IoT

MDPI

Article

FedMon: A Federated Learning Monitoring Toolkit

Moysis Symeonides [1], Demetris Trihinas [2,*] and Fotis Nikolaidis [3]

[1] Department of Computer Science, University of Cyprus, Nicosia CY-2109, Cyprus; symeonidis.moysis@ucy.ac.cy
[2] Department of Computer Science, School of Sciences and Engineering, University of Nicosia, Nicosia CY-2417, Cyprus
[3] SuperDuperDB GmbH., 10179 Berlin, Germany; fotis@superduperdb.com
* Correspondence: trihinas.d@unic.ac.cy

Abstract: Federated learning (FL) is rapidly shaping into a key enabler for large-scale Artificial Intelligence (AI) where models are trained in a distributed fashion by several clients without sharing local and possibly sensitive data. For edge computing, sharing the computational load across multiple clients is ideal, especially when the underlying IoT and edge nodes encompass limited resource capacity. Despite its wide applicability, monitoring FL deployments comes with significant challenges. AI practitioners are required to invest a vast amount of time (and labor) in manually configuring state-of-the-art monitoring tools. This entails addressing the unique characteristics of the FL training process, including the extraction of FL-specific and system-level metrics, aligning metrics to training rounds, pinpointing performance inefficiencies, and comparing current to previous deployments. This work introduces FedMon, a toolkit designed to ease the burden of monitoring FL deployments by seamlessly integrating the probing interface with the FL deployment, automating the metric extraction, providing a rich set of system, dataset, model, and experiment-level metrics, and providing the analytic means to assess trade-offs and compare different model and training configurations.

Keywords: federated learning; machine learning; edge computing; internet of things

Citation: Symeonides, M.; Trihinas, D.; Nikolaidis, F. FedMon: A Federated Learning Monitoring Toolkit. *IoT* **2024**, *5*, 227–249. https://doi.org/10.3390/iot5020012

Academic Editors: Arun Ravindran and Reshmi Mitra

Received: 29 February 2024
Revised: 5 April 2024
Accepted: 9 April 2024
Published: 11 April 2024

1. Introduction

Data combined with today's advancements in machine learning (ML) are key ingredients to enable the rapid design and growth of innovative products and services; while vast volumes of data are now being produced outside of data centers (e.g., healthcare, IoT), the traditional notion of consolidating all data into centralized locations for ML model training faces challenges due to regulatory constraints (e.g., EU GDPR (https://gdpr-info.eu/ (accessed on 1 April 2024)), US CCPA (https://theccpa.org/(accessed on 1 April 2024))), consumer distrust, and the short-lived nature of data (e.g., temperature data) [1]. Federated learning is a promising paradigm challenging these barriers, where multiple clients collaborate in the training of an ML model without the exposure and sharing of sensitive data (i.e., biosignals, health records) during the learning process [2]. This makes FL an appealing paradigm for applications in healthcare, financial services, IoT [3], as well as new emerging services for augmented reality and human-driven virtual twins [4]. Hence, a key benefit of FL adoption is that all client data remain localized and only model parameterization is exchanged with the FL server entrusted with the overall management of the training process and interim model updating [5]. FL is appealing for IoT applications, as the data are processed where they originate, thereby giving "breathing space" to the compute continuum that can be overwhelmed by the volume of data consumed for model training [6].

With the advent of open-source frameworks such as Flower [7], OpenFL [8], and FATE [9], AI practitioners are now equipped with high-level programming abstractions to swiftly design and deploy highly specialized FL setups focusing solely on the model

building (business logic) and leaving the complexity that comes with the coordination of the distributed training process to the entrusted FL framework [10]. Nonetheless, while experimenting and deploying a small-scale FL cluster can be a (much) easier endeavor with these tools, scaling to hundreds and thousands of endpoints requires careful optimization to limit performance inefficiencies, model losses, and costs.

The first step to optimize any digital service (including FL) is to assess the current situation through a rigorous monitoring process measuring key performance indicators from probing interfaces attached to the deployments' stress points [11]. However, monitoring, and observability in general, can easily overwhelm AI practitioners due to the complexity of the monitoring instrumentation and analysis. For example, the majority of the open FL frameworks provide metrics relevant to the trained model (i.e., model loss) and some for the experiment run (i.e., duration), but there is a consistent absence of system-level insights (i.e., CPU usage, memory footprint, network latency). In contrast, there are a plethora of system monitoring tools, and, although one can configure the metric temporal periodicity, no tool tailors metrics on a per-training-round basis for the participating clients and their distribution across the deployment [12]. This establishes a configuration nightmare where AI practitioners are sidelined by their key focus. This often includes homogenizing metrics extracted from different sources and calibrating and aligning the insights to the time granularities that make sense for the FL paradigm, not only to evaluate the current emitted model, but also to compare against previous and future model versions.

This brings us to the focal point of our work. In this article, we introduce FedMon https://github.com/unic-ailab/FedMon (accessed on 1 April 2024), a toolkit for AI practitioners designed to ease the burden of monitoring FL deployments by seamlessly integrating the probing interface with the FL deployment, automating the metric extraction, and providing the analytic means to assess trade-offs and compare different model and training configurations. FedMon is equipped with a rich set of system-, dataset-, model-, and experiment-level metrics collected at different (and configurable) periodicities that can be reported at a per-round, client, and experiment basis, so that no manual (and extremely time-consuming) effort is lost in calibrating metrics to specific stages of the federated learning process. In turn, FedMon provides a client-side library that allows users to perform post-experimentation exploratory data analysis (e.g., box plots, histograms, etc.) on the combined dataset(s) of FL trial metrics. The library keeps metrics in tabular-based structures (dataframes), which can be transformed and stored in well-adopted standards, like CSV files, for future analysis.

The rest of this article is structured as follows: Section 2 provides a brief overview of the federated learning paradigm. Section 3 highlights our motivation for designing a monitoring tool for FL deployments. Section 4 sheds light on the design of FedMon, while Section 5 elaborates on implementation details for FedMon. Section 6 presents a comprehensive experimentation showcasing the wide applicability of FedMon, while Section 8 concludes the article and outlines future directions.

2. Background

Algorithm 1 provides a high-level overview of the federated learning paradigm. A typical FL experiment features a large pool of *clients*, where a number of these are selected to train, in a distributed fashion, an ML model under the supervision of an entrusted and central *server*. Under this modality, the central *server* first trains an initial version of the model, denoted as $W \in \mathbb{R}^n$, where n is the dimensionality of the parameter space. This is considered the starting point prior to the distributed learning process ①. Next, the *server* obtains a set of *clients* from a pre-defined client pool $(c_1, \ldots, c_k)$, with each *client* meeting certain criteria (i.e., resource availability) and, subsequently, broadcasts to the clients a training program and an initial model parameterization $\mathbf{W}_{t_0}$, with t_0 denoting the initial training round ②. At this point, each *client* updates the model locally, $\mathbf{w}_{c_i,t_1}$, based on local knowledge without exchanging data among themselves ③. The amount of samples used during local training can differ per client. When finished, the server collects an aggregate of the client updates, creating

a new global model $\mathbf{W}_{t_1}$ ④. This process is repeated for several rounds until the *server* determines that a termination criterion is met. This may be a maximum number of rounds or the convergence to a certain model loss for early termination.

Algorithm 1 High-level and abstract overview of the FL model training paradigm

Input: Training rounds T, clients C, local training epochs E and batch size B, and initial dataset D
Output: Trained model W_T
Ensure: Central `server` is running
1: $W_0 \leftarrow$ `initialize(D)`
2: **for** `each training round` t `in` T **do**
3: $K \leftarrow$ `ClientSelection(C)`
4: **for** `each` c_k `in` K **in parallel do**
5: $w_{t,c_k} \leftarrow$ `ClientUpdate`(W_{t-1}, E)
6: **end for**
7: $W_t \leftarrow$ `Aggregate`$(w_{t,c_1}, \ldots, w_{t,c_K})$
8: **end for**
9: **return** W_T

The most common implementation of the FL algorithmic paradigm is the FedAvg [2]. This algorithm is often considered the baseline for FL. Figure 1 provides an illustrative view of FedAvg. In a nutshell, the clients participating in each training round are selected at random under a uniform distribution. Then, during the parallel `ClientUpdate()` process, each *client* goes through E epochs of Stochastic Gradient Descent (SGD) with a batch size of B, where the local model weights are updated to optimize the model loss based only on client samples. At the end of the round, the derived model weights are collected per client by the server. Aggregation is then performed using a weighted average, where s_k is the number of samples used by each client during local training and $S = \sum_k s_k$:

$$W_t = \sum_k \frac{s_k}{S} w_{t,c_k} \tag{1}$$

For FedAvg, clients that have used more samples during the training process have a larger influence on the new state of the model. This criticism has led to different implementations of the FL algorithmic process. For instance, FedProx [3] is a generalization of FedAvg where the clients extend the SGD process so that clients optimize a regulated loss with a proximal term that enforces the local optimization of the loss in the vicinity of the global model per training round.

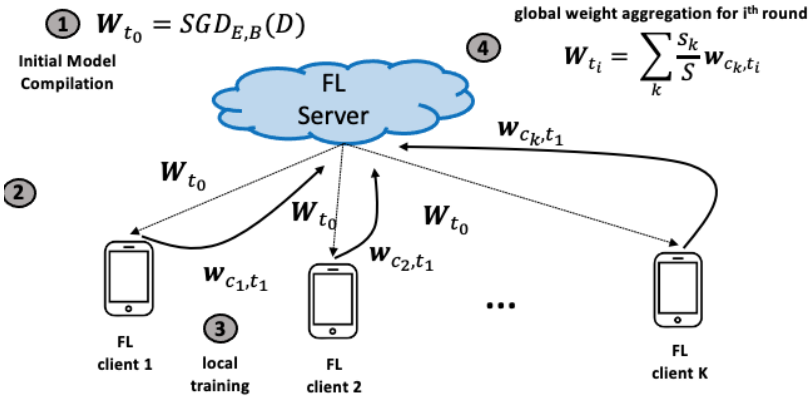


Figure 1. High-level overview of federated learning employing the FedAvg algorithm. At first, a global model $\mathbf{W}_{t_0}$ is prepared by the server (step 1) and disseminated to the clients selected for the first training round t_0 (step 2), the clients update their local model state at t_1 (step 3), and, afterward, the central FL server employs a weighted mean to infer a new global model state $\mathbf{W}_{t_1}$ (step 4). This process is repeated for a number of rounds or until convergence to a certain model loss.

3. Motivation

This section elaborates on the necessity of designing a monitoring service to obtain continuous insights at various levels of federated learning experiments deployed across the compute continuum.

3.1. Challenge 1: Multi-Level Instrumentation

The optimization of a federated learning deployment, by nature, is a multi-faceted problem. In particular, FL systems usually employ a vast array of clients distributed across the compute continuum, where resource heterogeneity is not the exception, but, rather, the norm [6]. In turn, resorting to FL usually entails the distributed training of a complex model (millions of parameters) with vast volumes of data scattered across the client realm [5]. Hence, the computational performance is often a key factor attributing to the scalability of the FL system [13], while the dynamics of both the training data and the selected model configurations are crucial to achieving a well-performing system [14]. However, most of the research in FL is heavily focused on the predictive quality of the trained global model (i.e., model loss), ignoring the impact that the aforementioned components play in actually achieving a performant system at scale. Hence, for AI practitioners employing FL systems in production, as shown in Figure 2, it is important to monitor the following:

- *The provisioned resources utility*, including metrics such as CPU usage, memory footprint, and network latency.
- *The input dataset*, including metrics such as drift, skewness, and staleness, at both a local and global level.
- *The output model*, including metrics such as model loss, robustness, and bias with the reporting at a per-training-round level.
- *Federated training run*, including metrics such as the duration of the experiment, breakdown of training rounds and time, client contribution, costs, and failures.

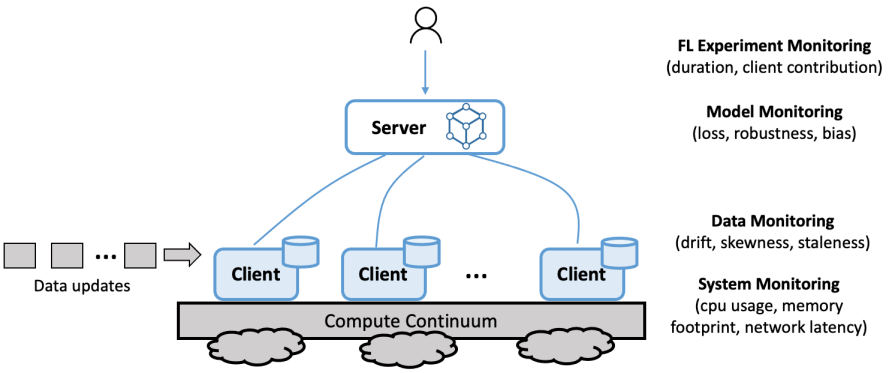


Figure 2. Multiple levels of monitoring instrumentation are required for federated learning.

3.2. Challenge 2: Training Round Temporal Granularity

To date, system-level monitoring tools (i.e., Netdata [15], JCatascopia [11]) provide admins with the ability to customize the granularity of the monitoring collection, either for all metrics or on a per-metric (group) basis. Doing so allows for metrics with higher volatility (i.e., CPU usage) to be assessed at a finer granularity so that key events are not reported latent (or even completely missed). In turn, for less volatile metrics, the reporting at a higher temporal periodicity eases the intensity of the monitoring process and its imposed overhead [16]; while a level of efficiency can be achieved through the fine-tuning of the (per-)metric intensity, in FL, the reporting granularity can significantly differ. In particular, AI practitioners, undertaking the assessment of the different components of an FL experiment run, perform their investigation on a per-training-round basis [17]. Hence, despite resource utility data collection being performed at a second/minute granularity, these must be aggregated and reported per round, just like model quality metrics such as loss and entropy. This is a key problem for FL monitoring (from a user perspective), as

simply looking at a single datapoint value (i.e., CPU 70%) at the end of a training round that may span several minutes can be misleading and hide several events occurring within that timeframe. Moreover, the current state of the art in FL/ML monitoring tools is solely focused on model quality metrics [14]; thus, to extract meaningful system-level insights, one must keep track of when each round has started/ended and, next, manually aggregate and report the measurements for each training round.

3.3. Challenge 3: High Churn Rate

In the FL paradigm, there are two clear and distinct entities. Clients perform the actual data munging to output model weights by utilizing their local dataset, while the central server is entrusted with the experiment coordination and the per-round weight aggregation to form the new global model. As these roles clearly present a different behavior, so should the monitoring process. This entails different metrics, collection intensities, as well as obtaining knowledge of the experiment topology. Regarding the latter, one may view/present the FL clients as a "flat" horizontal arrangement. However, FL significantly differs from paradigms such a distributed data processing (i.e., MapReduce, DataFlow), where the processing entities talk to each other and exchange data to achieve the desired goals. In FL, clients maintain all their (private) data locally and only exchange model parameterization and coordination messages exclusively with the server, and remain ignorant of the existence of their (nearby) peers in the federated network. What is more, during an FL experiment, the clients are selected on a per-round basis, and, therefore, may be different from round to round [18]. With this said, the monitoring process should follow a similar pattern in both the collection and reporting of metrics, so that the privacy aspect of the FL paradigm is not poisoned through the exposure of the monitoring data.

3.4. Challenge 4: Cross-Experiment Correlation

ML/FL models in production are never trained and left in production forever. On the contrary, models are retrained to ensure consistency and model performance with data from the current production environment and conditions. The retraining can be periodic or triggered when significant data distribution shifts are observed through monitoring of the model and data at runtime (i.e., concept drift) [19]. Hence, FL experiment runs should not be considered a "one-off" process. The empirical nature of ML model building requires AI practitioners to experiment with multiple different configurations until the model quality thresholds are achieved [17]. The complexity of this challenge increases when also dealing with an FL realm, as one must also access local datasets, client availability, and contribution (among others) with the model quality being highly susceptible to the choice of experiment configurations [20]. Hence, AI practitioners must be able to utilize their monitoring tools to easily compare key metrics of the current outputted model vs. past model versions and assess various trade-offs from the different (temporal) experiment configurations.

4. FedMon Framework Overview

To effectively tackle the challenges inherent to FL observability, monitoring, and instrumentation, we introduce the FedMon framework. FedMon addresses the multi-level instrumentation challenge by providing a unified observability and monitoring system that captures metrics across the distributed ecosystem of FL clients and FL server. Specifically, it provides fine-grained monitoring observations for both system-level metrics, such as CPU usage and memory footprint, and FL-specific metrics, like model loss and accuracy, across all participants. Moreover, FedMon keeps the necessary metric details based on their volatility without overwhelming the monitoring storage with data from less volatile metrics by implementing diverse granularities in metric collection. Through this approach FedMon aggregates FL metrics on a per-round basis, aligning with the cyclic nature of FL training, eliminating the need for manual data collection and aggregation by users, and utilizes interval-based utilization metric collectors for the infrastructure-based metrics.

Additionally, FedMon seamlessly integrates metrics from both clients and the server, while handling lifelong learning endeavors by storing and organizing the captured metrics in files. These files can be examined by the users at any time, performing comparative data analysis across different data or model versions over time, thereby aiding in the continuous improvement of FL models. Through these capabilities, FedMon presents a comprehensive solution to the challenges of FL observability, ensuring scalable, efficient, and effective federated learning deployments.

Figure 3 depicts a high-level overview of the FedMon framework. The initiation of user engagement with our framework involves the instantiation of abstract classes from the FedMon library, specifically the **FedMon client** and the **FedMon server** class. Specifically, the FedMon server class is responsible for instantiating the FL server, while FedMon clients start the FL client's training process and subscribe the node to the server. We elected to extend the Flower framework's client and server classes, rather than creating a new FL framework from scratch, to focus solely on the creation of an extensive array of probes designed for real-time metric extraction. To this end, users are only required to implement essential FL clients' methods, including (1) *load_data*, for retrieving datasets from local storage; (2) *get_parameters*, to extract and share local model parameters with the centralized FL server; (3) *train*, which employs the *load_data* method for machine learning training; and (4) *test*, for assessing model accuracy using the *load_data* method. Similar to the Flower framework, FL server configuration involves parameters such as strategy, the number of training rounds, and round timeout.

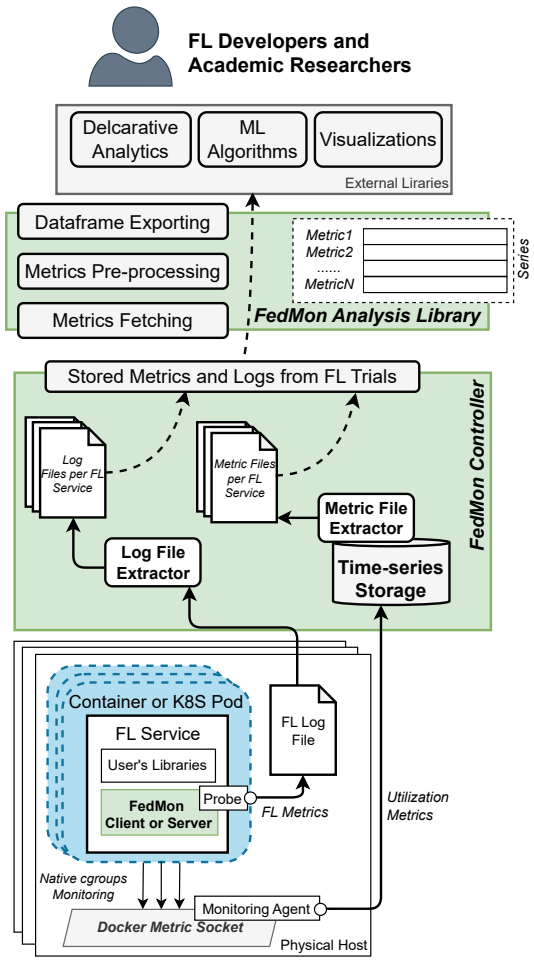


Figure 3. FedMon framework overview.

Furthermore, the FedMon framework automates the creation of Docker images for running FL services in a containerized environment, allowing us to extract service uti-

lization monitoring and system observability of the containerized environment. The only precondition for users is to assign a unique identifier to each container using the label *fedmon_identifier* and establish a shared folder for FedMon monitoring log files. Both *fedmon_identifier* and the shareable folder facilitate the aggregation and combination of FL node results.

At the runtime, **FL FedMon clients and their probes** gather FL-specific metrics such as *training duration*, *testing duration*, *local accuracy*, *data size*, *model parameter size*, and more (Table 1), storing this information in JSON format within the shared directory. Similarly, the **FL FedMon server** compiles overarching data at the conclusion of the FL process, detailing metrics like *overall model loss*, *per-round accuracy*, and *round duration*. Beyond FL-specific data, our tool also captures system-level metrics from the containerized FL environment. A **Monitoring Agent** exists on each cluster node and collects real-time system-level data, including *CPU utilization*, *memory usage*, *network I/O* metrics, etc. These data are periodically aggregated by a monitoring storage system and stored in the Time-Series Storage (Prometheus [21]).

Table 1. Representative FedMon metrics (non-exhaustive list).

Metric	Category	Description	Default Granularity
`CPU Utilization`	System-level	The CPU utilization of FL client or server	5 s (Configurable)
`Memory Utilization`	System-level	The memory utilization of FL client or server	5 s (Configurable)
`Network I/O`	System-level	The network data (both incoming and outgoing) of FL client or server in bytes	5 s (Configurable)
`Accuracy`	Model metric	The model accuracy overall or per client	Per round
`Loss`	Model metric	The model loss overall or per client	Per round
`Model Size`	Model metric	Number of parameters and size in MB of them for the model	Per round
`Round Duration`	Experiment metric	The overall round duration and per client	Per round
`Training Duration`	Experiment metric	The training duration per round and per client	Per round
`Testing Duration`	Experiment metric	The testing duration per round and per client	Per round
`Overall Duration`	Experiment metric	The overall FL duration	Per experiment
`Load Data Duration`	Dataset metric	Data loading duration per round and per client	Per round
`Data Size`	Dataset metric	The size of the client's dataset portion	Per experiment

To enhance our analysis, we integrated and correlated information from the following two distinct metric sources: (i) the **Time-Series Storage** containing system-level metrics, and (ii) the FL-performance metrics recorded in **FL log files**. Specifically, the Time-Series Storage exists on the FedMon Controller, and the FedMon framework extract the system-level utilization metrics from the time-series database via the **Metric File Extractor** after the FL experimentation. For the FL-performance log metrics, the FedMon Controller retrieves them from all clients using the **Log File Extractor**. These metrics are organized into files by the FedMon Metric File Extractor, each corresponding to an FL node, using the *fedmon_identifier*.

For post-experimentation analysis, the **FedMon Analysis Library** enables users to process monitored data by simply providing parameters such as *fedmon_identifier* of FL client nodes and FL trial name. The library fetches and preprocesses the raw data in order to create dataframes, presenting each metric as an in-memory time-series vector. This format allows for the application of data science tools and libraries for interactive analysis, visualization, and further exploration of FL performance metrics.

5. Implementation Details

5.1. System Metrics and Monitoring Stack

FedMon introduces a sophisticated and transparent monitoring stack designed to capture a comprehensive array of system utilization metrics from the evaluated infrastructure. These metrics include CPU utilization, memory consumption, network bandwidth usage, etc. The core of the monitoring system deployment is a containerized monitoring agent, deployed on each node within the cluster. This agent employs a highly efficient probe-based, multi-threaded approach, as described in [11]. Specifically, we use the cAdvisor [22] as a FedMon system-level monitoring agent, which systematically collects critical performance data from the underlying operating system processes (containers). It achieves this by launching targeted probes tailored to each distinct system sub-component, e.g., cgroup probe for container metrics. This monitoring agent exposes a dedicated HTTP endpoint, which is seamlessly integrated with monitoring storage servers, like Prometheus [21]. Prometheus, a state-of-the-art monitoring server, is configured to periodically query this endpoint, enabling it to efficiently harvest and archive the gathered metrics within its robust time-series database infrastructure. The periodicity of the monitoring agent and storage server is configurable by the FL practitioner, who is responsible for selecting the best granularity for the utilization metrics. In this subsequent phase, the Prometheus-integrated FedMon Metric File Extractor requests the stored performance metrics from Prometheus. Utilizing these data, the extractor generates structured monitoring CSV files. These files encapsulate the extracted knowledge of the raw performance metrics, transforming them into a more accessible and analyzable format. Once generated, these files are stored on the FedMon server, ensuring their availability for post-experimentation in-depth analysis. This comprehensive approach to monitoring and data management underscores FedMon's capability to delivering a robust, transparent, and highly effective performance monitoring solution, tailored to meet the needs of complex FL system evaluation scenarios.

5.2. Client and Server FL Training Metrics

Except for system-level metrics, FedMon also focuses on FL-related metrics. To collect client-specific metrics, the FedMon client, running on each client node, instantiates a number of metric probes (one per metric group) to perform the actual instrumentation so datapoints are collected at the desired temporal granularity given by the user or otherwise left to default. These probes are instantiated (as threads) when a new training round starts and are notified to stop the instrumentation when the round is completed. This entire process is automated, without requiring manual effort from the user side, unless the user wants to create a new probe type for a metric not offered by FedMon. An example of a probe offered by FedMon is the data size probe, which counts how many datapoints (dataset rows) are used by the client during the round for local training. This is an important metric, as several FL aggregation algorithms (i.e., FedAvg) utilize this to weight the client contribution in the overall global aggregation. Upon completion of a training round, and, subsequently, the lifespan of a probe, the collected metric updates are propagated to a Profiler. A Profiler is a class introduced in FedMon, whose primary functionality is to record and maintain a comprehensive log of various internal FL-related metrics for each round, storing these metrics into files for future reference.

When the Profiler is initialized, it requires as input a logging file's name, which serves as a repository for the gathered data. Subsequently, the Profiler is employed as a decorator (in other words, a programming annotation) on selected methods to capture and document their execution time and the size of the data they produce. This process involves the Profiler capturing these critical metrics and storing them in a structured key-value format, commonly known as a dictionary, within the memory of each FL client. An illustrative example provided in the snippet in Figure 4, which demonstrates the Profiler's use in a prototype FL client developed under the FedMon framework. In this scenario, specific functions within the class, such as *load_data*, *get_parameters*, *train*, and *test*, are monitored. Further expanding on its utility, the class incorporates the training function within the fit

method of the Flower framework. This integration allows users to track and log essential machine learning metrics, such as accuracy or loss. These metrics are then systematically stored through the execution of a *store_metrics* function, which efficiently serializes the gathered data into a new JSON object within a JSON line file, ensuring a structured and accessible format for data analysis. If the FL server selects a particular FL client for model evaluation, the evaluation function is triggered. In the provided example, the user has opted to record key performance indicators, specifically, evaluation accuracy and loss, showcasing the Profiler's versatility in capturing a wide array of metrics. Moreover, for server-side metrics, the Profiler retrieves historical data from the Flower library, like loss and accuracy, consolidating this information into well-organized files. Finally, the Profiler tags each new entry in the JSON line file with the current timestamp. This functionality allows for the seamless aggregation and analysis of metrics based on specific time points, enhancing the interpretability and utility of the collected data in understanding the system's performance over time.

```
import Profiler from FedMon
import flower as fl

profiler = Profiler(path = "...")
class FedMonClient(fl.client.Client):

    @profiler
    def load_data(self):
        # user code for loading dataset

    @profiler
    def get_parameters(self):
        # user code for retrieving model parameters

    @profiler
    def train(self):
        train, test = self.load_data()
        # user code for training

    @profiler
    def test(self):
        train, test = self.load_data()
        # user code for model testing

    def fit(self, ...):
        results = self.train()
        profiler.log_metric("accuracy": results["accuracy"])
        profiler.log_metric("loss": results["loss"])
        ...
        profiler.store_metrics()
        return results

    def evaluate(self, ...):
        results = self.test()
        profiler.log_metric("evaluation_accuracy": results["accuracy"])
        profiler.log_metric("evaluation_loss": results["loss"])
        ...
        return results
```

Figure 4. Snippet of FedMonClient and Profiler.

5.3. Metrics Analysis Library

The FedMon Analysis Library extends beyond monitoring functionalities, presenting a specialized suite for the advanced analysis of monitoring data. This library can fuse data related to resource utilization and FL operations, sourced through the FedMon monitoring toolkit, to construct varied analytical perspectives. These perspectives span different temporal scales and FL nodes (both server and clients), formatted as accessible tabular structures for the user's convenience. Specifically, users simply indicate the directory containing FedMon's monitoring and log files. The library then efficiently processes these files, holding the data in memory for immediate access. It employs a data structure with two dictionaries for data organization, one for utilization and one for the rest of FL metrics,

linking them through common node ID keys. This setup facilitates the retrieval of detailed metrics in the form of Pandas dataframes, enabling users to query specific node metrics by their ID.

However, the basic metric extraction offers limited insights, failing to uncover potential correlations between utilization and FL metrics or effects among different nodes. Addressing this, the library introduces advanced functions to synthesize richer datasets. Users can, for instance, merge utilization metrics across client nodes into a unified dataset, enhancing comparative analysis. Moreover, the library supports the integration of utilization and FL metrics for individual clients, employing timestamp alignment and data merging strategies to ensure coherence. The FedMon Analysis Library refines the timestamps for both utilization and FL metrics by rounding them to the nearest time interval (e.g., second). Subsequently, it executes an outer join based on the time interval fields. It is important to highlight that null values may arise due to the system's method of recording utilization metrics at fixed intervals, e.g., every 5 s, in contrast to the FL metrics, which are logged at the end of each FL round. Following this, the library addresses any missing values resulting from the join, either by filling them in or, should the user prefer, omitting them entirely. Moreover, the process of filling provides diverse options, with the default option being to carry forward the last available value. Moreover, the library facilitates the aggregation of joined dataframes from individual FL clients and FL server into a comprehensive single dataframe. However, users should remain aware of potential performance implications associated with this process.

Finally, FedMon Analysis Library provides visualization capabilities, allowing users to derive meaningful insights without deep technical knowledge of the underlying processes, while it also supports exporting metrics in various formats, such as JSON and CSV.

5.4. User Interface

As the main user interface of our framework, we utilized Jupyter Notebooks. A Jupyter Notebook is an online interactive platform that facilitates data analysis, allowing users to insert code cells that are run by an external execution environment, or kernel. The outputs, such as plots, are shown immediately after the code runs. The widespread use of Jupyter Notebooks has led to the development of numerous alternative web interfaces. For example, JupyterLab provides a comprehensive set of features common in many Integrated Development Environments (IDEs), including code highlighting and editing, while still emphasizing interactive and exploratory computing. Acknowledging these properties, we created a containerized version of Jupyter Notebook in which we have pre-installed all necessary Python artifacts that the FedMon Analysis Library needs. To this end, users need to introduce the folder with the monitoring metrics into the container as a virtualized folder, and the FedMon-enabled Jupyter Notebook allows them to perform their analysis.

6. Experimentation

In this section, we highlight our usability evaluation with a representative Cloud–Edge continuum testbed. For the experimentation, we deployed FedMon on top of a bare metal cluster of 48 CPUs, all clocked at 2.45 GHz, and 176 GB RAM. We embraced the Fogify emulation framework [23] to replicate necessary network effects and enforce compute resource constraints over the requested realm of the FL client pool. Specifically, we deployed a number of different containerized FL clients (10 and 28) with various processing capabilities, depending on the different experiment descriptions.

6.1. Use Case Overview

6.1.1. Description

FedMon can find applicability in a wide range of FL settings to assess in the evaluation of the learning process and, ultimately, aid in resource capacity optimization. Examples include FL-based VANETs [24] and medical imaging classification [25]. As a use case, we introduce an edge-based distributed learning system that can be implemented to

enhance public services such as automated form processing, digital archiving, and real-time translation of handwritten documents. This use case leverages a network of IoT devices (e.g., smartphones, tablets, smart kiosks) equipped with cameras or touchscreens to capture handwritten inputs from users. By utilizing federated learning, the system trains a shared model to improve handwriting recognition accuracy without centralizing sensitive data. In this use case, the architecture comprises three main components: IoT devices as data collection points, edge nodes for local computation and model training, and a central server for model aggregation and updates. IoT devices capture handwritten data and perform initial preprocessing. Edge nodes located close to IoT devices facilitate the local model training using federated learning algorithms. These nodes handle ML tasks, reducing the need for constant cloud connectivity and addressing bandwidth and latency issues.

6.1.2. Dataset

For our analysis, we evaluated a well-known dataset in computer vision, namely, MNIST. It is a basic collection of black-and-white images of handwritten digits, comprising 60,000 training and 10,000 testing images, each 28 $\times$ 28 pixels. The database is categorized into 10 classes, each representing a digit from 0 to 9. MNIST's simplicity and small image size make it an ideal dataset for initial evaluations and prototype development of machine learning algorithms. It is particularly useful for testing various classification methods and benchmarking new model performances.

6.1.3. Models

For the MNIST dataset, we deployed a Convolutional Neural Network (CNN) using PyTorch and TensorFlow frameworks. This CNN model was structured with six layers that could be adjusted during training, including two 2D convolutional layers, two 2D dropout layers to prevent overfitting, and two fully connected linear layers. The model also incorporated layers that did not undergo training, such as the ReLU activation functions after each of the first two convolutional layers to introduce non-linearity, a MaxPool2D layer for down-sampling following the second ReLU activation, and a Log-Softmax layer for outputting the final probabilities. In addition to CNNs, we provided a linear regression model for the MNIST dataset, utilizing the SKlearn framework.

6.2. Exploratory Data Analysis

In our initial examination, we concentrated on a single FL deployment, aiming to study its performance metrics. Specifically, we deployed the TensorFlow CNN model on 28 FL clients with uniform capabilities (two cores at 2 GHz and 4 GB of memory). For our analysis, we utilized the FedMon library, employing its Analysis Library to facilitate our investigation. The goal of this use-case scenario is to showcase that FedMon is a suitable tool to address the C1 and C2 challenge vectors, where users are offered a plethora of system-, model- and experiment-level metrics without needing any manual (re-)configuration from their side. We began by examining the model's average accuracy and loss over each training round, as depicted in Figure 5. The results indicate that our model performs well, showing significant improvements in both accuracy and loss from the 10th round onwards, achieving an accuracy rate of 99% by the conclusion of the FL training. Subsequently, we shifted our attention to another key performance metric: the training and inference (test) duration. Figure 6 presents the average training and testing durations per round across all participating clients, revealing consistent durations across FL rounds, with training taking approximately sixty seconds and testing under ten seconds.

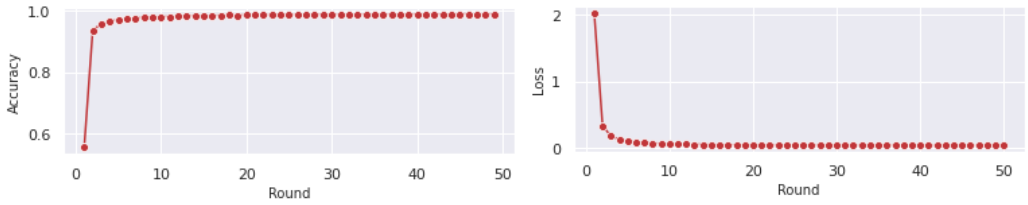


Figure 5. Accuracy (**left**) and loss (**right**) of the global model at the i-th round after the aggregation has been applied (e.g., weighted mean for FedAvg) with the evaluation/testing extracted from the underlying FL engine.

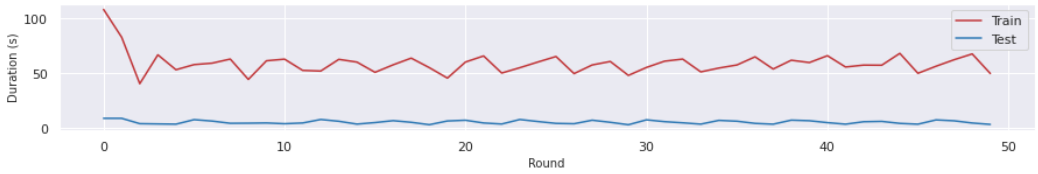


Figure 6. FL training and testing duration in seconds per round.

Further analysis was conducted on the temporal performance metrics, particularly focusing on the average CPU time and network I/O across five specific rounds of the FL process (rounds 22–27), as shown in Figure 7. This analysis reveals a decrease in CPU time by clients after the local training, followed by the central server's aggregation of client weight vectors. Significant network activity spikes mark the subsequent dissemination of the updated model by the server. Clients then resume local training, leading to a rise in CPU usage for the duration of the training. This pattern highlights the cyclical nature of FL, characterized by periodic network spikes and CPU usage variations. Notably, the figure indicates two network spikes, an observation not immediately deducible from theoretical models. The latter is attributed to the FL coordination process, wherein weight vectors are distributed twice: once for loss computation and again for initiating the next round.

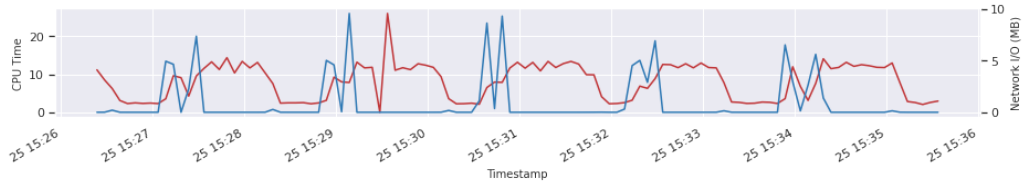


Figure 7. Average CPU time (left y-axis and red line) and the sum of network I/O in megabytes (MB) (right y-axis and blue line) from all FL clients for rounds 22–27.

Subsequently, our attention narrowed to an in-depth analysis of a specific round, particularly round 25 (Figure 8), which is the middle point of the sequence of FL rounds depicted in Figure 7. This examination aimed to highlight the individual contributions of each worker to the round's overall utilization metrics. In this detailed view, we initially detected the consistent presence of two distinct network spikes across all clients. This was followed by a steady increase in CPU time, due to the high computational demands during the FL training phase. What captured our interest further was the observation of varying CPU time fluctuations and notable spikes towards the end of this CPU-intensive phase across numerous FL clients. This shows a dynamic computational resource allocation, which could be attributed to diverse factors including the ML library's and FL framework's implementation, or, even, the complexity of the model updates being processed.

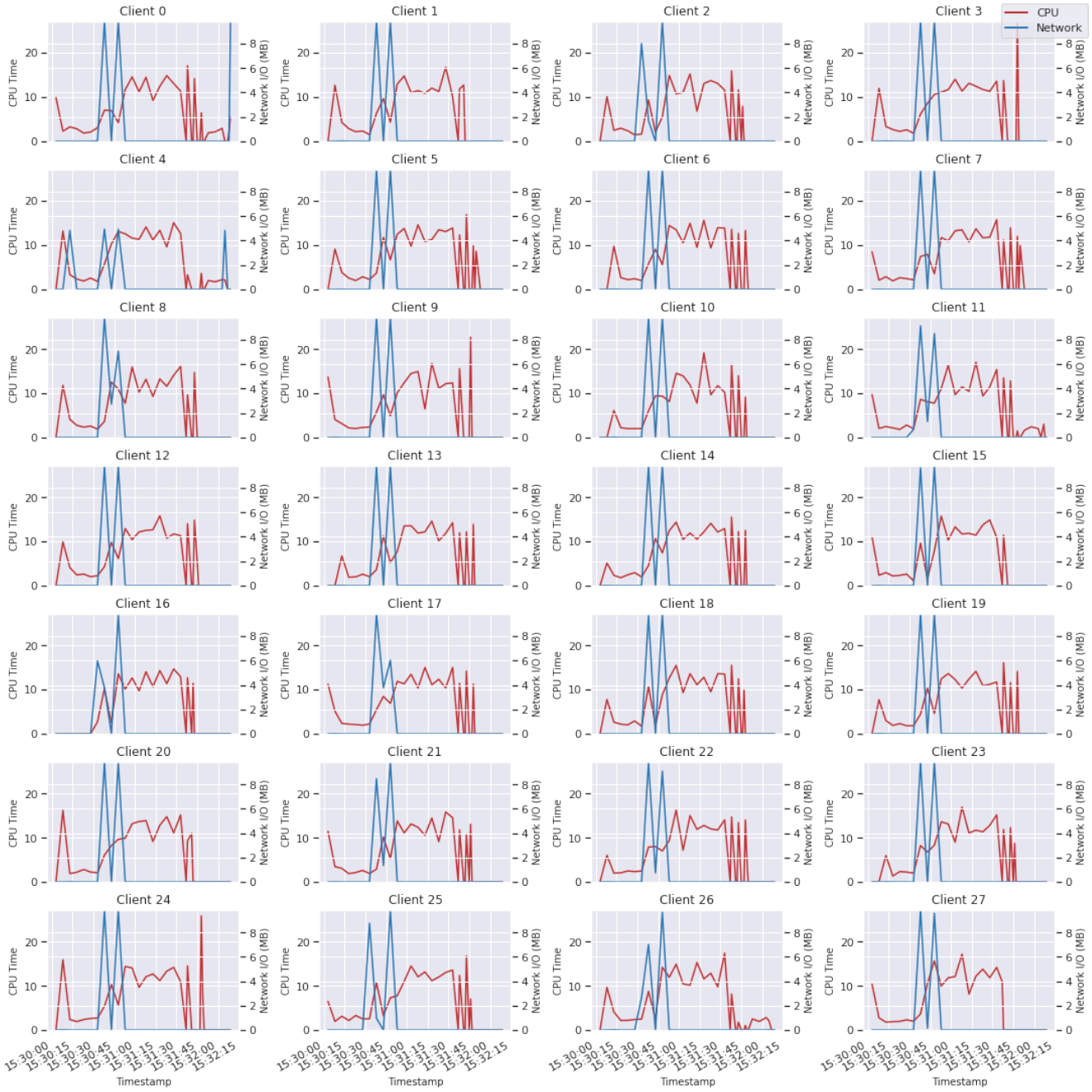


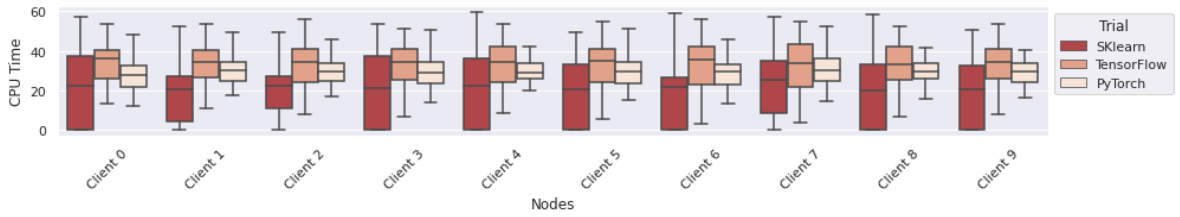
Figure 8. Average CPU time (left y-axis and red line) and the sum of network I/O in megabytes (MB) (right y-axis and blue line) for each FL client within round 25.

6.3. Comparing Different Configurations

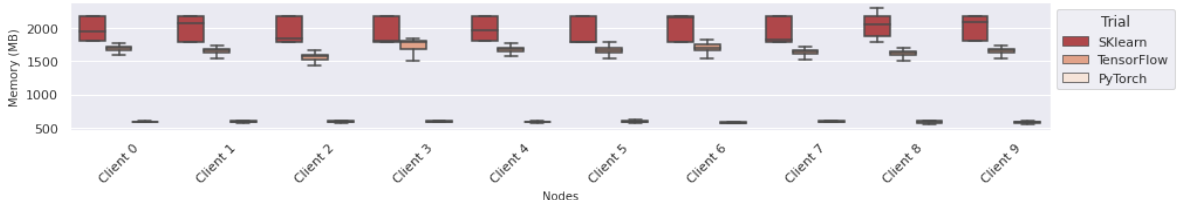
In this study, the focus was on a comprehensive evaluation and comparison of various configurations across multiple AI and ML libraries and models. With this experiment, we showed that the analytic insights provided by FedMon, address the need to compare multiple and different experiment configurations (C4) to find a suitable model structure that meets the user requirements. The infrastructure included 10 FL clients with four cores at 2.4 GHz and a homogeneously distributeddataset among them (6000 non-overlapping records on each client). The experiment leveraged the CNN model from both PyTorch and TensorFlow, alongside the linear regression model from the SKlearn library. FedMon records all relevant metrics, encompassing both resource utilization and FL performance, enabling

the FL developer to select the most suitable framework and model tailored to her/his specific requirements.

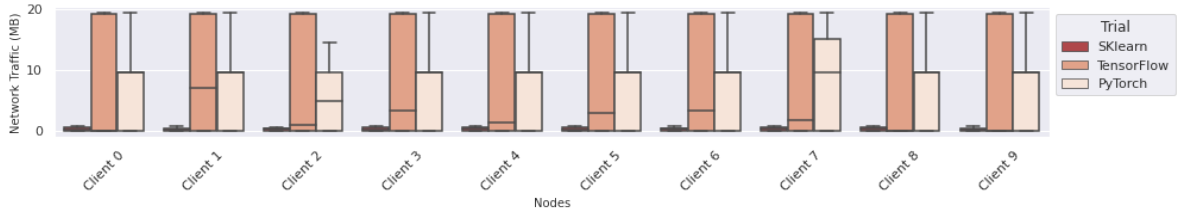
The subfigures of Figure 9 provide a detailed view of the system-level metrics through box plots. Specifically, Figure 9a presents a comparison of CPU time across each participant in the FL network, indicating the computational efficiency of each framework. The analysis reveals that the SKlearn implementation is notably more efficient in terms of CPU time utilization per interval throughout the FL process, in contrast to TensorFlow, which emerges as the most CPU-demanding framework, with PyTorch occupying an intermediate position. Furthermore, Figure 9b highlights the memory consumption patterns of each framework through a box plot visualization. A surprising observation is that SKlearn exhibits higher memory usage compared to both PyTorch and TensorFlow. This anomaly was traced back to SKlearn's implementation approach, which involves loading the entire dataset into the memory of each client prior to training. When comparing the memory footprints of TensorFlow and PyTorch, it was noted that PyTorch's processes generally require less memory, enhancing its suitability for FL clients with limited memory resources. The investigation also extended to the analysis of network traffic, encompassing both incoming and outgoing data across different deployments (Figure 9c). As anticipated, the SKlearn's linear regression model demonstrates lower network data transfer volumes, attributed to its fewer model parameters in comparison to the more complex neural network models. The network traffic patterns for PyTorch and TensorFlow are closely matched, although TensorFlow exhibits a wider range of data transfer volumes, indicating variability in network efficiency across different FL deployments.



(**a**) CPU time



(**b**) Memory usage in megabytes (MB)

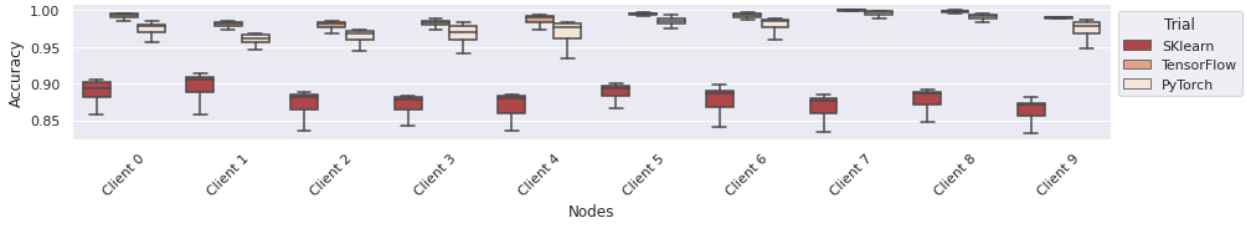


(**c**) Network traffic in megabytes (MB)

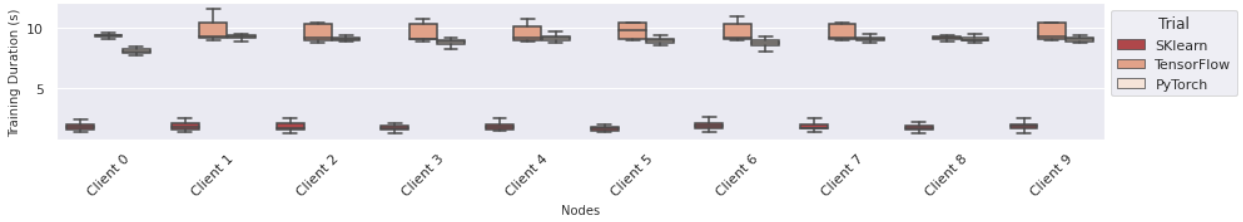
Figure 9. Summary of utilization metrics for FL trials.

In the subsequent phase of our study, we delved into the performance metrics derived from FedMon. This encompassed the accuracy, as well as the duration of training and testing phases for each FL participant across various trials, as depicted in Figure 10. Notably, the linear regression model consistently exhibits the lowest accuracy among all clients, specifically between approximately 85 and 90%, whereas clients employing deep learning models

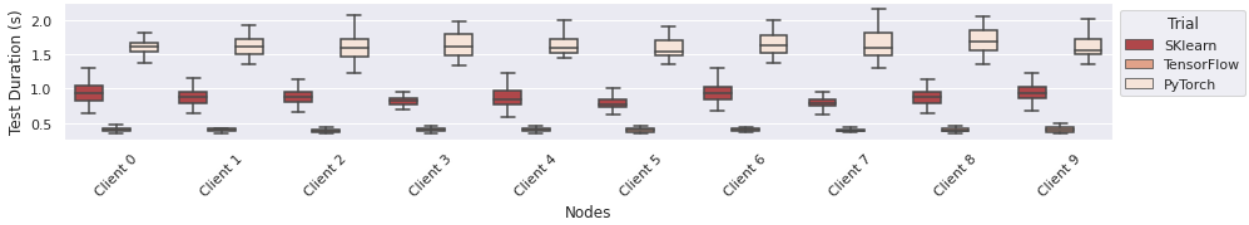
achieve accuracies exceeding 95%, as shown in Figure 10a. Since TensorFlow and PyTorch trained the same model with the same underlying dataset, the expected accuracy should be similar, with Figure 10a confirming that the two models present only a 1–2% difference. Further analysis was conducted on the training durations for each round, illustrated in Figure 10b. Here, the SKlearn linear regression model has the shortest training duration at around 3 s per round. TensorFlow and PyTorch provide comparable performance metrics, though TensorFlow experiences a slight additional latency in the training process. The examination extends to the test (inference) durations for each round across all clients, as highlighted in Figure 10c. In this aspect, TensorFlow's FL client implementations outperform those of SKlearn and PyTorch. This observation is particularly noteworthy given SKlearn's previously identified memory utilization inefficiencies, which appear to significantly impact the model's inference time.



(**a**) Accuracy



(**b**) Training duration in seconds



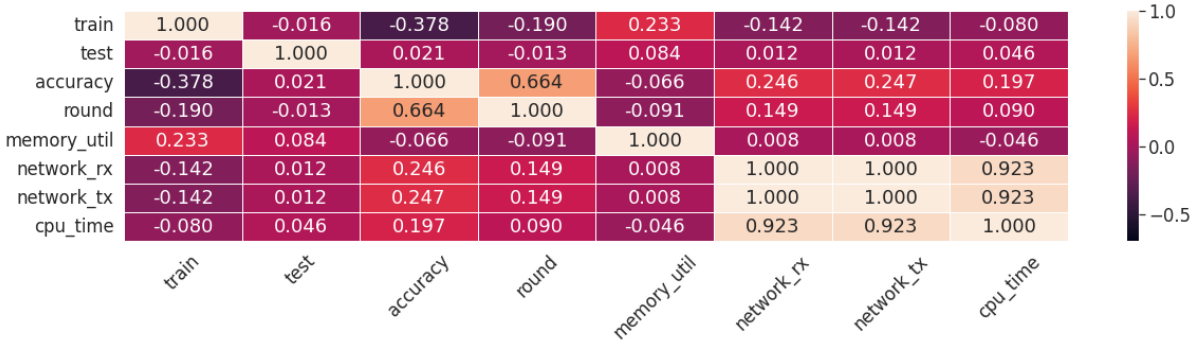
(**c**) Test duration in seconds

Figure 10. Summary of FL-related metrics for FL trials.

Figure 11 showcases the correlation matrices for the SKlearn, TensorFlow, and PyTorch FL trials, illustrating the complex relationships between various performance metrics within each framework. One of the primary insights is the strong correlation between the number of rounds and model accuracy across all frameworks, suggesting that a larger number of rounds is beneficial for the model's accuracy. Additionally, there is a notable symmetry in the correlation between incoming (*network_rx*) and outgoing (*network_tx*) network data, which might reflect the patterns of the TCP protocol.

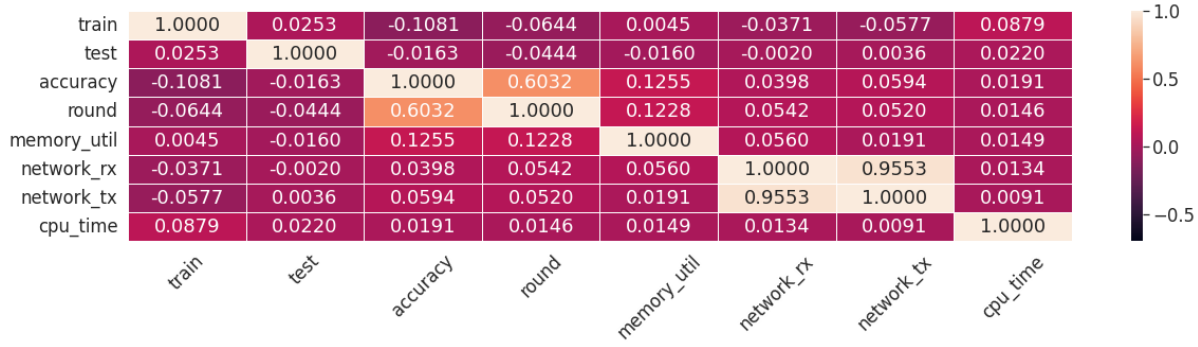
In the SKlearn framework, there is an observable relationship between model accuracy and network traffic, as well as between CPU time (*cpu_time*) and *accuracy*, indicating that more computational time may contribute to higher accuracy. Memory usage (*memory_util*), aka the utilization of memory for the specific timestamp, also shows a link with the duration of training (*train*), which could show that SKlearn memory over-subscription may also impact the training time. For the TensorFlow framework, a correlation is seen between memory usage and the number of rounds, potentially indicating inefficient memory

management or 'garbage' accumulation over multiple iterations. An inverse relationship between CPU time and network traffic is identified, contrasting with SKlearn's pattern. Finally, PyTorch framework's correlations align with the other frameworks regarding the positive link between accuracy and the number of rounds. However, PyTorch's metrics do not exhibit strong correlations with each other, suggesting a more independent relationship between the measured variables in this particular framework. Overall, these matrices provide a quantitative overview through which the operational characteristics of each FL framework can be understood, offering valuable insights into how each framework's performance metrics influence one another.



(**a**) Correlation matrix for SKlearn

(**b**) Correlation matrix for TensorFlow

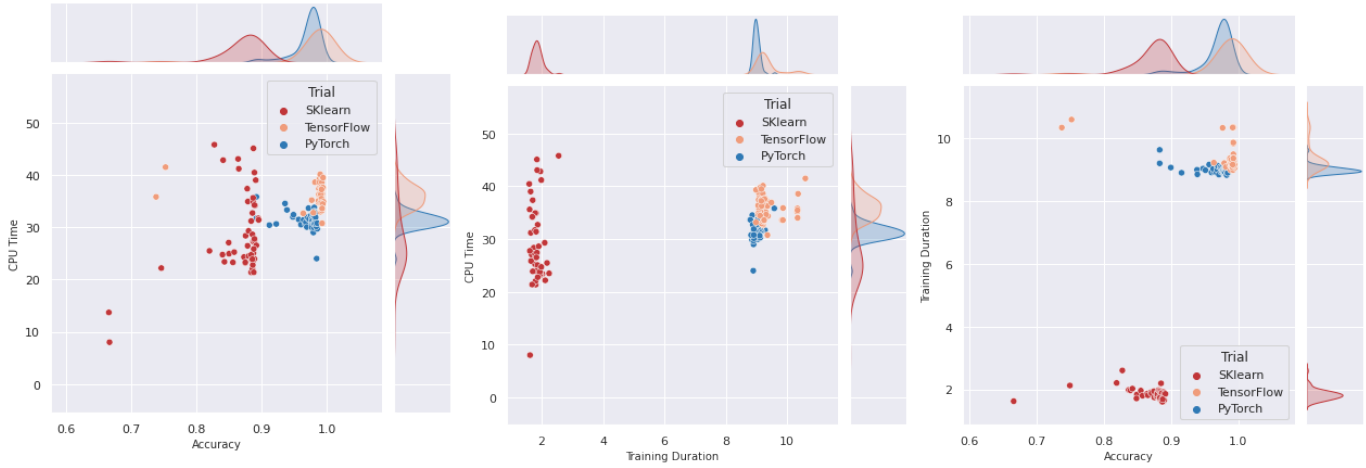


(**c**) Correlation matrix for PyTorch

Figure 11. Correlation metrics for each trial.

Next, in Figure 12, we delve into a detailed comparative analysis that spans various metrics and their respective distributions across different trials. This figure introduces a scatter plot for each pair of metrics under consideration, showcasing the interplay between them. For each variable of the scatter plot, there is a histogram, located on each axis, highlighting the distribution of each metric. Beginning with the comparison of CPU

time against accuracy, as depicted in Figure 12a, the datapoints representing TensorFlow and PyTorch trials are located in the same region, contrary to SKlearn's linear regression model, where it has decreased CPU usage but also limited accuracy. This observation is further underscored by the histograms on both the x- and y-axes, which reveal a normal distribution of the datapoints, but in different areas. Additionally, the relationship between CPU time and training duration is highlighted (Figure 12b). Here, it becomes evident that SKlearn consumes less time for its training process when compared to TensorFlow and PyTorch. This distinction could be important for applications where time efficiency is paramount. Lastly, the analysis shifts towards comparing training duration with accuracy (Figure 12c). This comparison unveils a contrast between the performances of the SKlearn linear regression model and the more complex CNN models employed by TensorFlow and PyTorch. Notably, while TensorFlow and PyTorch's CNN models display comparable outcomes in terms of accuracy, PyTorch has a slightly better training time. This difference might influence the choice between these frameworks based on the specific requirements of a project, such as the balance between accuracy and training efficiency.



(**a**) Accuracy vs. CPU Time (**b**) CPU Time vs. Training Duration (**c**) Training Duration vs. Accuracy

Figure 12. Comparison of Diverse metrics of FL Trials.

6.4. Comparing Trials in Different Time Periods

This experiment run showed the usability of FedMon as a tool providing valuable insights in FL deployments with churn where either capacity and/or data dynamics of the client pool are skewed (C3). For this, we considered a scenario in which the characteristics of FL nodes (e.g., data distribution) may change over time. To simulate such a scenario, we initially considered the same deployment of TensorFlow as in the previous Section 6.3 for the *Homogeneous* trial, in which all nodes have the same resources and the same portion of the data. Then, we considered that the data distribution had shifted over time. So, system operators would like to retrain the model (TensorFlow CNN model) and observe the overhead in FL training on the overall performance. For the latter scenario, we distributed the dataset following the Pareto distribution for our second trial (*Heterogeneous*). After that, we performed a comparative analysis between the two trials.

Figure 13 illustrates the average duration of rounds across the deployments alongside the total network data traffic. It is clear that an uneven data distribution negatively impacts the mean round duration. In a synchronized FL system, the process must wait for the slowest participant. With an unequal data distribution, specifically a Pareto distribution, the node which holds the majority of data becomes the bottleneck, and, thus, extends the average duration of a round. However, the overall network data traffic remained relatively consistent across both experiments, which was expected, given that, in FL systems, data transfer primarily involves the sharing of model parameters between FL nodes and the server.

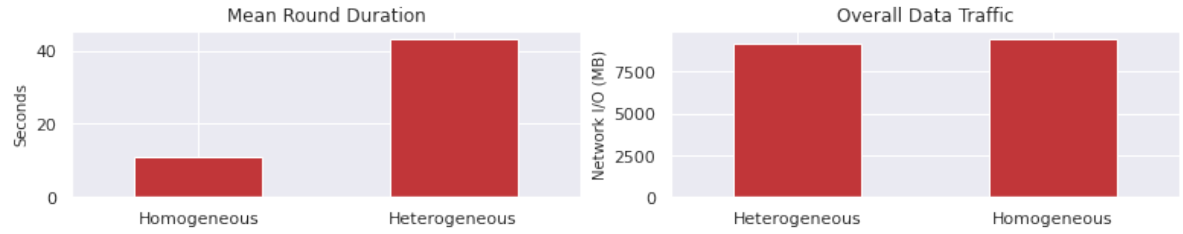
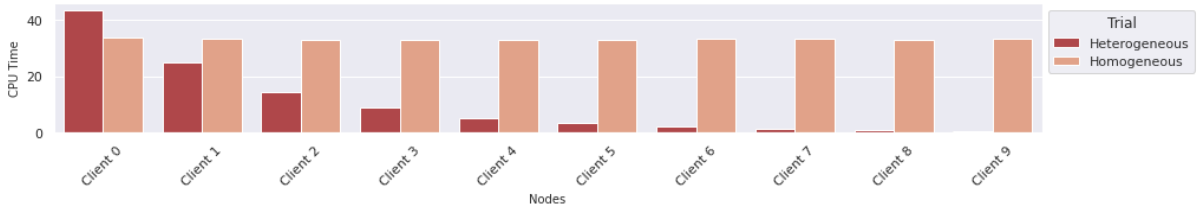
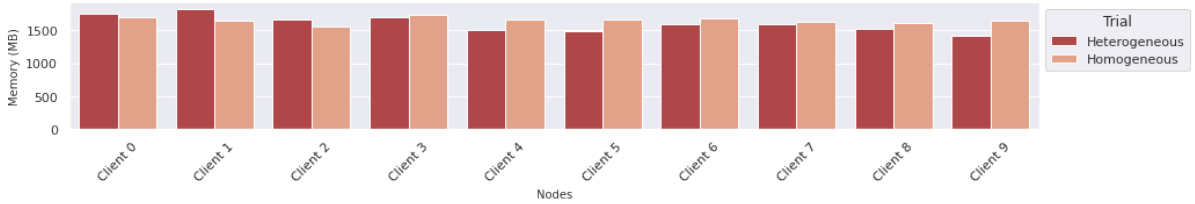


Figure 13. Per-round duration in seconds and overall network traffic in megabytes (MB).

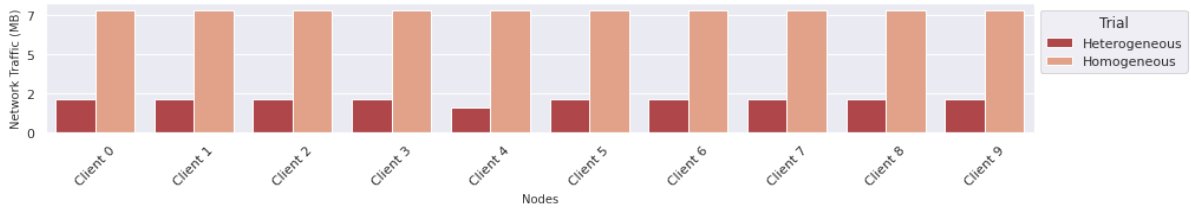
Subsequently, we focused on the metrics that highlight the utilization of the clients, as illustrated in Figure 14. A key finding from our analysis is the direct correlation between the CPU time on each node and the volume of data it processes (as seen in Figure 14a). To put this into perspective, Client 0 of the Heterogeneous trial keeps the majority of the data, and, at the same time, demands significantly more CPU resources than the other clients in the same deployment. This observation illustrates the uneven distribution of computational load among the clients in such a setup. On the other hand, a homogeneous data distribution ensures a more balanced demand for CPU resources across all clients participating in the FL process. Apart from this, the Homogeneous trial demonstrates better efficiency in terms of round completion times, highlighting a more effective use of resources.



(**a**) CPU time



(**b**) Memory utilization in megabytes (MB)



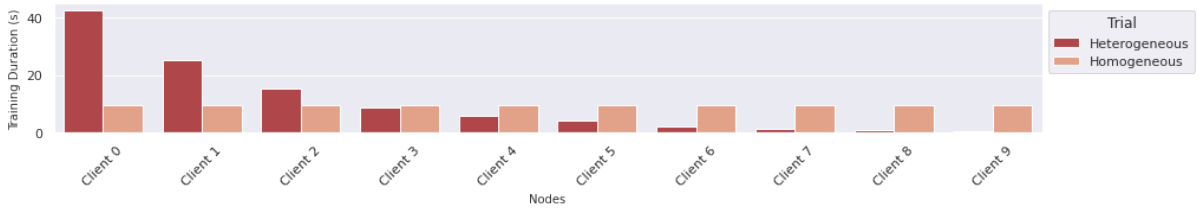
(**c**) Network traffic in megabytes (MB)

Figure 14. Resource utilization for heterogeneous and homogeneous data distribution.

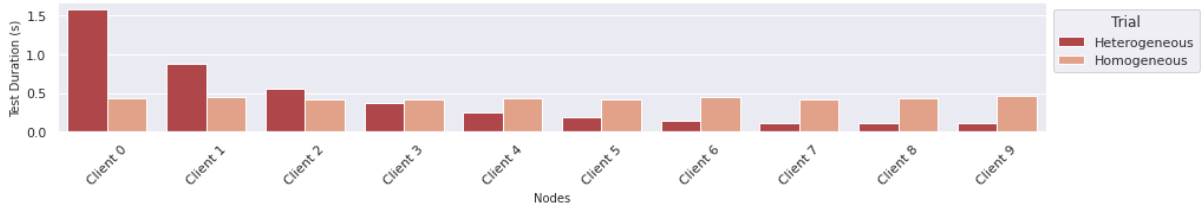
Furthermore, the comparison of memory usage between the two trials reveals a surprisingly negligible impact of data distribution on memory load (as depicted in Figure 14b). This illustrates TensorFlow's management of memory, likely through a mechanism that involves batching and sequential fetching of data from the storage, thereby optimizing memory usage regardless of the size of data partitions.

Shifting the focus to the aspect of network data transmission, Figure 14c provides insight into the average data traffic observed among the nodes over a specified monitoring period of 5 s. In scenarios involving the Heterogeneous trial (Pareto data distribution), there is a noticeable reduction in the volume of data exchanged per unit of time. This trend is logical when considering the duration of FL rounds and the overall execution time, along with the comparative analysis of total network traffic as depicted in Figure 13. Such findings suggest that, despite the differences in data distribution and CPU utilization, the network load remains relatively consistent.

Following this, Figure 15 provides a detailed representation of the durations associated with both training and testing phases for each round, demonstrating that these durations are in line with the distribution patterns of the data, as seen in Figure 15a,b. However, the CPU utilization observed in Figure 14a deviates from the expected proportional relationship between the Heterogeneous and Homogeneous trials. This deviation highlights a key insight: clients in the Homogeneous trial achieve more efficient CPU usage throughout the FL rounds. In contrast, clients involved in a Heterogeneous trial tend to be underutilized for significant portions of the FL process. The latter shows that while data distribution directly impacts the duration of training and testing rounds, it does not straightforwardly translate to CPU time efficiency. In the Homogeneous trial, the balanced distribution of data among clients allows for more consistent and effective utilization of CPU resources, contributing to overall efficiency in the FL rounds. On the other hand, the uneven data spread in the Heterogeneous trial leads to varying levels of CPU usage, with some nodes experiencing periods of underutilization, which could indicate inefficiencies in the FL workflow. This observation underlines the importance of considering not just the distribution of data, but also the optimization of resource utilization in the design and implementation of FL deployments.



(**a**) Training duration in seconds



(**b**) Testing duration in seconds

Figure 15. FL duration metrics for heterogeneous and homogeneous data distribution.

7. Related Work

The system monitoring landscape features numerous monitoring tools that span the compute continuum [26]. These include Nagios [27] and Zabbix [28] for the physical (bare metal) level, Netdata [15] and Ganglia [29] for virtual machines, and cAdvisor [22] and Prometheus [21] for containerized execution environments. In turn, several tools such as Datadog [30] and AppDynamics [31] are now performing application-level instrumentation and offering Monitoring-as-a-Service. Moreover, initiatives from the monitoring research community focus on monitoring agent placement [32], low-cost and adaptive metric collection [33], and advanced traceability and SLO configuration [34].

Recently, a number of tools have been designed to offer ML monitoring where the focus of the instrumentation is on the quality and robustness of the ML model and its underlying data dynamics [35]. Examples of ML monitoring tools are the commercial EvidentlyAI [36], SageMaker [37], and open-source WhyLabs [38]. Notable are also ModelDB and DeepChecks. In particular, the authors of ModelDB [17] advocate for the need for data-level monitoring of models to ensure that the offline and live data fed to a model are similar. In turn, DeepChecks [39] is an ML monitoring tool that, besides data integrity and drift tracing, enables model-to-model (version) comparison so that a user may view changes in the model performance between experiments; while the aforementioned constitute interesting tools to gain monitoring insights, none provide intelligence in the realm of resource consumption and system utilization. Most importantly, support for federated learning is not available.

The following are a handful of solutions attacking some of the aspects relevant to FL monitoring. Sun et al. [40] introduced a tool for FL capable of selecting an expected model structure by examining trade-offs among the training efficiency and model performance. To achieve this, a monitoring tool was developed and metrics for training time, network utility, accuracy, and inference latency were made available to users. Al Mallah et al. [12] introduced a framework designed explicitly for FL monitoring where metrics related to the data distribution of each client are extracted per training round in order to perform behavior pattern analysis to detect malicious behavior in the early stages of the learning process. In a similar fashion, Meng et al. [41] introduced VADAF, an FL service providing users with monitoring insights at a client level for the model loss, accuracy, and data heterogeneity, so that anomaly detection can be performed to finding abnormal client behavior. Taejoon et al. [42] introduced CARLA-FLMon, a monitoring and graphical plotting tool for FL that can be used to visually assess the results of an FL experiment run and users can also extract information about client utilization to detect performance degradation. However, as the name denotes, this tool is limited to the application of FL on the CARLA autonomous car driving simulator. Quan et al. [43] introduced HFLens a tool designed to enable the execution of FL experiments and then provide fine-grained analysis for the experiment administrators, such as anomaly detection and client contribution assessment. In turn, Wang et al. [44] introduced HetVis, a visual analysis tool that, other than providing model-level monitoring metrics, enables users to iterate through each training round to assess data heterogeneity among the global model contributing clients. Finally, Fan et al. [45] proposed FATE-Board, a monitoring service add-on that can be integrated with the open-source FATE FL execution framework. This service enables users to view the experiment job status and graph, as well as summarize (statistically) the execution logs of a recently run FL experiment. A significant downside is that this tool is tailored to FL over a single administrative domain as the data must be collected in a single central database (i.e., MySQL), meaning the privacy-preserving aspect is lost to enable experiment monitoring.

8. Conclusions and Future Directions

In conclusion, FedMon is a toolkit that offers a comprehensive yet user-friendly solution to the challenges of monitoring FL deployments. Automating the extraction of FL performance metrics and infrastructure utilization metrics allows users to solely focus on the creation of FL pipelines without caring about the monitoring metrics extraction. Furthermore, by providing a comprehensive suite of analytic methods, FedMon significantly reduces the manual effort and expertise required to manage and optimize FL systems. The latter not only streamlines the monitoring process, but also enhances the efficiency and effectiveness of FL deployments across various edge computing scenarios. With FedMon, FL practitioners can navigate the complexities of distributed model training, ensuring that the potential of FL is fully realized in facilitating large-scale, privacy-preserving AI solutions. To this end, we highlighted a representative use case

with numerous FL trials, which can be done over edge deployments, and showcased the usability of our system.

Our future plans include diverse directions and extensions of our FedMon framework. A top priority for the (near) future is the integration of tools (e.g., HFLens [43]) providing algorithmic mechanisms for detecting malicious behavior by clients falsely reporting updates and (possibly) attempting to "poison" the global model. Second, the encapsulation of other FL frameworks like TensorFlow FL will give us the ability to compare different frameworks and extract correlations, not only from ML libraries, but also from FL systems highlighting their bottlenecks and weaknesses. Moreover, we plan to introduce extra features on top of the FedMon analytic model, like components for AI/ML training models on captured monitoring data. The latter will allow us to generate prediction models for FL performance, configurations, and placement on the edge continuum ecosystem. Furthermore, a more user-friendly interface through an interactive dashboard, enabled by a script-to-webapp framework such as Streamlit, https://streamlit.io/ (accessed on 1 April 2024), will provide users with real-time observability of the FL and system-related metrics. Last but not least, enforcing adaptivity on monitoring metric extraction, e.g., by changing the monitoring interval based on the variability of metrics, will minimize the overhead of our monitoring tool in network traffic and storage needs.

Author Contributions: Conceptualization, D.T., M.S. and F.N.; methodology, D.T. and M.S.; software, M.S. and F.N.; validation, M.S. and D.T.; writing—review and editing, M.S., D.T. and F.N.; project administration, D.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: This work utilized popular and openly available ML/FL datasets to ensure the reproducibility of the findings. In addition, the raw output files of the experiments can be obtained by interested researchers/practitioners with a brief request to the authors and relevant attribution (citation) of the origins of the data.

Conflicts of Interest: The authors declare no conflicts of interest. Fotis Nikolaidis is employed by SuperDuperDB GmbH. The funder had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Truong, N.; Sun, K.; Wang, S.; Guitton, F.; Guo, Y. Privacy preservation in federated learning: An insightful survey from the GDPR perspective. *Comput. Secur.* **2021**, *110*, 102402. [CrossRef]
2. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; Arcas, B.A.Y. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the 20th AISTATS, PMLR, Fort Lauderdale, FL, USA, 20–22 April 2017; Volume 54, pp. 1273–1282.
3. Li, T.; Sahu, A.K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; Smith, V. Federated Optimization in Heterogeneous Networks. *arXiv* **2020**, arXiv:1812.06127.
4. Okegbile, S.D.; Cai, J.; Zheng, H.; Chen, J.; Yi, C. Differentially Private Federated Multi-Task Learning Framework for Enhancing Human-to-Virtual Connectivity in Human Digital Twin. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 3533–3547. [CrossRef]
5. Gadekallu, T.R.; Pham, Q.V.; Huynh-The, T.; Bhattacharya, S.; Maddikunta, P.K.R.; Liyanage, M. Federated Learning for Big Data: A Survey on Opportunities, Applications, and Future Directions. *arXiv* **2021**, arXiv:2110.04160.
6. Nikolaidis, F.; Symeonides, M.; Trihinas, D. Towards Efficient Resource Allocation for Federated Learning in Virtualized Managed Environments. *Future Internet* **2023**, *15*, 261. [CrossRef]
7. 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* **2022**, arXiv:2007.14390.
8. Foley, P.; Sheller, M.J.; Edwards, B.; Pati, S.; Riviera, W.; Sharma, M.; Narayana Moorthy, P.; Wang, S.H.; Martin, J.; Mirhaji, P.; et al. OpenFL: The open federated learning library. *Phys. Med. Biol.* **2022**, *67*, 214001. [CrossRef]
9. Liu, Y.; Fan, T.; Chen, T.; Xu, Q.; Yang, Q. FATE: An Industrial Grade Platform for Collaborative Learning with Data Protection. *J. Mach. Learn. Res.* **2021**, *22*, 1–6.
10. Symeonides, M.; Nikolaidis, F.; Trihinas, D.; Pallis, G.; Dikaiakos, M.D.; Bilas, A. FedBed: Benchmarking Federated Learning over Virtualized Edge Testbeds. In Proceedings of the 2023 IEEE/ACM 16th International Conference on Utility and Cloud Computing (UCC), Taormina, Italy, 4–7 December 2023.
11. Trihinas, D.; Pallis, G.; Dikaiakos, M.D. Monitoring Elastically Adaptive Multi-Cloud Services. *IEEE Trans. Cloud Comput.* **2018**, *6*, 800–814. [CrossRef]

12. Mallah, R.A.; Badu-Marfo, G.; Farooq, B. On the Initial Behavior Monitoring Issues in Federated Learning. *IEEE Access* **2021**, *9*, 161046–161054. [CrossRef]
13. Sculley, D.; Holt, G.; Golovin, D.; Davydov, E.; Phillips, T.; Ebner, D.; Chaudhary, V.; Young, M. Machine Learning: The High Interest Credit Card of Technical Debt. In Proceedings of the SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop), Montreal, QC, Canada, 13 December 2014.
14. Breck, E.; Zinkevich, M.; Polyzotis, N.; Whang, S.; Roy, S. Data Validation for Machine Learning. In Proceedings of the SysML, Stanford, CA, USA, 31 March–2 April 2019.
15. Netdata. 2024. Available online: https://www.netdata.cloud/ (accessed on 1 April 2024).
16. Zavala, E.; Franch, X.; Marco, J. Adaptive monitoring: A systematic mapping. *Inf. Softw. Technol.* **2019**, *105*, 161–189. [CrossRef]
17. Vartak, M.; Madden, S. Modeldb: Opportunities and challenges in managing machine learning models. *IEEE Data Eng. Bull.* **2018**, *41*, 16–25.
18. Cho, Y.J.; Wang, J.; Joshi, G. Client Selection in Federated Learning: Convergence Analysis and Power-of-Choice Selection Strategies. *arXiv* **2020**, arXiv:2010.01243.
19. Jothimurugesan, E.; Hsieh, K.; Wang, J.; Joshi, G.; Gibbons, P.B. Federated learning under distributed concept drift. In Proceedings of the International Conference on Artificial Intelligence and Statistics, PMLR, Valencia, Spain, 25–27 April 2023; pp. 5834–5853.
20. Mulay, A.; Gaspard, B.; Naidu, R.; Gonzalez-Toral, S.; Vineeth, S.; Semwal, T.; Manish Agrawal, A. FedPerf: A Practitioners' Guide to Performance of Federated Learning Algorithms. *Proc. Mach. Learn. Res.* **2021**, *148*, 302–324.
21. Prometheus. 2024. Available online: https://prometheus.io/ (accessed on 1 April 2024).
22. cAdvisor. 2024. Available online: https://github.com/google/cadvisor (accessed on 1 April 2024).
23. Symeonides, M.; Georgiou, Z.; Trihinas, D.; Pallis, G.; Dikaiakos, M.D. Fogify: A Fog Computing Emulation Framework. In Proceedings of the IEEE/ACM SEC, San Jose, CA, USA, 12–14 November 2020; pp. 42–54.
24. Asad, M.; Shaukat, S.; Javanmardi, E.; Nakazato, J.; Bao, N.; Tsukada, M. Secure and Efficient Blockchain-Based Federated Learning Approach for VANETs. *IEEE Internet Things J.* **2024**, *11*, 9047–9055. [CrossRef]
25. Nazir, S.; Kaleem, M. Federated Learning for Medical Image Analysis with Deep Neural Networks. *Diagnostics* **2023**, *13*, 1532. [CrossRef] [PubMed]
26. Verginadis, Y. A Review of Monitoring Probes for Cloud Computing Continuum. In Proceedings of the International Conference on Advanced Information Networking and Applications, Juiz de Fora, Brazil, 29–31 March 2023; Barolli, L., Ed.; Springer: Cham, Switzerland, 2023; pp. 631–643.
27. Nagios. 2024. Available online: https://www.nagios.com/ (accessed on 1 April 2024).
28. Zabbix. 2024. Available online: https://www.zabbix.com/ (accessed on 1 April 2024).
29. Ganglia. 2024. Available online: http://ganglia.info (accessed on 1 April 2024).
30. Datadog. 2024. Available online: https://www.datadoghq.com/ (accessed on 1 April 2024).
31. AppDynamics. 2024. Available online: https://www.appdynamics.com/ (accessed on 1 April 2024).
32. Kashansky, V.; Kimovski, D.; Prodan, R.; Agrawal, P.; Marozzo, F.; Iuhasz, G.; Marozzo, M.; Garcia-Blas, J. M3AT: Monitoring Agents Assignment Model for Data-Intensive Applications. In Proceedings of the 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Västerås, Sweden, 11–13 March 2020; pp. 72–79. [CrossRef]
33. Trihinas, D.; Pallis, G.; Dikaiakos, M. Low-Cost Adaptive Monitoring Techniques for the Internet of Things. *IEEE Trans. Serv. Comput.* **2018**, *14*, 487–501. [CrossRef]
34. Pusztai, T.; Morichetta, A.; Pujol, V.C.; Dustdar, S.; Nastic, S.; Ding, X.; Vij, D.; Xiong, Y. Slo script: A novel language for implementing complex cloud-native elasticity-driven slos. In Proceedings of the 2021 IEEE International Conference on Web Services (ICWS), Virtual, 5–11 September 2021; pp. 21–31.
35. Schröder, T.; Schulz, M. Monitoring machine learning models: A categorization of challenges and methods. *Data Sci. Manag.* **2022**, *5*, 105–116. [CrossRef]
36. EvidentlyAI. 2024. Available online: https://www.evidentlyai.com/ml-in-production/model-monitoring (accessed on 1 April 2024).
37. Amazon. 2023. Available online: https://aws.amazon.com/sagemaker/ (accessed on 1 April 2024).
38. WhyLabs. 2024. Available online: https://whylabs.ai/ (accessed on 1 April 2024).
39. Chorev, S.; Tannor, P.; Israel, D.B.; Bressler, N.; Gabbay, I.; Hutnik, N.; Liberman, J.; Perlmutter, M.; Romanyshyn, Y.; Rokach, L. Deepchecks: A Library for Testing and Validating Machine Learning Models and Data. *arXiv* **2022**, arXiv:2203.08491.
40. Sun, R.; Li, Y.; Shah, T.; Sham, R.W.H.; Szydlo, T.; Qian, B.; Thakker, D.; Ranjan, R. FedMSA: A Model Selection and Adaptation System for Federated Learning. *Sensors* **2022**, *22*, 7244. [CrossRef]
41. Meng, L.; Wei, Y.; Pan, R.; Zhou, S.; Zhang, J.; Chen, W. VADAF: Visualization for Abnormal Client Detection and Analysis in Federated Learning. *ACM Trans. Interact. Intell. Syst.* **2021**, *11*, 26. [CrossRef]
42. Lee, T.; Mun, H.; Lee, Y. *A Federated Learning Monitoring Tool for Self-Driving Car Simulation (Student Abstract)*; AAAI Press: Washington, DC, USA, 2023. [CrossRef]
43. Li, Q.; Wei, X.; Lin, H.; Liu, Y.; Chen, T.; Ma, X. Inspecting the Running Process of Horizontal Federated Learning via Visual Analytics. *IEEE Trans. Vis. Comput. Graph.* **2022**, *28*, 4085–4100. [CrossRef] [PubMed]

44. Wang, X.; Chen, W.; Xia, J.; Wen, Z.; Zhu, R.; Schreck, T. HetVis: A Visual Analysis Approach for Identifying Data Heterogeneity in Horizontal Federated Learning. *arXiv* **2022**, arXiv:2208.07491.
45. Fan, T. FATE-Board: The FATE Monitoring and Visualization Toolkit. 2018. Available online: https://github.com/FederatedAI/FATE-Board (accessed on 1 April 2024).