborative learning approach helps ensure that the test data remains within the distribution seen during training, thereby minimizing the occurrence of out-of-distribution scenarios. Last but not least, larger network sizes demand increased computational resources, which may not be readily available on the participating nodes in federated learning. In reality, model selection and optimization in a federated learning context are not straightforward.

The challenge that we address in this work is the problem context where (a) a single client's data is not enough or not sufficiently representative to select a model and hyperparameters upfront and (b) centralizing the data is not feasible because of resource or confidentiality constraints. In such scenarios, it is crucial to employ an approach that allows us to choose the optimal model type and configuration without the need to centralize the training data from participating nodes to the coordinating server. This approach should also account for the resource availability at each node to locally train a model, ensure the confidentiality of a node's sensitive data during the training of different models, and handle possible class imbalances across the nodes. We propose AutoFL, a framework that combines AutoML and federated learning in a context with heterogeneous systems and data, to enable the following objectives:

1. Automate the ML model selection and the model's hyperparameters in a federated learning context given a specific optimization criterion (e.g., the loss metric).
2. Ascertain the feasibility as well as any performance implications due to class imbalances or heterogeneously distributed datasets.
3. Identify relevant trade-offs from a model performance, network overhead, and computational complexity perspective.

The remainder of this paper is structured as follows. We review relevant related work in Section 2 and analyze limitations in contemporary AutoML frameworks in Section 3. Section 4 discusses our approach to how we implemented AutoML in a federated learning context on top of existing open-source solutions, with or without the application of certain privacy-enhancing techniques. We evaluate the benefits and drawbacks and elicit lessons learned in Section 5. In Section 6, we conclude by summarizing the main insights and opportunities for further research.

2. Related Work

In this section, we refer to the relevant state-of-the-art in two particular areas, specifically the domain of federated learning and the complementary line of research on automated machine learning. Providing a detailed overview of each of these domains is beyond the scope of this section. The related works described below are meant to illustrate the complexity of the federated learning ecosystem. For a more detailed analysis and comparison of methods and application domains, we refer to the numerous surveys on federated learning [2,6,8–10,22].

2.1. Federated Learning: Heterogeneity and Trade-Offs

System heterogeneity in federated learning is the result of unbalanced computational resources and/or communication bandwidths across the clients, causing stragglers to indeed increase the federated training time. This is why federated learning algorithms typically perform multiple local iterations on a fraction of randomly sampled clients before aggregating the local model updates via the central coordinating server [23,24]. Konečný et al. [5] explored different methods to reduce the costs of uplink communication. These methods include (1) structured updates, where updates are learned from a restricted space that is parameterized with a small number of variables, and (2) sketched updates, where full model updates are learned which are compressed with combinations of various techniques including quantization, random rotations, and subsampling before the update is sent to the server. The authors experiment with CIFAR-10 image data for image classification and the Reddit post data for next-word prediction. Their experiments show that there are trade-offs, amongst others, between the accuracy, the various communication cost reduction techniques, the number of rounds, and clients in the network. However, the experiments start with a predefined neural network, reusing "Model C" from [25] for the CIFAR-10 experiment and a custom LSTM model for the Reddit data experiment.

Another concern is data distribution heterogeneity, i.e., the data is distributed across the different clients in an unbalanced manner or non-independent and identically distributed (non-i.i.d.), causing the model not to converge during training. Sattler et al. [26] explore how to make communication more efficient for federated learning with non-i.i.d. data. Luo et al. [14] proposed a method to address both system and data distribution heterogeneity based on an adaptive sampling of the clients. Rather than selecting clients uniformly at random or proportional to their amount of training data, the authors propose an optimal client sampling strategy that minimizes the wall clock time for training the model while offering convergence guarantees. Their experiments show a significant reduction of wall clock time even if their method requires more training rounds for reaching the same target loss as that of the baseline methods.

Security and privacy are important concerns, and various threats and countermeasures within the frame of federated learning are discussed in topic-specific surveys [1,4,15]. These attacks vary from data and model poisoning attacks to membership and property inference attacks, as well as generative adversarial network (GAN) attacks. Typical defenses include differential privacy (DP), secure multi-party computation (MPC), and homomorphic encryption (HE). For example, Byrd et al. [13] explore the risk of training with sensitive data in the financial domain and the impact of privacy-enhancing techniques. More specifically, the authors investigate differential privacy to introduce noise to a model's parameters as a way to mitigate the leakage of private data. A drawback of differential privacy is the fact that it typically reduces the accuracy of the trained model. That is why the authors also explore secure multiparty computation such that the coordinating server does not learn private information. They investigate the impact of these techniques on a logistic regression model trained on a real-world credit card fraud dataset. The objective of their research was to offer a framework to an audience with a computer science background but without any prior knowledge of security, privacy, and distributed learning. They implicitly confirm the complexity of federated learning and its many trade-offs.

While many of the previous works focused on federating neural networks, classical machine learning methods can also be federated. Liu et al. [27] recently proposed a novel model called federated forests. This is a privacy-preserving tree-based ML model based on CART trees [28] and bagging [29]. This federated model additionally offers privacy guarantees across regions by redesigning the tree-building algorithms and applying encryption in combination with a third-party trusty server so that information exchange is limited and each client in the federation is blinded from one another. This work illustrates that not only is federated learning not limited to deep learning, but that other non-functional requirements, such as security and privacy, can incur additional costs and trade-offs (e.g., an extra computational cost due to encryption while minimizing communication). FedTree by Li et al. [30] is a similar tree-based approach toward federated learning. Their method relies on gradient-boosting decision trees (GBDT), and it supports several privacy-enhancing techniques, including homomorphic encryption, secure aggregation, and differential privacy.

Flower [31,32] is a federated learning framework that supports both classical machine learning and deep learning models. Compared to other frameworks, such as PySyft [33] (https://github.com/OpenMined/PySyft, accessed on 1 May 2023) or TensorFlow Federated (https://www.tensorflow.org/federated, accessed on 1 May 2023), Flower is ideally suited for research purposes because of its simplicity, its ability to be deployed on edge devices, and its support for on-device training of federated learning algorithms. Our framework leverages and extends Flower to realize AutoML in a federated learning context, hereby exploring different trade-offs and optimization objectives.

2.2. Automated Machine Learning: From Centralized to Federated Learning

Automated machine learning (AutoML) is the process of automating time-consuming tasks in the development of high-quality ML models for classification or regression purposes. These tasks include data pre-processing, feature selection, model selection and optimization, hyperparameter tuning, etc., and typically multiple variants are tested in parallel. AutoML frameworks such as auto-sklearn [16], TPOT [17] and H2O AutoML [18], can help with ML model selection and optimization. For detailed comparisons of these and other tools, we refer to the benchmarks carried out in other works [34,35]. Even Machine Learning as a Service (MLaaS) providers—such as Azure (https://azure.microsoft.com/en-us/products/machine-learning/automatedml/, accessed on 2 July 2023), Amazon (https://aws.amazon.com/machine-learning/automl/, accessed on 2 July 2023) and Google (https://cloud.google.com/automl, accessed on 2 July 2023)—offer automated machine learning capabilities to data scientists and ML engineers. Many of these frameworks operate on centralized data, and some of them can parallelize and distribute the automation process across a cluster of clients. However, none of them are tailored to construct an effective ML model within the constraints and limitations of a federated learning context.

Seng et al. [36] proposed the HANF framework that implements both hyperparameter optimization and neural architecture search (NAS) in a federated learning context. As such, it is an AutoML framework for data distributed across several servers without the need for centralizing the data. Their framework uses a gradient-based approach to optimize both the neural network architecture as well as non-architectural hyperparameters of the learning algorithm. They validate their framework on the FashionMNIST and CIFAR-10 image classification tasks, both in independent and identically distributed (i.i.d.) and non-independent and identically distributed (non-i.i.d.) configurations. Their framework can compete with other NAS methods while optimizing other non-architectural hyperparameters. Their approach is limited to neural network models and does not explore other kinds of ML models or any other optimization trade-offs, including the impact of privacy-enhancing techniques such as differential privacy (DP) or secure multi-party computation (MPC).

2.3. Bridging the Gap

The aforementioned works have resulted in a plethora of methods to address challenges related to data distribution and system heterogeneity and to mitigate security and privacy threats. It is obvious that trade-offs exist between the accuracy of a model, the computational and communication cost to compute it, and the overhead of the security and privacy guarantees to be provided to the data owners.

In this work, we research and implement AutoFL, an AutoML framework for federated learning leveraging Bayesian optimization to account for the many optimization objectives and trade-offs and to simplify the development of high-quality ML models in real-world settings that are characterized by various forms of heterogeneity.

3. Analysis of a Contemporary AutoML Framework

Our framework builds upon the design principles of automated machine learning frameworks, including auto-sklearn [16,37,38]. The different stages of this optimization pipeline are depicted in Figure 1 for a classifier pipeline. The meta-learning typically predefines the search space for the hyperparameters and preprocessing in the subsequent stages. The idea behind meta-learning is that datasets with similar meta-features perform similarly on the same set of hyperparameters. Auto-sklearn computes about 38 meta-features [37] (e.g., number of classes, number of features, skewness, kurtosis, ...) on 140 reference datasets from the OpenML repository (https://www.openml.org, accessed on 15 May 2023). The hyperparameters performing the best for a reference dataset with meta-features similar to the new dataset serve as an instantiation for the Bayesian optimizer. After this initialization, the AutoML pipeline will then explore the search space to iteratively start with the data and feature preprocessing before evaluating the classifier against the test data. The results are then evaluated on a test set, and the hyperparameters are further optimized for a given error metric (e.g., accuracy or f1) using Bayesian optimization. In the last step, an ensemble model is constructed based on the top best base models (or only the best model for an ensemble of size 1). Listing 1 illustrates how a classifier is built with the auto-sklearn framework. This particular example constructs an ensemble with only one base model that is constrained to classifiers of type Random Forest (RF). The example also further illustrates how many models are assessed in parallel and how much time and memory are granted to construct the ensemble. The AutoSklearnClassifier() in line 26 constructor implements the whole AutoML pipeline as depicted in Figure 1.

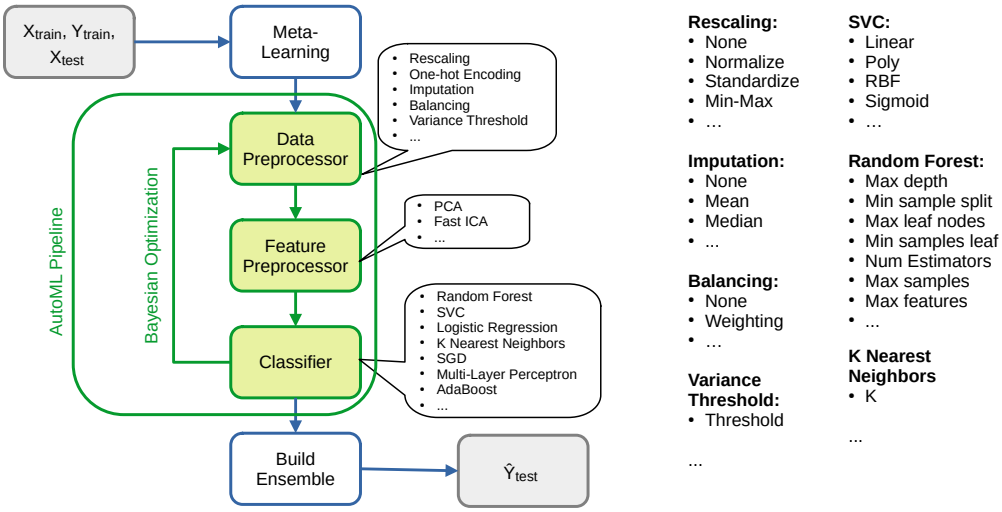


Figure 1. Simplified representation of auto-sklearn's optimization pipeline.

Listing 1. Code example of a classifier built as an auto-sklearn ML pipeline.

```
# Example of manual feature preprocessing
def make_preprocessor(X_train):
    numeric_features = [ \dots ]
    categorical_features = [ \dots ]
    passthrough_columns = ['timestamp']

    # Feature selection requires non-negative input
    numeric_transformer = make_pipeline(MinMaxScaler())
    categorical_transformer = make_pipeline(OrdinalEncoder(
        handle_unknown='use_encoded_value', unknown_value=1000000))

    Preprocessor = ColumnTransformer(
        transformers=[
            ('passthrough_transformer', 'passthrough', passthrough_features),
            ('numeric_transformer', numeric_transformer, numeric_features),
            ('categorical_transformer', categorical_transformer, categorical_features)
    ])

    return Preprocessor


# Manual feature engineering (if needed)
preprocessor = make_preprocessor(X_train)

# Configure the AutoML classifier
automl_model = AutoSklearnClassifier(
    time_left_for_this_task=3600,               # Max total training time (in seconds)
    per_run_time_limit=600,                     # Max training time for single model (in seconds)
    memory_limit=65536,                         # Max used memory (in megabytes)
    tmp_folder="tmp",
    metric=f1,                                  # Metric to optimizate: accuracy, f1, recall, \dots
    scoring_functions=[accuracy, f1, precision, recall],
    n_jobs=20,                                  # Max training jobs in parallel
    ensemble_size=1,                            # Limit ensemble to 1 base model only
    initial_configurations_via_metalearning=0,
    include={
        "classifier": ["random_forest"]         # Limit classifier type in ensemble to Random Forest
    }
)

# Construct the pipeline with the manual preprocessor and the AutoML pipeline
model = make_pipeline(preprocessor, automl_model)

# Fit the model on training set
model.fit(X_train, Y_train)

# Evaluate on test set
Y_pred = model.predict(X_test)

print("Accuracy:  ", accuracy_score(Y_test, Y_pred))
print("F1 score:  ", f1_score(Y_test, Y_pred, average='weighted'))
print("Precision: ", precision_score(Y_test, Y_pred, average='weighted'))
print("Recall:    ", recall_score(Y_test, Y_pred, average='weighted'))
```

While auto-sklearn is a powerful framework, it is not immediately suitable for automated machine learning in a federated learning context. Here is an overview of some of the more practical challenges:

1. Auto-sklearn supports parallel computation and evaluation of ML models via the Dask.distributed framework (https://distributed.dask.org, accessed on 15 May 2023). Furthermore, parallelizing auto-sklearn across multiple machines is technically feasible (by configuring a Dask scheduler, a client, and multiple workers), though not as straightforward as a single-machine deployment. However, the nature of the distributed computing does not correspond with a federated learning context where no data but only model updates are shared with a centralized coordinating server.
2. Auto-sklearn leverages the scikit-learn [39] library, which specializes in classical machine learning pipelines. It has support for simple neural networks, such as a multilayer perceptron (MLP) classifier, but not for the construction of more sophisticated

neural networks for which specialized libraries, such as TensorFlow and PyTorch, exist. Whether or not to explore both classical ML models and deep learning models is a decision to be made by the MLOps engineer or data scientist, but ideally, the opportunity should be granted by the federated AutoML framework.

3. The scikit-learn library implements a multitude of classifiers and regressors for which implementing a federated equivalent is not trivial. For example, the iterative approach of federated learning works well for logistic regression (LR) and multi-layer perceptron (MLP) but not for support vector classification (SVC). Indeed, for LR model updates, the internal coefficients of the model can be easily merged with those of other model updates to construct an aggregated model, for example, through federated averaging. For other methods, such as SVC, the implementation does not allow direct access to these coefficients or to construct a new model via the aggregated coefficients.
4. As depicted in Figure 1, auto-sklearn not only selects ML models and optimizes their hyperparameters, but it also implements many techniques for automated data and feature pre-processing. In federated learning, it is not trivial to implement these. Regarding data pre-processing, one-hot encoding only works effectively if all clients in the federation have the same categorical values and use the same process to compute the derived features. However, this scenario is rather unlikely. The rescaling of numeric features may be a bit easier to compute collaboratively, but it assumes that at least some meta-features (e.g., min and max values) are being shared by the clients with the coordinating server. For feature pre-processing, techniques including dimensionality reduction are far less trivial to realize without centralizing the training data.
5. Auto-sklearn constructs an ensemble model that is optimized for a particular error metric (e.g., f1, see line 31 in Listing 1 while computing other metrics after constructing the ensemble (see line 32 in the same figure). However, in a federated learning context, there are multiple trade-offs, including the resource usage (CPU, memory, network) for the clients in the federation as well the coordinating server. While it is possible to implement a custom error metric, the auto-sklearn framework can only be extended by following the provided APIs, and those are only data-oriented. As such, auto-sklearn does not support multi-objective optimization out of the box.
6. In a federated learning context, there are many more hyperparameters for collaboratively learning an ML model. Examples include the number of clients to involve in each training round (all clients or only a subset), the method to select these clients (random or adaptive), the relative impact of each model update (uniform, weighted by the amount of data), the size of each model update (full model update or only slices of a model update), etc. There are many more hyperparameters to explore in federated learning. Some of them might be decided upon upfront by the MLOps engineer, whereas others are subject to optimization in a given deployment context.
7. Security and privacy are important concerns, and various defenses have been proposed to counter threats. Whether these threats are relevant to the application and data at hand is something that cannot be decided upon automatically. If an MLOps engineer or data scientist identifies a certain threat, a multitude of countermeasures may be available that influence not only the result but also how efficiently the resulting ML model was obtained. For example, differential privacy may be less resource intensive during training compared to cryptographic techniques, such as MPC and HE, but the final model may be less accurate due to the introduction of noise. Unfortunately, these countermeasures are not part of auto-sklearn but are (partially) available in state-of-practice federated learning frameworks, such as PySyft [33].
8. Building upon the previous challenge, in a federated learning scenario that deals with sensitive information, it is paramount that the AutoML framework never explores models without proper countermeasures. Otherwise, the Bayesian optimization might pick a configuration from the search space where the data or the model updates are

not properly protected, such that the optimization process itself might leak sensitive information during the automated machine learning.

Addressing all the above concerns within a single framework is beyond the scope of this work, but it clearly shows there are many more hyperparameters and trade-offs to be considered in federated learning scenarios. In the following subsection, we will highlight the approach behind our framework and how it can be further extended.

4. Design and Implementation of the AutoFL Framework

In this section, we will discuss the design principles of our AutoFL framework, as well as details about how the framework was implemented.

4.1. Conceptual Overview

In Figure 2, we can observe a high-level representation of the AutoFL framework, showcasing its key components and improvements compared to conventional AutoML frameworks such as auto-sklearn depicted in Figure 1. We will now delve into the similarities and differences between the two frameworks, emphasizing how these variances contribute to addressing the aforementioned limitations.

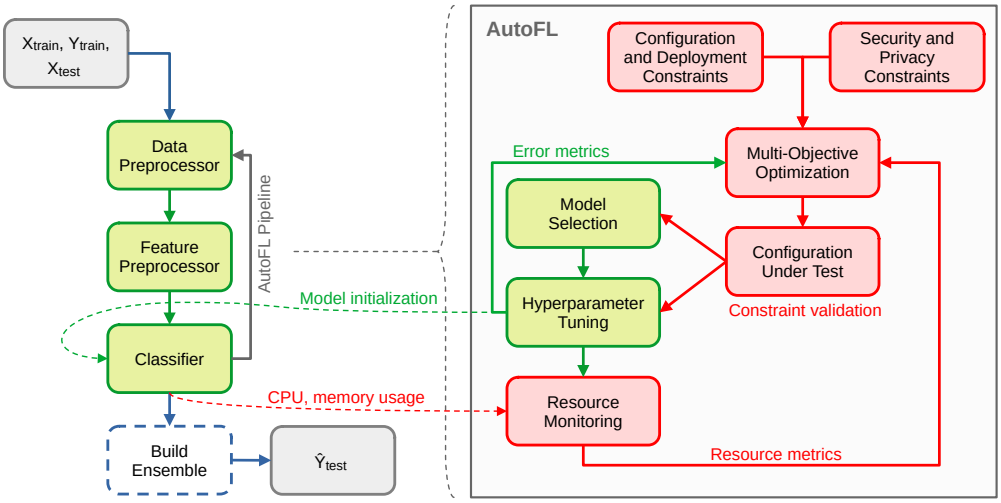


Figure 2. Conceptual block diagram of AutoFL with support for multi-objective optimization and application-specific constraints.

AutoFL has two significant components in common with auto-sklearn and other AutoML frameworks, and those are the *Model Selection* and the *Hyperparameter Tuning*. AutoFL supports a selection of traditional ML and deep learning models. In practice, multiple ML models and/or different hyperparameters for these models are tested in parallel. Where our framework differs is the search space in which the 'best' model needs to be found:

- *Model Selection*: Not every traditional ML model has an equivalent collaborative or federated learning implementation, and this reduces the search space for AutoFL.
- *Hyperparameter Tuning*: AutoFL must account for the hyperparameters of each ML model (e.g., the max depth of a decision tree), but also the hyperparameters for the federation itself (e.g., local training rounds per epoch). This typically increases the search space.

These capabilities of AutoFL effectively tackle limitations 1, 2, 3, and 6. To address limitation 5, AutoFL goes beyond optimizing machine learning models based on a single objective and additionally considers other concurrent optimization objectives if they are relevant to the specific application and deployment environment:

- *Model Objectives*: In AutoFL, these objectives commonly revolve around established *error metrics* such as accuracy, F1 score, precision, recall, ROC-AUC, and others. In

addition to these standard metrics, alternative criteria can also be considered, for example, to compare the *interpretability* of different machine learning models.

- *Resource Objectives*: The objectives encompass efficient resource utilization, such as CPU, memory, and network traffic, both during federated training and after model deployment. Two trivial examples are minimizing memory usage and reducing the time required to evaluate a single input sample during deployment.

AutoFL implements various runtime monitors and aggregates resource usage statistics across the nodes in the federation. The model error and resource usage metrics then feed the *Multi-Objective Optimization* component in Figure 2. This component of AutoFL leverages the SMAC3 library [40]. This library, which can be found at https://automl.github.io/SMAC3/ (accessed on 3 July 2023) and https://github.com/automl/SMAC3 (accessed on 3 July 2023), provides, amongst others, Bayesian optimization capabilities for both single- and multi-objective optimization. For multi-objective optimization, SMAC3 utilizes the ParEGO algorithm [41]. This algorithm combines multiple objectives into a single scalar objective, allowing SMAC3 to optimize it in a manner similar to single-objective optimization. Nonetheless, it retains the ability to identify configurations that lie on the Pareto front. Implementing these optimizations using the SMAC3 library is relatively straightforward.

The *Configuration Under Test* component is responsible for ensuring that any proposed configuration complies with pre-defined deployment, security, and privacy constraints before the model is trained in a federated manner. Additionally, it tracks metrics and other runtime statistics across multiple configurations. For instance, this component integrates techniques to detect class imbalance by analyzing the sample numbers of the minority and majority classes. It follows an approach similar to the one proposed in [42]. The component evaluates these class imbalances at both the individual node level and the federation level as a whole. This component addresses limitation 5.

The components responsible for *Configuration and Deployment Constraints* as well as *Security and Privacy Constraints* play a crucial role in defining the criteria for valid models and imposing necessary limitations. For instance, they impose limitations on the maximum amount of memory to be used either to train the model or after putting the model in production (e.g., physical constraints of target devices) or the average time required to evaluate the model for a single test sample (e.g., to support real-time data analysis such as network traffic monitoring). The latter component may introduce security and privacy measures during the aggregation process. For instance, secure aggregation techniques are enforced to address concerns related to an untrusted coordinating server, while differential privacy methods are enforced to uphold data confidentiality on individual nodes. If certain models do not support these security and privacy tactics, they will be excluded during the process of model selection. AutoFL cannot decide on its own whether these tactics are necessary; they have to be declared by the data scientist or MLOps engineer. These two components help address limitations 6, 7, and 8.

The current implementation of AutoFL successfully addresses all challenges except limitation 4. Although basic data pre-processing techniques such as min-max normalization can be performed in a federated manner, there is currently limited support for federated dimensionality reduction and data/feature pre-processing in a privacy-preserving manner.

4.2. Methodology

Similar to previous works, the goal of our framework is to explore the search space and develop high-quality models through federated learning while at the same time also considering non-trivial trade-offs and practical limitations.

4.2.1. Dataset Distribution

In our federated learning experiments, we make use of well-known datasets, such as MNIST (http://yann.lecun.com/exdb/mnist/, accessed on 1 May 2023), Fashion-MNIST (https://github.com/zalandoresearch/fashion-mnist, accessed on 1 May 2023) and CIFAR-

10 (https://www.cs.toronto.edu/~kriz/cifar.html, accessed on 1 May 2023). The way these datasets are distributed across the different clients in the federation can be configured:

- i.i.d. or non-i.i.d.: In the non-i.i.d. configuration, the different clients only have a specific number of labels or a variation thereof.
- balanced or unbalanced: When balanced, the data is uniformly distributed across the different clients; otherwise, it is not. This mode is only supported for i.i.d. data.

In practice, though, the data distribution across the different clients is an aspect that can be configured independently of the automated federated learning pipeline. For example, in a real-world dataset, it may not be known in advance whether the data and labels are uniformly distributed across the clients or whether there is some skewness. The reason we also test with well-known datasets which are distributed in a reproducible manner is to systematically compare the impact of particular configuration options and the trade-offs along the different optimization objectives.

4.2.2. Meta-Learning

As explained before, meta-learning allows us to define and constrain the search space of the ML models and hyperparameters by leveraging experience obtained through learning from reference datasets. As our goal is not only to optimize a particular error metric (e.g., accuracy or f1) but also to account for resource usage, privacy-enhancing techniques, etc., we currently do not implement meta-learning. For example, the application of privacy-enhancing techniques can have an impact on the error metric or the computational overhead. Due to this multi-objective optimization, it is not possible to reduce the search space similarly because of these trade-offs.

4.2.3. Data and Feature Pre-Processing

Data rescaling and dimensionality reduction are typical steps in a machine-learning pipeline. However, in a federated learning scenario, these steps are far less trivial to realize. Our framework implements a subset of the aforementioned techniques depending on whether clients are willing to share certain meta-features (e.g., min, max, mean, and variation of a feature value) in the same way clients in a federated learning scenario reveal the amount of data they individually train upon for the coordinating server to implement weighted federated averaging. Indeed, after receiving the model updates (e.g., the new weights w of a neural network) of each client k in round t, the coordinating server computes the aggregated model w_t via the local model updates w_t^k of the K clients through federated averaging as follows:

$$w_t \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_t^k \tag{1}$$

The impact of a single client depends on the amount of data n_k it locally trains upon. This means that the coordinating server knows this meta-feature n_k of each client k and the total amount of data n. The other meta-features (e.g., min, max, mean, variance of a feature value) can be shared with the coordinating server similarly.

However, when the client model updates are privacy sensitive or when the coordinating server cannot be trusted, then fortunately, state-of-practice federated learning frameworks offer alternative schemes (e.g., secure aggregation through secret sharing and secure multi-party computation). The question then becomes whether the additional meta-features can be shared not only with the coordinating server but with all clients such that, for example, each client implements the same min-max rescaling. Computing the min and max value of a particular feature value across all clients then becomes a multi-party variation of Yao's Millionaires' problem [43,44]. In the max value variation, the goal is not to learn which party has the highest value without revealing the individual values but rather to learn what is the maximum value across the parties without sharing the individual values or knowing which party has the highest value (except for that particular party itself). More generically, given K clients, we compute a meta-feature z through a deterministic

function f known by all clients or parties and based on private inputs x_k such that each party obtains no additional information other than z:

$$z = f(x_1, x_2, \ldots, x_K) \tag{2}$$

In our work, we now assume that any adversarial party in the federation follows a *semi-honest* or *honest-but-curious* threat model, i.e., all parties follow the protocol but are curious to know more information about the private inputs x_k of the other parties.

Due to their innate complexity to implement them in a distributed and privacy-preserving manner, our framework does not support any of the various dimensionality reduction techniques that are commonly applied in classical machine learning pipelines with high dimensional datasets.

4.2.4. Classifier

Our framework currently supports automated federated learning for binary and multi-class classification tasks with classical machine learning as well as deep learning methods, but there is currently no support yet for automated and federated regression or clustering tasks. As our framework leverages the scikit-learn [39] library for classical machine learning, we first selected a subset of classifier methods that can be applied in a federated setting, and we grouped them into two categories:

- *Coefficient-based models:* There are ML models that can easily be represented by a set of coefficients that can be incrementally learned. Example models include logistic regression and neural networks. For example, the `sklearn.linear_model.LogisticRegression` implementation of scikit-learn (as explained on the documentation website https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html, accessed on 4 May 2023) offers access to the internal representation via the attributes `coef_` and `intercept_`. Additionally, we only consider models whose attributes can both be read and directly written to, such that a new aggregated model can be computed by averaging the coefficients. For certain models, such as the linear SVC classifier, it is possible to read these attributes but not to modify them. Models with these kinds of implementation restrictions will not be considered by our framework.
- *Ensemble models:* These types of models typically combine a multitude of baseline models or estimators. Example models of this type include the random forest, AdaBoost, and XGBoost classifiers. For example, a random forest classifier uses decision tree models as their underlying estimators. These decision trees can be accessed via the `estimators_` attribute (as explained at https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, accessed on 4 May 2023). A trivial but perhaps not the best way to learn an aggregated random forest classifier in a federation is for each client to locally train a random forest classifier, and the coordination server to construct a new random forest based on the estimators of the clients' models. This way, there is no need to average model updates or to improve the aggregated model in multiple rounds. However, there might be better ways to improve the accuracy of the aggregated model at the expense of a less resource-efficient federated training approach.

4.2.5. Bayesian Optimization

Similar to auto-sklearn, our framework also adopts a Bayesian optimization approach to find the best model feasible through federated learning. However, contrary to the single optimization objective approach of auto-sklearn, our automated federated learning framework considers multiple optimization objectives in parallel, as outlined earlier. Additionally, the list of hyperparameters to be optimized is further extended with some federated learning-specific ones, such as the number of local training rounds and the number of participating clients in each training round. Under the hood, our framework uses the same optimization framework as auto-sklearn, namely SMAC3 [40]. This makes it

fairly trivial to configure the hyperparameter search space for different kinds of ML models and their respective hyperparameters, as illustrated in Listing 2. The code example is a reduced version of the actual implementation that configures several more ML models and hyperparameters, as well as hyperparameters specifically for federated learning. In addition, the example only implements a single optimization objective.

Listing 2. Simplified code example for the configuration of the hyperparameter search space of a subset of ML classifiers using the SMAC3 optimization framework.

```
cs = ConfigurationSpace()
model_type = CategoricalHyperparameter('model', ['logistic_regression', 'mlp', 'random_forest'],
                                       default_value='logistic_regression')
cs.add_hyperparameter(model_type)

########################################################################################################

hidden_layer_depth = UniformIntegerHyperparameter(name="hidden_layer_depth", lower=1, upper=3,
                                                  default_value=1)
num_nodes_per_layer = UniformIntegerHyperparameter(name="num_nodes_per_layer", lower=16, upper=264,
                                                   default_value=32, log=True)
activation = CategoricalHyperparameter("activation", choices=["tanh", "relu"], default_value="relu")
alpha = UniformFloatHyperparameter("alpha", lower=1e-7, upper=1e-1, default_value=1e-4, log=True)
learning_rate_init = UniformFloatHyperparameter("learning_rate_init", lower=1e-4, upper=0.5,
                                                default_value=1e-3, log=True)
early_stopping = CategoricalHyperparameter("early_stopping", choices=[True, False], default_value=
     True)
max_depth = UniformIntegerHyperparameter(name="max_depth", lower=5, upper=25, default_value=15,
                                         log=True)
max_features = UniformIntegerHyperparameter(name="max_features", lower=1, upper=3, default_value=2)
n_estimators = UniformIntegerHyperparameter(name="n_estimators", lower=5, upper=25, default_value=15)

cs.add_hyperparameters([
    hidden_layer_depth, num_nodes_per_layer, activation, alpha, learning_rate_init, early_stopping,
    max_depth, max_features, n_estimators
])

########################################################################################################

cs.add_conditions([
    InCondition(child=hidden_layer_depth, parent=model_type, values=['mlp']),
    InCondition(child=num_nodes_per_layer, parent=model_type, values=['mlp']),
    InCondition(child=activation, parent=model_type, values=['mlp']),
    InCondition(child=alpha, parent=model_type, values=['mlp']),
    InCondition(child=learning_rate_init, parent=model_type, values=['mlp']),
    InCondition(child=early_stopping, parent=model_type, values=['mlp']),
    InCondition(child=max_depth, parent=model_type, values=['random_forest']),
    InCondition(child=max_features, parent=model_type, values=['random_forest']),
    InCondition(child=n_estimators, parent=model_type, values=['random_forest']),
])

# Scenario object specifying the optimization environment
scenario = Scenario({'run_obj': 'quality',
                     'runcount-limit': runcount,
                     'cs': cs,
                     'deterministic': 'true'
          })

smac = SMAC4BB(scenario=scenario, tae_runner=test_model_config)
incumbent = smac.optimize()
```

Although it is also possible to configure the application of privacy-enhancing techniques through hyperparameters, it is rather straightforward that excluding these techniques will lead to models that are more accurate (e.g., no additional noise) and/or computationally less expensive to compute (e.g., no cryptographic or secure computation techniques) at the expense of possibly leaking sensitive information. However, the loss of privacy or confidentiality is hard to quantify and systematically compare across different privacy-enhancing techniques. That is why the application of these techniques is decided upon up front and not as a trade-off in the Bayesian optimization process.

Important to note is that SMAC3 offers two different strategies to support multi-objective optimization. Still, they have in common that the multiple objectives are aggregated into a single scalar objective, and that single objective is then optimized by SMAC3.

4.2.6. Implementation of AutoFL

AutoFL is implemented in Python and leverages various well-known frameworks and libraries. An overview of the major building blocks is provided below:

- *scikit-learn*: Selection of classical ML classifiers.
- *TensorFlow*: Deep learning models.
- *SMAC3*: Bayesian optimization of hyperparameters.
- *mpyc*: Secure multi-party computation for federated data pre-processing.
- *Flower*: Federated learning framework.

In practice, it should also be possible to introduce other ML frameworks (e.g., PyTorch). Next to that, we use different Python libraries to monitor resource usage (e.g., CPU, memory, and network usage) on the clients and the coordinating server.

5. Evaluation

After providing more details about our experimental setup, we will evaluate our framework in different scenarios and with different datasets.

5.1. Experimental Setup

Our experimental setup consisted of different types of computing nodes with different resource availability.

- *Laptop*: An HP ZBook Power laptop with an Intel Core i7-11800H running at 2.30 GHz, 32 GB of memory, and an NVIDIA T1200 GPU for deep learning. This device is used for small-scale centralized federation simulations and benchmark purposes.
- *Pi4*: A Raspberry Pi 4 with a Broadcom BCM2711, Quad-Core Cortex-A72 (ARM v8) 64-bit SoC running at 1.8 GHz, and 4 GB of memory. This device is used in a federation with heterogeneous resources.
- *JetsonTX2*: An NVIDIA Jetson TX2 development board with a Dual-Core NVIDIA Denver 2, a 64-Bit CPU Quad-Core ARM Cortex-A57, with 8 GB of memory. The device is also equipped with a 256-core NVIDIA Pascal architecture GPU. This device is used in a federation with heterogeneous resources.
- *Server*: A high-end server with an AMD EPYC 7502 32-Core processor running at 3.32 GHz, and 256 GB of memory. This machine is used for larger-scale centralized federation simulations and in a federation with heterogeneous resources.
- *Client*: Twenty client machines with either an Intel Core i5-4570S CPU running at 2.90 GHz or an Intel Core i5-6500 CPU running at 3.20 GHz, and 8 GB of memory. These machines are used for experiments in a homogeneous federation.

All machines and edge devices were connected to a wired Gigabit network and typically run a Ubuntu 20.04 or 22.04 LTS Linux operating system with a recent Miniconda Python 3.10 stack (https://repo.anaconda.com/miniconda/, accessed on 4 May 2023). The Jetson TX2 board ran the latest supported operating system, i.e., Ubuntu 18.04 LTS, while the Raspberry Pi 4 ran the Debian Bullseye 11.7 operating system. The two edge devices both ran an Aarch64 version of Miniforge Python 3.10 (https://github.com/conda-forge/miniforge, accessed on 4 May 2023).

5.2. Baseline Experiment

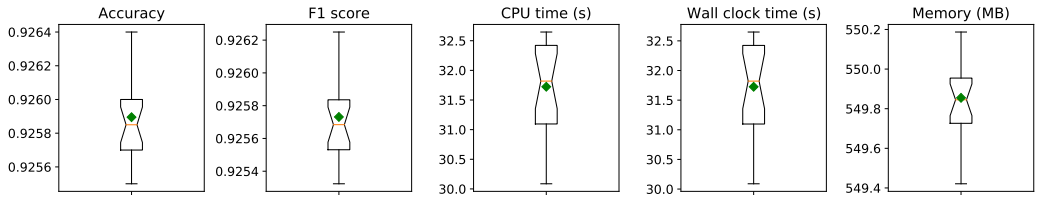
We first trained a selection of classical ML models and neural networks on the MNIST dataset on the various systems and compared their baseline accuracy as well as their resource usage and the time passed. The ML models were trained locally in a centralized manner. The initialization of the ML models and their hyperparameters are listed in Listing 3. The full Python code to prepare the MNIST datasets and train the LogisticRe-

gression() model is listed in Appendix A, respectively, in Listings A1 and A2. The code examples for the other ML models are similar.

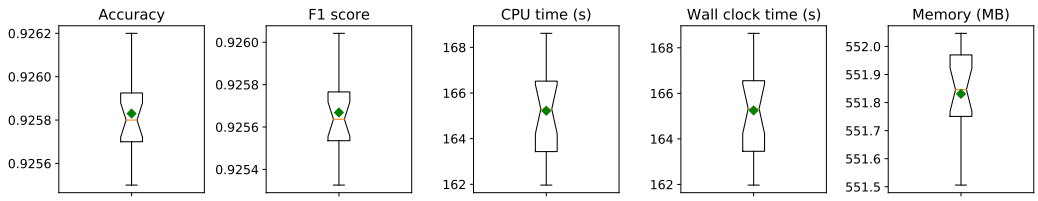
Listing 3. Classical ML models and neural networks for baseline MNIST benchmarking.

```
clf1 = RandomForestClassifier(max_depth=20, n_estimators=100, max_features=2)

clf2 = LogisticRegression(solver='saga', penalty='l2', max_iter=20, C=50, tol=0.01, verbose=0)

clf3 = MLPClassifier(hidden_layer_sizes=(50,), max_iter=20, alpha=1e-4, solver='sgd', verbose=0,
                     random_state=1, learning_rate_init=.1)

clf4 = tf.keras.models.Sequential([
           tf.keras.layers.Flatten(input_shape=(28, 28)),
           tf.keras.layers.Dense(128, activation='relu'),
           tf.keras.layers.Dropout(0.2),
           tf.keras.layers.Dense(10)
       ])

clf5 = tf.keras.models.Sequential([
           tf.keras.layers.Conv2D(32, 3, activation="relu", input_shape=(28, 28, 1)),
           tf.keras.layers.MaxPooling2D(),
           tf.keras.layers.Flatten(),
           tf.keras.layers.Dense(64, activation="relu"),
           tf.keras.layers.Dense(10)
       ])
```

Note that the hyperparameters of these models were manually selected and were not optimal. Hence, they only served to demonstrate resource heterogeneity across the machines. Figure 3 depicts the results of 20 runs of the LR model on the different systems. What we can observe in these results is the fact that the results are more or less consistent (i.e., no big deviations across the different runs). Furthermore, the accuracy and f1 score are the same across all devices, as expected. The overall memory usage of 550 MB is very similar too. It is also not surprising that the CPU usage and wall clock time are the same within a particular device type due to the fact only one CPU core was used. However, there are differences across the different device types.

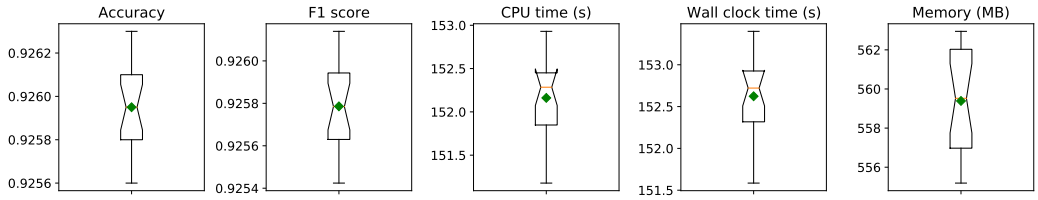


(**a**) Laptop

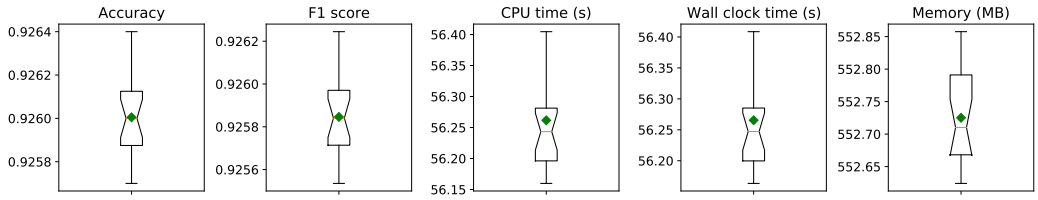


(**b**) Raspberry Pi 4

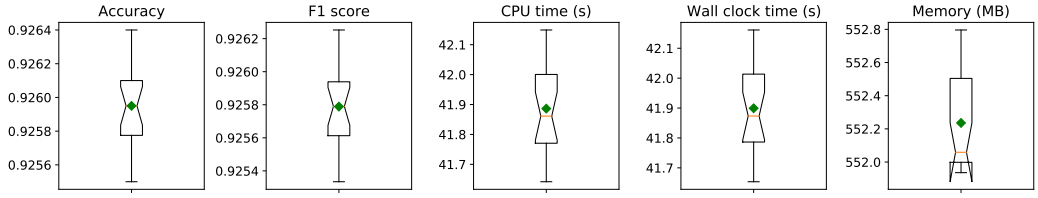
Figure 3. *Cont.*



(**c**) NVIDIA Jetson TX2



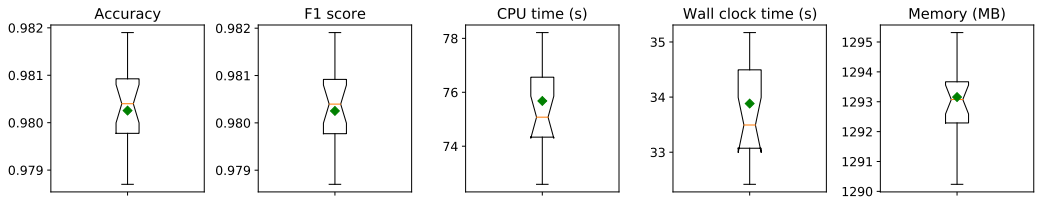
(**d**) Client



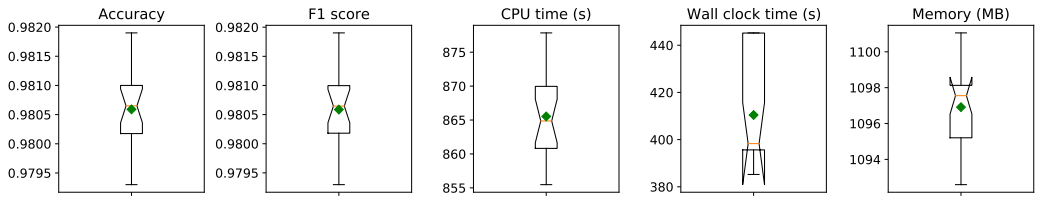
(**e**) Server

Figure 3. Boxplots of benchmarks of scikit-learn's logistic regression (LR) on different devices (the green diamond determines the mean value; the ends of the notched box represent the lower and upper quartiles; the orange line inside the notched box indicates the median value).

Figure 4 depicts a similar story, but now for the feed-forward neural network implemented with TensorFlow (i.e., clf4). In these benchmarks, a device used its CPU even if a GPU was available to accelerate the training. The accuracy and f1 score are on par across the board. Memory usage is significantly higher and reasonably consistent within the same device type, but less consistent across the devices. Since multiple CPU cores are used, the CPU time is at least twice as high compared to the wall clock time.

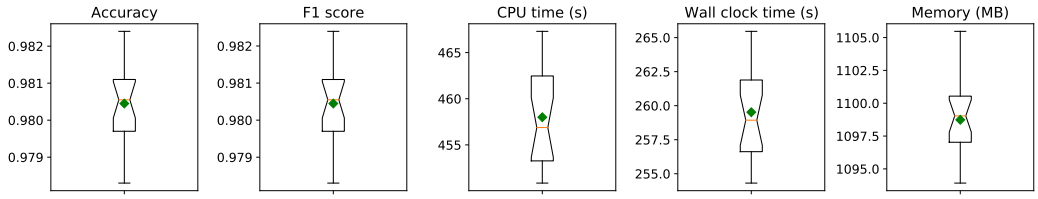


(**a**) Laptop

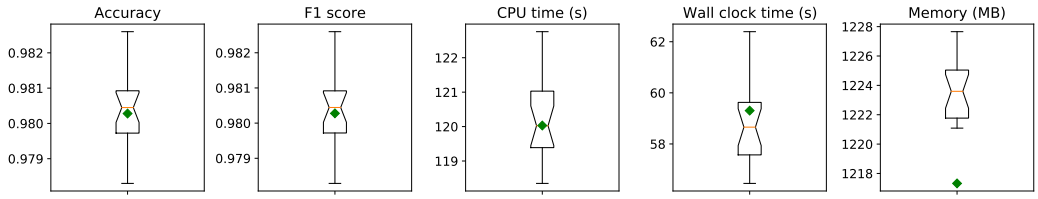


(**b**) Raspberry Pi 4

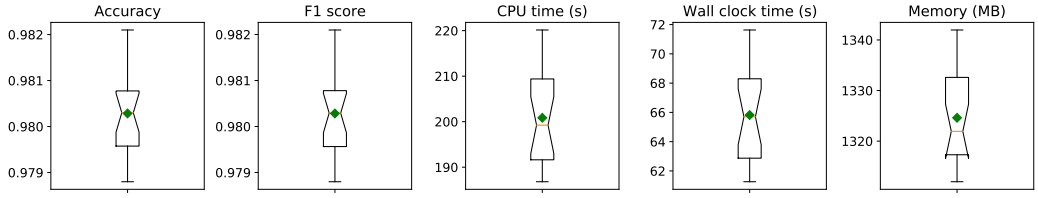
Figure 4. *Cont.*



(**c**) NVIDIA Jetson TX2



(**d**) Client



(**e**) Server

Figure 4. Benchmarking TensorFlow's feed-forward neural network (clf4) on different devices.

In Table 1, we provide a detailed metric overview of different baseline models for MNIST classification on different devices. The reported values are the average of 20 runs. Since the dataset is well-balanced, the accuracy is almost identical to the f1 score. All models except random forest (RF) were trained in multiple rounds. We fixed the number of training rounds to 20 for the logistic regression (LR), multi-layer perceptron (MLP), and feed-forward (FF) models, and set the number of epochs for the convolutional neural network (CNN) to 10. Table A1 in Appendix A reports the results for the same model configurations, but now on the FashionMNIST dataset. The latter has similar characteristics in terms of data format and size but is more challenging to classify compared to MNIST. In Table A2, we report the results for the CIFAR-10 dataset for the same models. This dataset is more sophisticated, and hence the performance results are subpar, as expected. For example, the MLP model has an accuracy of 0.1, which for a dataset with 10 classes is the same as random guessing. From these baseline experiments, it is clear from a computational and accuracy point of view that the MNIST dataset is the least challenging, whereas the CIFAR-10 dataset is the most challenging.

Table 1. MNIST baseline benchmark for classical ML models and neural networks.

	Pi 4	Jetson TX2	Laptop	Client	Server
Accuracy					
Random Forest	0.953	0.953	0.953	0.953	0.953
Logistic Regression	0.926	0.926	0.926	0.926	0.926
Multi-Layer Perceptron	0.972	0.972	0.972	0.972	0.972
TensorFlow-FF	0.981	0.980	0.980	0.980	0.980
TensorFlow-CNN	0.986	0.986	0.986	0.986	0.986
F1 score					
Random Forest	0.953	0.953	0.953	0.953	0.953
Logistic Regression	0.926	0.926	0.926	0.926	0.926
Multi-Layer Perceptron	0.972	0.972	0.972	0.972	0.972
TensorFlow-FF	0.981	0.980	0.980	0.980	0.980
TensorFlow-CNN	0.986	0.986	0.986	0.986	0.986
CPU time (s)					
Random Forest	24.013	18.996	3.468	5.401	4.768
Logistic Regression	165.221	152.161	31.724	56.262	41.886
Multi-Layer Perceptron	308.138	161.850	162.388	43.824	996.448
TensorFlow-FF	865.530	457.998	75.678	120.031	200.849
TensorFlow-CNN	3644.037	1216.834	243.083	555.559	1441.772
Wall clock time (s)					
Random Forest	24.019	19.149	3.470	5.426	4.773
Logistic Regression	165.250	152.623	31.725	56.265	41.899
Multi-Layer Perceptron	77.298	40.905	10.887	11.017	15.608
TensorFlow-FF	410.429	259.521	33.883	59.304	65.821
TensorFlow-CNN	1052.609	385.810	38.621	185.502	134.152
Memory (MB)					
Random Forest	742.297	743.723	759.760	731.340	737.709
Logistic Regression	551.831	559.388	549.855	552.725	552.236
Multi-Layer Perceptron	552.848	559.618	551.349	555.092	552.052
TensorFlow-FF	1096.914	1098.737	1293.157	1217.322	1324.608
TensorFlow-CNN	1131.936	1140.021	1306.701	1252.293	1572.459

5.3. Homogeneous Federation: i.i.d. versus Non-i.i.d

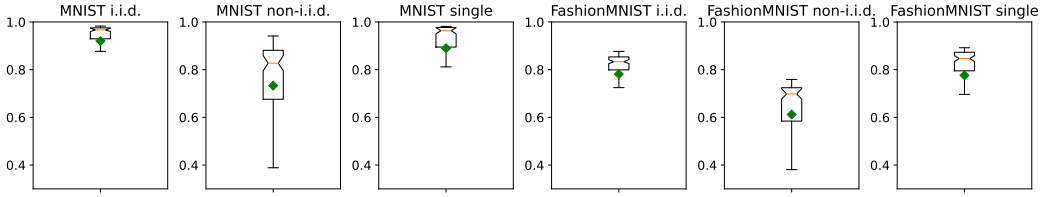
The previous experiments demonstrated the computational complexity of a selection of manually configured classical ML models and neural networks. In this particular experiment, we used Flower version 1.4 [31,32] to implement and configure a federation with ten clients and one coordinating server each having a similar resource availability and compare that against a federation with a coordinating server and only one client. Additionally, we apply our SMAC3-based Bayesian optimization framework to automatically find the best model and corresponding hyperparameters.

To simplify the comparison of the results, we constrained the experimental setup with the following settings:

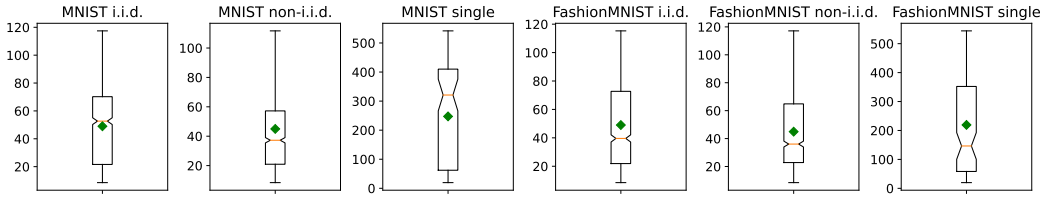
- The Bayesian optimization procedure was fixed to evaluating 100 different classical ML or deep learning model configurations, during which we not only assessed the accuracy but also measured the CPU, memory, and network usage on the clients as well as the coordinating server.
- We excluded all scikit-learn classifiers that could not be aggregated through federated averaging of the model's weights or coefficients (e.g., random forest). Next to TensorFlow-based deep learning models, our framework explored these scikit-learn classifiers: `LogisticRegression()`, `Perceptron()`, `MLPClassifier()` and `PassiveAggressiveClassifier()`.
- For each model configuration, the coordinating server carried out 20 federated averaging rounds, and each client locally trained their model for one epoch before sending the model update to the coordinating server.
- All clients were involved in each round of the federated learning (instead of only a random subset), and each client trained on its own partition of the training dataset.

- For the i.i.d. training data, each client had about the same amount of samples for each class (about 600 for MNIST and FashionMNIST). For the non-i.i.d. training data, each client had about 3000 samples of two classes each and only 2 samples of the other eight classes. In both distributions, each client has a similar amount of training data.

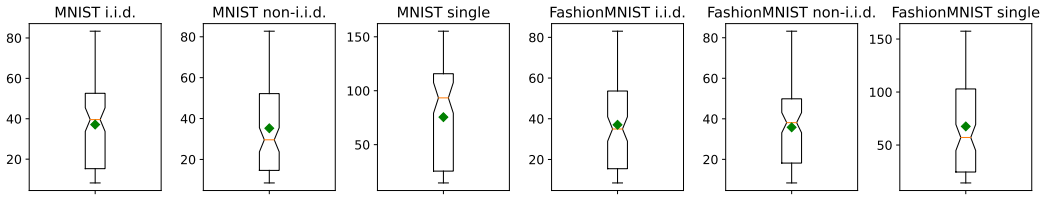
Figure 5 compares the distribution of the results for the 100 configurations evaluated by the Bayesian optimization process, and this for a baseline with only one client as well as two configurations where the MNIST or FashionMNIST training data was partitioned across 10 clients in either an i.i.d. or non-i.i.d. manner. Based on the above observations, the accuracy of the 100 single baseline configurations and the i.i.d. configurations are comparable, whereas the non-i.i.d. configurations did not converge to the same accuracy within the 20 federated training and averaging rounds.



(**a**) Accuracy



(**b**) CPU time on client (in seconds)



(**c**) Wall clock time on coordinating server (in seconds)

Figure 5. Benchmarking 100 ML models in i.i.d. and non-i.i.d. dataset configurations on clients with homogeneous resource availability.

Note though, that Figure 5 gives a statistical overview of all the 100 individual configurations, including the suboptimal ones. However, we should rather compare the best configuration after the Bayesian optimization process has been completed. Hence, the accuracy on the test set for the best model found is shown in Table 2. The accuracy values for the MNIST single baseline and i.i.d. configurations are comparable to the best ones reported in Table 1. The accuracy values for the CIFAR-10 dataset are reasonably low compared to the state-of-the-art, and we expect this outcome to be due to the limited number of training rounds (i.e., 20 rounds) as well as the low number of configurations (i.e., 100 configurations) for the Bayesian process to evaluate.

Table 2. Best accuracy of 100 configurations after 20 federated averaging rounds.

Dataset	i.i.d.	Non-i.i.d.	Single
MNIST	0.9826	0.9411	0.9813
FashionMNIST	0.8766	0.7589	0.8919
CIFAR-10	0.6367	0.5020	0.6839

When comparing the wall clock time for the coordinating server to complete the federated learning, there is hardly any difference between the i.i.d. and non-i.i.d. dataset configurations, neither for the MNIST dataset nor for the FashionMNIST dataset. This is as expected as the same amount of client nodes process their part of the training data. When compared to the single baseline configuration, the overall wall clock time on the coordinating server is higher as now a single client has to learn from the whole training dataset. The fact that the coordinating server now only needs to communicate with 1 client rather than with 10 clients has no significant effect. Similarly, the CPU time on the clients is significantly higher for the single baseline configuration, although not 10 times as high as compared to the i.i.d. dataset configuration.

As mentioned earlier, we are exploring other optimization objectives beyond the accuracy of the model. For example, in the case of the i.i.d. MNIST dataset, the best model obtained an accuracy on the test set of 0.9826. However, in the 100 configurations tested, there were 10 with an accuracy higher than 0.98, as depicted in Table 3.

Table 3. Configurations with an accuracy >= 0.98, with Pareto-optimal ones marked in bold.

Configuration	Accuracy	Memory (MB)	CPU Time (s)	Pareto-Optimal Configuration
0032	0.9824	707.252	52.782	
0037	0.9826	714.510	56.720	
0038	0.9810	714.653	56.625	0032, 0039
0039	0.9824	705.339	52.802	
0044	0.9822	734.507	65.311	0032, 0037, 0039
0057	0.9808	708.911	51.312	
0071	0.9825	744.964	76.580	0037
0093	0.9822	747.036	77.205	0032, 0037, 0039
0095	0.9804	708.087	51.691	
0100	0.9807	716.832	56.687	0032, 0039, 0057

Of these ten configurations, five are Pareto-optimal (marked in bold) because they have either (a) a higher accuracy, (b) a lower memory usage, or (c) a lower CPU usage compared to the other configurations. Alternatively, for the non-Pareto-optimal configurations, there exists a configuration (last column) that performs equally well or better for all the metrics.

The above result is a mere demonstration of how multiple optimization objectives can be considered in parallel. For example, if memory is not a concern, then this trade-off can be dropped or replaced with another one (e.g., network usage at either the coordinating server or the individual clients). Furthermore, the Pareto-optimal configurations were identified with relative comparisons. With a more strict condition that one configuration is better than another in terms of resource usage, if the reduction is at least 10% (rather than just strictly smaller), then the number of Pareto-optimal solutions would be significantly reduced. In that scenario, configurations 0032, 0039, 0057, and 0095 would no longer be Pareto-optimal, as their resource usage is above the threshold or more than 90% of the memory and CPU usage of configuration 0037, while their accuracy is lower than 0.9826.

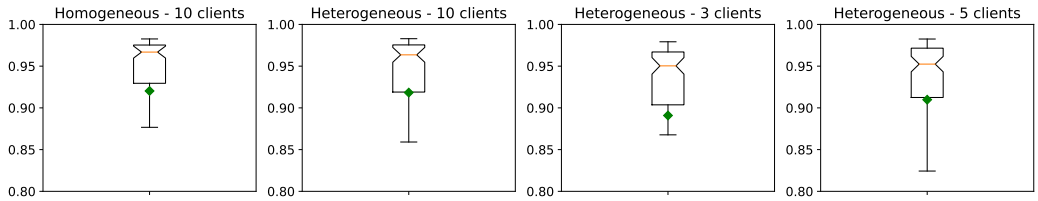
5.4. Heterogeneous Resource Availability

In the previous experiments, the clients in the federation had similar resource availability in terms of CPU, memory, and network capacity. In the following experiment, we replaced one of the ten clients with an ARM device, specifically the NVIDIA Jetson TX2 system. The other experimental settings remain the same:

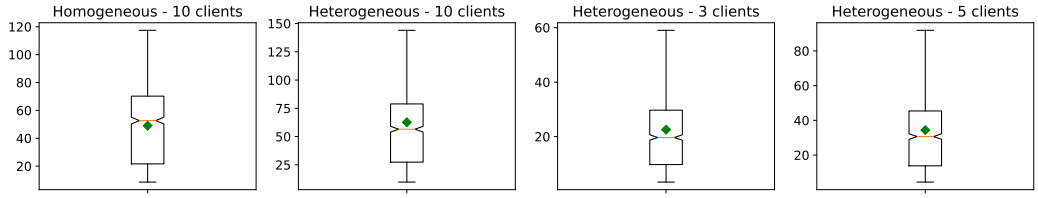
- Ten clients and one coordinating server in the federation.
- Training data is i.i.d. over all clients.
- One hundred ML model configurations to be selected through Bayesian optimization.
- Twenty federated averaging rounds by the coordinating server.
- One epoch for each local model training round.

Figure 6 depicts the results of three different experiments with the MNIST dataset that is i.i.d. over ten clients. In the deployment setting (1), all clients have homogeneous

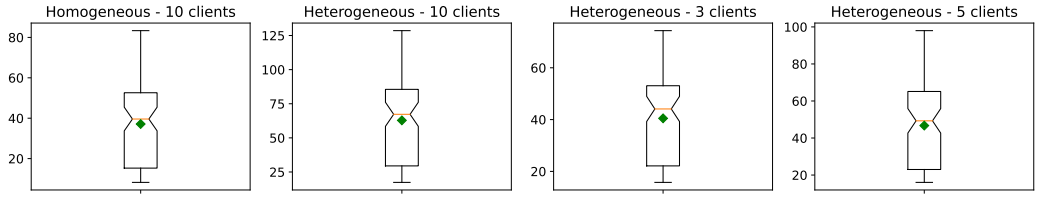
resource characteristics, and all ten clients are involved in training and evaluation. In deployment setting (2), one client is a low-end device, i.e., the Jetson TX2 development board. Deployment settings (3) and (4) are similar to deployment setting (2), but now only a random subset of, respectively, three and five clients are used in every federated training and evaluation round.



(**a**) Accuracy



(**b**) CPU time on client (in seconds)



(**c**) Wall clock time on coordinating server (in seconds)

Figure 6. Benchmarking MNIST in a federation with homogenous and heterogeneous resource availability and partial client involvement.

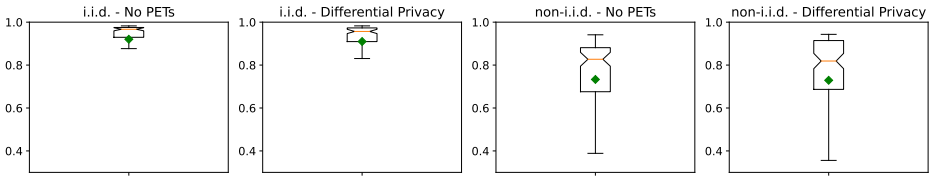
From these results, it is clear that the single resource-constrained device is slowing down the overall federated learning process. In the homogeneous deployment setting, the average wall clock time of the coordinating server is 37 s, whereas, for the second configuration, it increases to, on average, 63 s. The accuracy remains unaffected, as expected. However, when only using three random clients in each federation round rather than all ten, the wall clock time drops again to 40 s at the expense of the accuracy dropping from 0.92 down to 0.89. With five random clients, the wall clock time increases to about 47 s while the accuracy increases to 0.91. By using a subset of random clients, there is a chance that the slow client is not included in a federation round, thereby speeding up the overall federated learning process. Additionally, involving fewer clients means that the coordinating server needs less time to compute the federated average of the model updates, although this impact is minimal in our experiments. These four experiments show that there is, again, a trade-off between accuracy and wall clock time that is influenced by federation-specific hyperparameters. In our experiments, the network was never a bottleneck. If that would have been the case, we could have identified further trade-offs between network usage and the number of local training rounds on each client. In practice, however, the heterogeneous resource availability is likely to be much more diverse, even when only using a fraction of the clients, compared to the above deployment settings with one resource-constrained device and nine higher-end systems with similar resource availability.

5.5. Impact of Differential Privacy

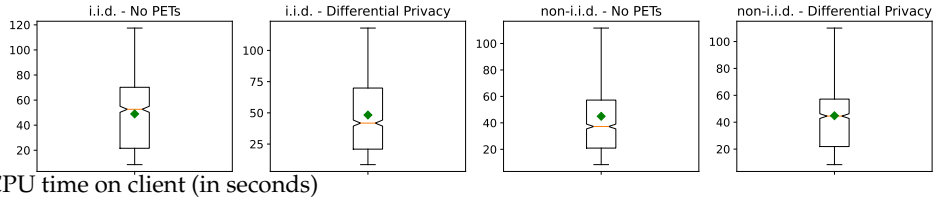
Privacy-enhancing techniques for federated learning, such as differential privacy or secure aggregation, may impact the accuracy of the model or the computational cost. In this experiment, we set up a federated learning configuration using MNIST and ten clients with homogeneous resource characteristics and with both the i.i.d. and non-i.i.d. datasets. The baseline did not have any privacy-enhancing techniques applied, and this baseline was compared with a similar setting with differential privacy applied. This privacy-enhancing technique bounds a client's model update by clipping the coefficient via a cap on the L2 norm of the update. Additionally, Gaussian noise is added to the federated average computed by the coordinating server. For the sake of convenience, we implemented the clipping of each client's updates within the coordinating server, but an alternative is that each client clipped their coefficients themselves. However, by implementing this strategy on the coordinating server, we expected the impact of applying this step for each client update to be more noticeable compared to a scenario where this step is carried out in parallel on each client.

The results of this benchmark are depicted in Figure 7. While there is a difference in accuracy between the i.i.d. and non-i.i.d. configurations of the training dataset, as discussed earlier, there is no significant difference for the wall clock time of the coordinating server when differential privacy is applied in this particular MNIST experiment. Hence, if the clipping of the model updates were executed by the clients themselves in parallel, the impact would be even less outspoken. For more complex datasets and with a higher number of clients, the outcome may be different.

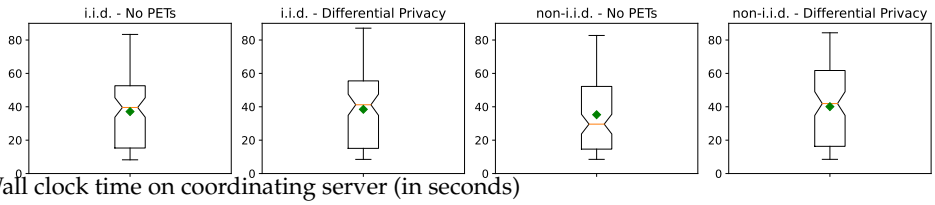
However, one must be careful with differential privacy in an automated federated learning setting. Traditionally, the privacy budget spent is incremented with each federated training round to compute the total privacy budget after the training process has ended. When Bayesian optimization is used to identify and evaluate different model configurations, this privacy budget must also be accounted for across these model configurations. Furthermore, if there is a maximum privacy budget, the Bayesian optimization process must stop if this threshold is reached; otherwise, there is a risk of information leakage. Currently, the Bayesian optimization does not take this privacy budget into consideration when identifying the next best model to evaluate.



(**a**) Accuracy



(**b**) CPU time on client (in seconds)



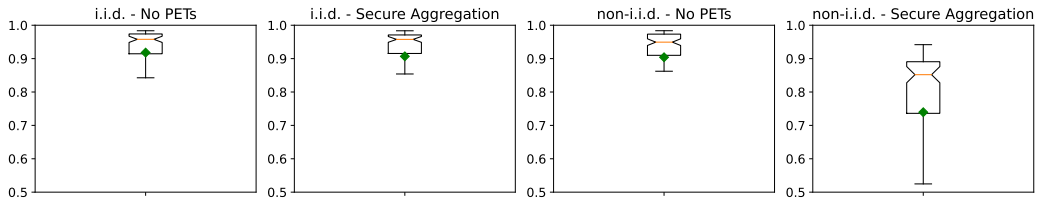
(**c**) Wall clock time on coordinating server (in seconds)

Figure 7. Benchmarking MNIST in a federation with and without differential privacy.

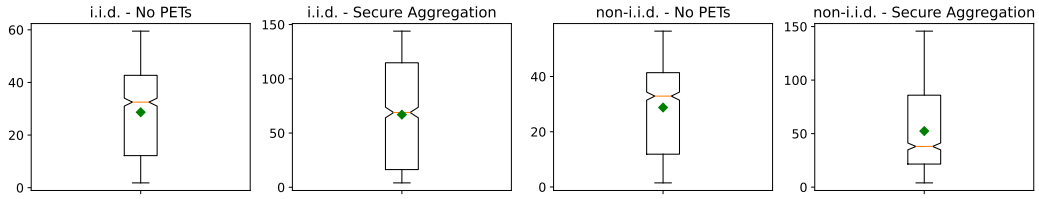
5.6. Impact of Secure Aggregation

Flower v1.14 does not support secure aggregation out of the box. Fortunately, Li et al. [45] implemented Salvia (cfr. https://hei411.github.io/projects/salvia.html, accessed on 4 May 2023) for a slightly older version of Flower v0.17, adding support for SecAgg(+) protocols under a semi-honest threat model. In the following experiment, we used the i.i.d. MNIST dataset in a federation with ten clients having similar resource availability and compared the traditional federated averaging strategy and this particular secure equivalent implemented in Salvia. For a fair baseline comparison, we used the older vanilla 0.17 version without secure aggregation to compare against.

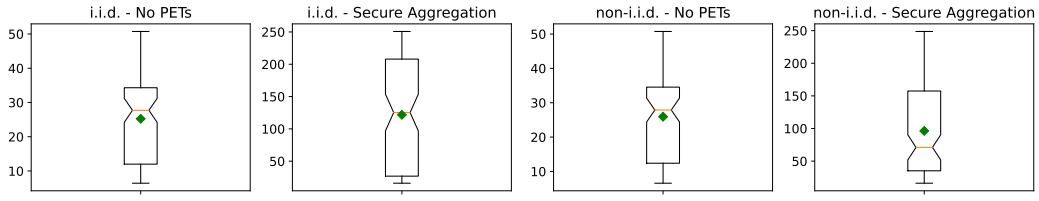
The results in Figure 8 indicate a difference in the baseline between Flower v1.14 and v0.17 in terms of wall clock time at the coordinating server. The mean accuracy for the 100 ML model configurations remains on par. For example, in the case of the i.i.d. dataset configuration, the mean accuracy is 0.918 vs. 0.907 for, respectively, without and with secure aggregation. A slight difference in mean values is to be expected as there is no guarantee that the Bayesian optimization process selects the same sets of 100 configurations. However, there is now a clear difference in the wall clock time caused by the computational impact of secure aggregation. Not only has the mean wall clock time gone up from 25 s to 121 s, but the variation of wall clock times is also more outspoken. The other metrics are indicated in Table 4.



(**a**) Accuracy



(**b**) CPU time on client (in seconds)



(**c**) Wall clock time on coordinating server (in seconds)

Figure 8. Benchmarking MNIST in a federation with and without secure aggregation.

Table 4. Quartile and minimum/maximum values of the coordinating server's wall clock time for secure aggregation configurations of MNIST i.i.d.

Metric	w/o SecAgg (s)	w/ SecAgg (s)
Minimum value	6.43	15.92
Lower quartile (25th percentile)	12.00	26.85
Median value (50th percentile)	27.70	125.29
Upper quartile (75th percentile)	34.30	207.99
Maximum value	50.73	250.76

As the computational impact of secure aggregation is non-negligible, the trade-offs between accuracy on the one hand and CPU time on the client, on the other hand, may now be more meaningful. After exploring the 100 configurations with secure aggregation, we filtered those with an accuracy of 0.98 or higher. The accuracy, memory usage, and CPU time of the clients of the selected eleven configurations are listed in Table 5. The memory usage is comparable except for configuration 0010 (i.e., 914 MB vs. ±650 MB). When comparing the different values, we end up with seven Pareto-optimal configurations marked in bold.

Table 5. Secure aggregation configurations of MNIST i.i.d. with an accuracy >= 0.98, with Pareto-optimal ones marked in bold.

Configuration	Accuracy	Memory (MB)	CPU Time (s)	Pareto-Optimal Configuration
0002	0.9800	647.282	64.329	
0010	0.9806	914.833	113.37	0016, 0020, 0021, 022, 0096, 0100
0016	0.9811	640.106	65.076	
0020	0.9826	662.110	86.150	
0021	0.9834	652.775	86.362	
0022	0.9821	652.513	86.751	
0043	0.9823	675.852	124.59	0020, 0021, 0096
0073	0.9804	651.759	107.83	0016, 0096, 0100
0089	0.9822	660.127	106.97	0021
0096	0.9830	651.382	107.03	
0100	0.9806	650.543	85.550	

With a more strict condition that a Pareto-optimal configuration is at least 10% better in terms of memory or CPU usage, the remaining Pareto-optimal solutions are 0016 (low CPU time) and 0021 (higher accuracy). The difference in computational complexity can be explained by the ML model configurations that performed best in terms of accuracy (i.e., all neural networks). Here are some examples:

- 0002: Scikit-learn MLPClassifier with one hidden layer of size 286
- 0010: TensorFlow neural network with one hidden layer of size 390
- 0016: scikit-learn MLPClassifier with one hidden layer of size 291
- 0021: scikit-learn MLPClassifier with one hidden layer of size 397
- 0043: scikit-learn MLPClassifier with three hidden layers of size 397, 268, 139
- 0096: scikit-learn MLPClassifier with two hidden layers of size 397, 203

These specifications and benchmark results illustrate that a simple neural network with fully connected layers requires less memory in scikit-learn's MLPClassifier than in TensorFlow. Given that the neural network architectures of model configurations 0002 and 0016 are fairly similar, their computational complexity is similar, too, as expected.

5.7. Discussion

The previous experiments highlighted the impact of different kinds of classical ML models and neural networks, the frameworks used to implement these ML models, the way the training data is distributed over the clients (i.e., i.i.d. or non-i.i.d.), the resource availability of the clients, and the application of privacy-enhancing techniques. In these experiments, we constrained the hyperparameter search space to limit the time required while also enabling a fair comparison between different configurations. For example, we fixed the number of federated averaging rounds to 20 and the local training rounds on each client per federated averaging round to 1.

These hyperparameters and several others can also be optimized, possibly in terms of other error metrics, with or without data pre-processing. Additionally, there are opportunities to (a) experiment with other datasets, (b) identify more trade-offs with mixed hardware-accelerated federated learning, (c) try different variations of imbalanced datasets, (d) compare adaptive client sampling, and (e) sophisticated strategies for federated averaging, (f) support more types of classical ML models as well as neural network architectures, (g) analyze the impact of adversarial training, etc.

The search space of hyperparameters—those of the ML models as well as those for federated learning—is so large that a Bayesian optimization process limited to 100 configuration evaluations will leave further room to identify ML models that may perform better in terms of either accuracy or resource usage or both. Indeed, due to the limited exploration time, the model configurations selected through Bayesian optimization and trained on the i.i.d. MNIST training dataset on 10 clients only achieved an accuracy of up to 0.983. However, the manually crafted neural network depicted in Listing 4 achieves an accuracy of 0.9945 on the test set. We would need to grant the Bayesian optimization process more time to obtain similar results.

Listing 4. Federal learned neural network for MNIST achieving an accuracy of 0.9945.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',
                           input_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'),
    tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.4),

    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.3),

    tf.keras.layers.Dense(n_classes, activation="softmax")
])

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer='adam', loss=loss_fn, metrics=['accuracy'])
```

The previous experiments have demonstrated the practical feasibility of optimizing for multiple objectives while adhering to various constraints. However, it is important to note that these experiments were conducted using well-known datasets and may not fully represent real-world case studies.

One of the main objectives of AutoFL is to uncover significant trade-offs in terms of model performance and resource utilization, considering factors such as memory usage and computational complexity. However, AutoFL itself does not determine the metrics to optimize for. This responsibility lies with the MLOps engineer, who has to define the optimization objectives and set deployment, configuration, security as well as privacy constraints based on the specific application requirements.

It is crucial to understand that AutoFL can assist in identifying trade-offs, but the final model selection remains the responsibility of the MLOps engineer. AutoFL provides a range of model configurations that lie on the Pareto front, which represents the trade-off between

the predefined optimization objectives. It is up to the MLOps engineer to choose the 'best' model from the Pareto front that aligns most effectively with the application's requirements.

6. Conclusions

In this paper, we designed, implemented, and evaluated AutoFL, our AutoML framework for federated learning that aims to optimize the ML model selection and hyperparameter optimization for scenarios where the training data cannot be centralized and where the data of a single client is not sufficient or not representative enough to construct an ML model configuration that is effective for all clients.

We evaluated AutoFL in different federated learning scenarios to explore trade-offs across both classical ML scikit-learn-based classifiers and TensorFlow-based neural networks. These scenarios differ in terms of the clients having different resource availabilities, the way the data is distributed across the clients, and whether privacy-enhancing techniques are implemented. Our research and experiments on well-known public datasets demonstrate the practical feasibility of our framework, allowing data scientists or ML engineers to identify trade-offs between error metrics and resource usage.

As part of future work, we will validate our framework on datasets outside the computer vision domain, and we will investigate the computational impact of exploring additional model-specific and federated learning-specific hyperparameters, privacy-enhancing techniques, and their trade-offs.

Funding: This research is partially funded by the Research Fund KU Leuven, and by the Flemish Research Programme Cybersecurity. This paper was also partially supported by the AIDE project funded by the Belgian SPF BOSA under the program "Financing of projects for the development of artificial intelligence in Belgium" with reference number 06.40.32.33.00.10.

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found at aforementioned urls.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Listing A1. Decode and save the MNIST training and test datasets to binary files in NumPy format.

```
import numpy as np
import gzip
import struct

def load_dataset(path_dataset):
    with gzip.open(path_dataset, 'rb') as f:
        magic, size = struct.unpack(">II", f.read(8))
        nrows, ncols = struct.unpack(">II", f.read(8))
        data = np.frombuffer(f.read(), dtype=np.dtype(np.uint8).newbyteorder('>'))
        data = data.reshape((size, nrows * ncols))
        return data

def load_label(path_label):
    with gzip.open(path_label, 'rb') as f:
        magic, size = struct.unpack('>II', f.read(8))
        label = np.frombuffer(f.read(), dtype=np.dtype(np.uint8).newbyteorder('>'))
        return label

X_train = load_dataset('data/train-images-idx3-ubyte.gz')
y_train = load_label('data/train-labels-idx1-ubyte.gz')
X_test = load_dataset('data/t10k-images-idx3-ubyte.gz')
y_test = load_label('data/t10k-labels-idx1-ubyte.gz')

X_train = X_train / 255.
X_test = X_test / 255.

np.save('X_train.npy', X_train)
np.save('y_train.npy', y_train)
np.save('X_test.npy', X_test)
np.save('y_test.npy', y_test)
```

Listing A2. Benchmarking logistic regression: accuracy, f1 score, CPU time, wall clock time, memory.

```
import time
import psutil
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

X_train = np.load("X_train.npy")
y_train = np.load("y_train.npy")
X_test = np.load("X_test.npy")
y_test = np.load("y_test.npy")

process = psutil.Process()

start_time_ns = time.process_time_ns()
start_counter_ns = time.perf_counter_ns()
clf = LogisticRegression(solver='saga', penalty='l2', max_iter=20, C=50, tol=0.01, verbose=0)
clf.fit(X_train, y_train)
end_time_ns = time.process_time_ns()
end_counter_ns = time.perf_counter_ns()

Y_pred = clf.predict(X_test)

accuracy = metrics.accuracy_score(y_test, Y_pred)
f1 = metrics.f1_score(y_test, Y_pred, average='weighted')
cpu_time = (end_time_ns - start_time_ns) / 1000000
wall_clock_time = (end_counter_ns - start_counter_ns) / 1000000
memory = process.memory_info().rss / 1000000

print(accuracy, ",", f1, ",", cpu_time, ",", wall_clock_time, ",", memory)
```

Table A1. FashionMNIST baseline benchmark for classical ML models and neural networks.

	Pi 4	Jetson TX2	Laptop	Client	Server
Accuracy					
Random Forest	0.850	0.850	0.849	0.850	0.849
Logistic Regression	0.847	0.847	0.847	0.847	0.847
Multi-Layer Perceptron	0.869	0.869	0.869	0.869	0.869
TensorFlow-FF	0.885	0.885	0.887	0.886	0.886
TensorFlow-CNN	0.911	0.913	0.913	0.913	0.913
F1 score					
Random Forest	0.847	0.846	0.845	0.847	0.846
Logistic Regression	0.846	0.846	0.846	0.846	0.846
Multi-Layer Perceptron	0.869	0.869	0.869	0.869	0.869
TensorFlow-FF	0.884	0.885	0.887	0.885	0.886
TensorFlow-CNN	0.911	0.913	0.913	0.913	0.913
CPU time (s)					
Random Forest	29.410	23.557	4.757	7.779	6.979
Logistic Regression	182.476	183.250	30.482	60.991	41.556
Multi-Layer Perceptron	297.405	158.070	123.099	42.404	966.927
TensorFlow-FF	840.535	455.269	74.562	122.200	187.498
TensorFlow-CNN	3550.677	1231.090	241.700	560.821	1466.845
Wall clock time (s)					
Random Forest	29.417	23.741	4.757	7.796	6.984
Logistic Regression	182.495	183.790	30.530	60.997	41.567
Multi-Layer Perceptron	74.555	39.965	7.791	10.675	15.146
TensorFlow-FF	409.170	257.432	33.184	58.599	61.631
TensorFlow-CNN	1031.345	390.357	38.865	186.812	133.312
Memory (MB)					
Random Forest	732.901	735.800	754.356	721.377	728.786
Logistic Regression	551.607	559.243	549.812	552.248	552.479
Multi-Layer Perceptron	552.424	560.461	551.177	554.729	551.650
TensorFlow-FF	1095.873	1100.167	1292.974	1220.498	1325.968
TensorFlow-CNN	1132.218	1139.928	1306.494	1256.246	1583.449

Table A2. CIFAR-10 baseline benchmark for classical ML models and neural networks.

	Pi 4	Jetson TX2	Laptop	Client	Server
Accuracy					
Random Forest	0.435	0.439	0.437	0.438	0.438
Logistic Regression	0.408	0.408	0.408	0.408	0.408
Multi-Layer Perceptron	0.100	0.100	0.100	0.100	0.100
TensorFlow-FF	0.355	0.362	0.361	0.346	0.368
TensorFlow-CNN	0.605	0.603	0.602	0.606	0.609
F1 score					
Random Forest	0.429	0.433	0.431	0.432	0.431
Logistic Regression	0.406	0.406	0.406	0.406	0.406
Multi-Layer Perceptron	0.018	0.018	0.018	0.018	0.018
TensorFlow-FF	0.336	0.348	0.344	0.328	0.352
TensorFlow-CNN	0.602	0.599	0.598	0.605	0.606
CPU time (s)					
Random Forest	36.854	30.244	6.681	9.345	10.229
Logistic Regression	1235.646	1761.439	251.896	423.916	324.770
Multi-Layer Perceptron	310.884	158.127	138.585	47.509	882.467
TensorFlow-FF	2966.710	972.642	233.662	293.732	1019.983
TensorFlow-CNN	4574.359	1557.796	243.585	520.132	2441.152
Wall clock time (s)					
Random Forest	36.861	30.435	6.681	9.359	10.234
Logistic Regression	1235.711	1765.620	252.234	424.183	324.862
Multi-Layer Perceptron	78.086	40.185	8.770	12.191	13.888
TensorFlow-FF	942.335	372.020	43.544	108.560	85.231
TensorFlow-CNN	1265.737	453.673	37.621	156.305	80.470
Memory (MB)					
Random Forest	1221.639	1224.502	1260.681	1208.017	1215.606
Logistic Regression	853.261	860.812	841.360	841.119	833.300
Multi-Layer Perceptron	856.884	865.985	841.692	841.385	845.087
TensorFlow-FF	1928.634	1946.936	2114.177	2068.620	2430.885
TensorFlow-CNN	1981.774	2013.309	2137.812	2115.784	2726.141

References

1. Yang, Q.; Liu, Y.; Chen, T.; Tong, Y. Federated Machine Learning: Concept and Applications. *ACM Trans. Intell. Syst. Technol.* **2019**, *10*, 1–19. [CrossRef]
2. Li, T.; Sahu, A.K.; Talwalkar, A.; Smith, V. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Process. Mag.* **2020**, *37*, 50–60. [CrossRef]
3. Li, Z.; Sharma, V.; Mohanty, S.P. Preserving Data Privacy via Federated Learning: Challenges and Solutions. *IEEE Consum. Electron. Mag.* **2020**, *9*, 8–16. [CrossRef]
4. Lyu, L.; Yu, H.; Yang, Q. Threats to Federated Learning: A Survey. *arXiv* **2020**. [CrossRef]
5. Konečný, J.; McMahan, H.B.; Yu, F.X.; Richtárik, P.; Suresh, A.T.; Bacon, D. Federated Learning: Strategies for Improving Communication Efficiency. *arXiv* **2016**, arXiv:1610.05492.
6. Nguyen, D.C.; Ding, M.; Pathirana, P.N.; Seneviratne, A.; Li, J.; Vincent Poor, H. Federated Learning for Internet of Things: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1622–1658. [CrossRef]
7. Mills, J.; Hu, J.; Min, G. Communication-Efficient Federated Learning for Wireless Edge Intelligence in IoT. *IEEE Internet Things J.* **2020**, *7*, 5986–5994. [CrossRef]
8. Xu, J.; Glicksberg, B.S.; Su, C.; Walker, P.; Bian, J.; Wang, F. Federated learning for healthcare informatics. *J. Healthc. Inform. Res.* **2021**, *5*, 1–19. [CrossRef]
9. Antunes, R.S.; André da Costa, C.; Küderle, A.; Yari, I.A.; Eskofier, B. Federated Learning for Healthcare: Systematic Review and Architecture Proposal. *ACM Trans. Intell. Syst. Technol.* **2022**, *13*, 1–23. [CrossRef]
10. Nguyen, D.C.; Pham, Q.V.; Pathirana, P.N.; Ding, M.; Seneviratne, A.; Lin, Z.; Dobre, O.; Hwang, W.J. Federated Learning for Smart Healthcare: A Survey. *ACM Comput. Surv.* **2022**, *55*, 1–37. [CrossRef]
11. Yang, W.; Zhang, Y.; Ye, K.; Li, L.; Xu, C.Z. FFD: A Federated Learning Based Method for Credit Card Fraud Detection. In Proceedings of the Big Data–BigData 2019, San Diego, CA, USA, 25–30 June 2019; Chen, K., Seshadri, S., Zhang, L.J., Eds.; Springer: Cham, Switzerland, 2019; pp. 18–32.
12. Long, G.; Tan, Y.; Jiang, J.; Zhang, C. Federated Learning for Open Banking. In *Federated Learning: Privacy and Incentive*; Yang, Q., Fan, L., Yu, H., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 240–254. [CrossRef]

13. Byrd, D.; Polychroniadou, A. Differentially Private Secure Multi-Party Computation for Federated Learning in Financial Applications. In Proceedings of the ICAIF '2: First ACM International Conference on AI in Finance, New York, NY, USA, 15–16 October 2020. [CrossRef]
14. Luo, B.; Xiao, W.; Wang, S.; Huang, J.; Tassiulas, L. Tackling System and Statistical Heterogeneity for Federated Learning with Adaptive Client Sampling. In Proceedings of the IEEE INFOCOM 2022-IEEE Conference on Computer Communications, Virtual, 2–5 May 2022; pp. 1739–1748. [CrossRef]
15. Zhang, J.; Li, M.; Zeng, S.; Xie, B.; Zhao, D. A survey on security and privacy threats to federated learning. In Proceedings of the 2021 International Conference on Networking and Network Applications (NaNA), Lijiang, China, 29 October–1 November 2021; pp. 319–326. [CrossRef]
16. Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J.; Blum, M.; Hutter, F. Efficient and Robust Automated Machine Learning. In Proceedings of the Advances in Neural Information Processing Systems 28 (2015), Montreal, Canada, 7–12 December 2015; pp. 2962–2970.
17. Olson, R.S.; Moore, J.H. TPOT: A tree-based pipeline optimization tool for automating machine learning. In Proceedings of the Workshop on Automatic Machine Learning. PMLR, New York, NY, USA, 24 June 2016; pp. 66–74.
18. LeDell, E.; Poirier, S. H2O AutoML: Scalable automatic machine learning. In Proceedings of the AutoML Workshop at ICML, Online, 18 July 2020; Volume 2020.
19. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. In Proceedings of the International Conference on Learning Representations (ICLR), San Diego, CA, USA, 7–9 May 2015.
20. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016.
21. Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A.A. Inception-v4, inception-resnet and the impact of residual connections on learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017.
22. Zhang, J.; Zhu, H.; Wang, F.; Zhao, J.; Xu, Q.; Li, H. Security and Privacy Threats to Federated Learning: Issues, Methods, and Challenges. *Secur. Commun. Netw.* **2022**, *2022*, 2886795. [CrossRef]
23. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the Artificial Intelligence and Statistics. PMLR, Ft. Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
24. Bonawitz, K.; Eichner, H.; Grieskamp, W.; Huba, D.; Ingerman, A.; Ivanov, V.; Kiddon, C.; Konečnỳ, J.; Mazzocchi, S.; McMahan, B.; et al. Towards federated learning at scale: System design. *Proc. Mach. Learn. Syst.* **2019**, *1*, 374–388.
25. Springenberg, J.T.; Dosovitskiy, A.; Brox, T.; Riedmiller, M. Striving for Simplicity: The All Convolutional Net. *arXiv* **2015**, arXiv:1412.6806.
26. Sattler, F.; Wiedemann, S.; Müller, K.R.; Samek, W. Robust and communication-efficient federated learning from non-iid data. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *31*, 3400–3413. [CrossRef]
27. Liu, Y.; Liu, Y.; Liu, Z.; Liang, Y.; Meng, C.; Zhang, J.; Zheng, Y. Federated Forest. *IEEE Trans. Big Data* **2022**, *8*, 843–854. [CrossRef]
28. Breiman, L.; Friedman, J.H.; Olshen, R.A.; Stone, C.J. *Classification and Regression Trees*; Routledge: New York, NY, USA, 1984.
29. Breiman, L. Bagging predictors. *Mach. Learn.* **1996**, *24*, 123–140. [CrossRef]
30. Li, Q.; Cai, Y.; Han, Y.; Yung, C.; Fu, T.; He, B. Fedtree: A fast, effective, and secure tree-based federated learning system. *arXiv* **2022**, arXiv:2210.00060.
31. Beutel, D.J.; Topal, T.; Mathur, A.; Qiu, X.; Fernandez-Marques, J.; Gao, Y.; Sani, L.; Li, K.H.; Parcollet, T.; de Gusmão, P.P.B.; et al. Flower: A Friendly Federated Learning Research Framework. *arXiv* **2020**. [CrossRef]
32. Mathur, A.; Beutel, D.J.; de Gusmão, P.P.B.; Fernández-Marqués, J.; Topal, T.; Qiu, X.; Parcollet, T.; Gao, Y.; Lane, N.D. On-device Federated Learning with Flower. *arXiv* **2021**, arXiv:2104.03042.
33. Ziller, A.; Trask, A.; Lopardo, A.; Szymkow, B.; Wagner, B.; Bluemke, E.; Nounahon, J.M.; Passerat-Palmbach, J.; Prakash, K.; Rose, N.; et al. Pysyft: A library for easy federated learning. In *Federated Learning Systems: Towards Next-Generation AI*; Springer International Publishing: Cham, Switzerland, 2021; pp. 111–139.
34. Truong, A.; Walters, A.; Goodsitt, J.; Hines, K.; Bruss, C.B.; Farivar, R. Towards Automated Machine Learning: Evaluation and Comparison of AutoML Approaches and Tools. In Proceedings of the 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), Portland, OR, USA, 4–6 November 2019; pp. 1471–1479. [CrossRef]
35. Ferreira, L.; Pilastri, A.; Martins, C.M.; Pires, P.M.; Cortez, P. A comparison of AutoML tools for machine learning, deep learning and XGBoost. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Virtual, 18–22 July 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–8.
36. Seng, J.; Prasad, P.; Dhami, D.S.; Kersting, K. HANF: Hyperparameter And Neural Architecture Search in Federated Learning. *arXiv* **2022**, arXiv:2206.12342.
37. Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J.T.; Blum, M.; Hutter, F., Auto-sklearn: Efficient and Robust Automated Machine Learning. In *Automated Machine Learning: Methods, Systems, Challenges*; Hutter, F., Kotthoff, L., Vanschoren, J., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 113–134. [CrossRef]
38. Feurer, M.; Eggensperger, K.; Falkner, S.; Lindauer, M.; Hutter, F. Auto-sklearn 2.0: Hands-free automl via meta-learning. *J. Mach. Learn. Res.* **2022**, *23*, 11936–11996.

39. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
40. Lindauer, M.; Eggensperger, K.; Feurer, M.; Biedenkapp, A.; Deng, D.; Benjamins, C.; Ruhopf, T.; Sass, R.; Hutter, F. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *arXiv* **2022**, arXiv:2109.09831.
41. Knowles, J. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Trans. Evol. Comput.* **2006**, *10*, 50–66. [CrossRef]
42. Wang, L.; Xu, S.; Wang, X.; Zhu, Q. Addressing Class Imbalance in Federated Learning. *Proc. AAAI Conf. Artif. Intell.* **2021**, *35*, 10165–10173. [CrossRef]
43. Yao, A.C. Protocols for secure computations. In Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), Chicago, IL, USA, 3–5 November 1982; IEEE: Piscataway, NJ, USA, 1982; pp. 160–164.
44. Ben-Efraim, A.; Lindell, Y.; Omri, E. Optimizing semi-honest secure multiparty computation for the internet. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 578–590.
45. Li, K.H.; de Gusmão, P.P.B.; Beutel, D.J.; Lane, N.D. Secure Aggregation for Federated Learning in Flower. In Proceedings of the DistributedML '21: 2nd ACM International Workshop on Distributed Machine Learning, New York, NY, USA, 7 December 2021; pp. 8–14. [CrossRef]