AtlasFL: A Federated Learning Workload Generator with Energy and Carbon Emission Support

Moysis Symeonides
msymeo03@ucy.ac.cy
University of Cyprus
Nicosia, Cyprus

Demetris Trihinas
trihinas.d@unic.ac.cy
University of Nicosia
Nicosia, Cyprus

Abstract

Federated Learning (FL) is prevailing as the dominating service paradigm facilitating the training of Machine Learning (ML) models among a set of collaborating entities in a distributed fashion. Although FL settings are ideal for scalable model training and can limit data privacy exposure for the collaborating entities, evaluating the impact of FL on the underlying infrastructure is challenging due to several different configuration knobs. To address this, we introduce AtlasFL, a framework designed to simplify the benchmarking of FL deployments and enable users to generate realistic workloads that can be used to assess what-if scenarios. A key feature of AtlasFL is that on top of FL-level metrics, AtlasFL also exposes execution traces for the underlying infrastructure utilization, such as compute and memory usage as well as energy consumption. Moreover, AtlasFL can integrate with services that expose energy grid data and energy availability from direct access to renewables to provide carbon footprint estimations for requested FL experiment configurations. To illustrate the utility of AtlasFL, a benchmarking process embracing 5 FL implementations was conducted over an edge micro-DC testbed and afterwards, a use-case scenario is designed to introduce the impact of time-shifting when running together a total of 15 FL experiments.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Distributed artificial intelligence**; • **General and reference** → **Experimentation**; • **Computer systems organization** → **Cloud computing**.

Keywords: Federated Learning, Internet of Things, Workload Generation

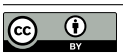
ACM Reference Format:
Moysis Symeonides and Demetris Trihinas. 2025. AtlasFL: A Federated Learning Workload Generator with Energy and Carbon Emission Support. In *3rd International Workshop on Testing Distributed Internet of Things Systems (TDIS '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3719159.3721224

1 Introduction

With an abundance of IoT devices now scattered across the globe, a continuously maturing technology such as Federated Learning (FL) is shaping into a key driver enabling intelligent IoT services. FL as a service paradigm is suitably geared to harvest the vast amounts of data IoT devices produce and take advantage of their available computational power to cooperatively train Machine Learning (ML) models [6]. In a typical FL setting a *Server* is entrusted with the orchestration of the model training process, while the computational effort is spread among a pool of *Clients* [7]. To achieve this, the *Server* initially obtains a set of available clients that meet certain criteria (e.g., availability) and subsequently broadcasts to the *Clients* an initial version of the model. At this point, the *Clients* update the model locally based on local knowledge without exposing (possibly) sensitive information among peers. When the training round is complete, the *Server* collects the local versions from each client and forms through aggregation a new global model. This process is repeated for either a pre-determined set of rounds or the convergence to a certain model loss. With such a rigorous setting, FL is making strives on its promises to preserve (user) data privacy with localized training on individual IoT devices [16] and reducing the bandwidth associated with data transfer over the IoT-Edge-Cloud continuum for centralized model training [14].

While FL deployments offer inherent scalability and limit privacy exposure, assessing their impact on the underlying infrastructure remains a significant challenge with the offering of several different configuration knobs. For example, different ML backends, such as TensorFlow and PyTorch, have been found to exhibit distinct computational profiles during FL [8], and variations in the data distribution of clients further contribute to the disparities in execution times [13]. Additionally, the underlying infrastructure is a critical factor, often comprising heterogeneous computing nodes with varying processing capabilities [4]. When considering the energy efficiency profiles of these diverse nodes, along with efforts to minimize the carbon footprint of FL—such as integrating renewable energy sources or prioritizing workloads in regions with low-carbon energy grids [17]—the evaluation of such systems becomes increasingly complex.

To address these challenges, several initiatives have introduced FL development tools [1, 10] and benchmarking frameworks [2, 13], enabling users to design, execute, and test FL deployments on physical or virtual infrastructures. However, despite the availability of such tools, users still face significant barriers for adoption. Deploying benchmarks for (distributed) ML often requires upfront monetary investments for suitable infrastructure before even assessing suitability for the desired application [3]. Moreover, the instrumentation and measurement of the real energy consumption of FL deployments necessitates integrating smart meters or specialized software tools into the setup [9]. Additionally, evaluating different algorithms, such as node selection or workload scheduling algorithms, demands re-executing FL workloads under identical conditions—an endeavor that is nearly impossible to achieve in practice. Finally, simulation frameworks that aim to simplify the evaluation process focus exclusively on FL evaluation metrics, such as model accuracy and loss, without accounting for the impact of FL on the underlying infrastructure [11].



TDIS '25, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1526-6/2025/03
https://doi.org/10.1145/3719159.3721224

In this paper, we introduce AtlasFL[1], an open and modular framework designed to simplify the benchmarking of FL deployments over physical testbeds and collect execution traces from various FL implementations. A key feature of AtlasFL is that on top of FL-level metrics, such as model accuracy and loss, AtlasFL also exposes traces for the underlying infrastructure utilization, such as compute and memory usage as well as energy consumption. With these traces, AtlasFL can create statistical models and afterwards, enable users to generate realistic workloads to optimize future deployments by running and apprehending the impact of multiple what-if scenarios presenting different FL experiment configurations. Additionally, AtlasFL integrates carbon intensity data and information from renewable energy sources, enabling users to assess the carbon footprint of their designed scenarios. Users can define custom what-if scenarios, specifying parameters such as the FL workload(s), underlying infrastructure, target region, and scheduling algorithm. The system then evaluates all scenarios, providing detailed statistics for each, thereby facilitating A/B testing and performance analysis across different configurations. To illustrate the utility of AtlasFL, a benchmarking process that includes 5 FL implementations was carried out on an edge micro-DC testbed provided by CloudFerro[2] with energy grid and renewable energy data extracted from PV panel racks in Poland and Cyprus. Finally, a use-case scenario is designed to introduce the impact of time-shifting for FL when running together 15 different FL experiment configurations.

The rest of this paper is structured as follows: Section 2 provides a comprehensive overview of the AtlasFl architecture and building blocks as well as implementation details. Section 3 introduces a use-case to showcase the efficacy and efficiencies that AtlasFL can introduce. Section 4 provides a brief overview of the related work, while Section 5 concludes the paper and outlines future work.

2 The AtlasFL Framework

2.1 Overview

Figure 1 depicts a high-level and abstract architectural overview of AtlasFL. This figure illustrates how AtlasFL seamlessly integrates several components to enable users to benchmark, analyze, and simulate FL workloads.

Following a bottom-up approach, the flow of an AtlasFL pipeline begins with the FL benchmarking process conducted offline on a (typically small-scale) testbed. During this phase, users select a set of pre-defined FL implementations with a diverse set of properties. Upon execution, a rich set of monitoring data is extracted from both the underlying testbed and the FL training process. In particular, AtlasFL harvests QoS metrics, such as model accuracy, loss, training time, and convergence ratios, and system performance metrics such as CPU usage, memory footprint, network I/O, energy consumption, etc. The current AtlasFL prototype leverages FedBed [13] to design the provided FL implementations and automate the deployment process over different testbeds. In brief, FedBed allows users to define and customize various FL parameters, including underlying ML backends, model architectures, and data distributions. Subsequently, it performs an automated deployment of a FL experiment over Docker-enabled nodes, including heterogeneous

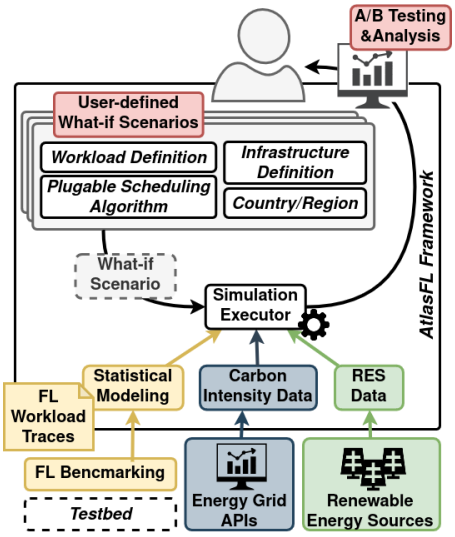


Figure 1. High-level Overview of the Architecture

and geo-distributed IoT environments. During the execution of the FL deployment, AtlasFL extracts the aforementioned monitoring data using from an integrated monitoring system. It is important to note that users are not restricted to FedBed; other FL benchmarks or tools can be employed. However, users must ensure that FL traces are generated in a schema compatible with the AtlasFL framework, enabling seamless integration and analysis.

With the FL monitoring traces in hand, AtlasFL processes them to extract statistical insights, creating a dedicated model for each FL deployment. These models capture key properties for both client and server nodes, with the most critical metrics including: (i) *CPU Requirements*, represented by the 95th percentile of CPU usage on the host node; (ii) *Average CPU Utilization*, which is the mean CPU usage over the duration of different phases of different phases of the FL process (e.g., local training, communication with server; (iii) *Memory Requirements*, defined by the peak memory usage observed during execution; (iv) *Average Memory Usage*, that is the mean memory consumption over the workload's runtime; and (v) the *Average Power Drawn*, which represents the typical power consumption for the different FL training process operations. These statistical properties provide a comprehensive profile for each FL implementation, enabling detailed simulation and analysis in subsequent stages. In addition, AtlasFL incorporates external datasets, including regional carbon intensity data and renewable energy availability, providing users with insights on the energy efficiency and environmental impact of FL scenarios.

Despite the simplicity in defining benchmarking requirements, this process can be time-consuming and requires access to a testbed resembling the end-environment of the application. As previously mentioned, this can be a significant adoption barrier for FL. As such, for users wanting a faster approach, the benchmarking phase can be omitted by opting for precompiled traces and simulation models made readily available in the AtlasFL repository.

In either case, when AtlasFL has compiled the required statistical models, users can interact with the framework to define custom what-if scenarios. Here, users specify critical parameters such as FL workload characteristics (e.g., model type and dataset), the underlying infrastructure (e.g., hardware configurations and energy profiles), the geographic region (affecting carbon intensity), and the

[1] https://github.com/UCY-LINC-LAB/GreenAnalyzer-Project/tree/main/Analysis%20%26%20Datasets/aerOS-deployment
[2] https://cloudferro.com/

Figure 2. CloudFerro Containerized Edge Micro-DC testbed

scheduling algorithm to be used. A set of these scenarios are fed into the framework and then AtlasFL transitions into the simulation phase, managed by the Simulation Executor. This component uses the statistical models derived from the previous phase to mimic the execution behavior of the FL implementation under the conditions specified by the user. The simulator accounts for hardware heterogeneity, energy efficiency profiles, and regional energy characteristics to create statistical runs of a scenario. As scenarios are processed, the framework infers detailed metrics for each scenario, including execution time, energy consumption, and carbon intensity. These metrics allow users to perform in-depth analyses and A/B testing, enabling comparisons between different configurations and identifying optimal setups for their FL deployments.

To enhance user interaction and decision making, AtlasFL illustrates evaluation results through plotted visualizations, to compare different experiment scenarios and assess performance trade-offs. Users can explore various configurations and make informed decisions based on the presented data. In addition, based on the results, users can refine their configurations, adjust parameters, or explore alternative strategies. By bridging the gap between FL evaluation metrics and infrastructure-level considerations, AtlasFL enables a comprehensive approach to designing and optimizing FL deployments, particularly in energy- and carbon-sensitive environments.

2.2 Benchmarking & FL Traces

As previously mentioned, the benchmarking phase can be omitted, with AtlasFL users opting to use precompiled traces and simulation models made readily available in the AtlasFL repository. This section describes how the available traces were created.

AtlasFL traces are formatted as comma-separated value (CSV) files that contain metrics such as CPU utilization, memory footprint, power consumption, etc. To evaluate our framework, we generated these traces by conducting benchmarking experiments on a real-world testbed. The testbed of choice was the CloudFerro WAW2-1 site made available to the authors through the aerOS H2020 project. The CloudFerro site offers (literally) a containerized edge micro-DC with Kubernetes administrated compute nodes (Fig 2). From this testbed, a cluster of 10 compute nodes was allocated, each equipped with 48 CPU cores and 256 GB of memory. A monitoring agent installed on each node collected the necessary utilization metrics from the operating system, while the PowerTOP[3] software tool was used to measure power consumption.

The FL deployment was set up so that 1 compute node was assigned the role of the Server and the other 9 nodes were configured as Clients. This was achieved by leveraging FedBed [13]. The FedBed codebase allows for flexibility through a set of environmental variables, enabling adjustments to the FL service deployment without requiring the reconstruction of containerized images. A running FedBed container executes a specific FL service—server or client—and activates corresponding components such as aggregation algorithms, ML, and datasets.

Five different implementations of FL experiments were realised so that traces can be collected. Specifically, the 5 FL experiments are configured as follows: (i) a Convolutional Neural Network (CNN) implemented using PyTorch as the ML backend and subject to being trained for 200 rounds while embracing the MNIST dataset; (ii) the same PyTorch CNN model, but trained for 50 rounds; (iii) a CNN model implemented using TensorFlow as the ML backend and subject to being trained for 50 rounds using the MNIST dataset; (iv) the MobileNetV2 model implemented via PyTorch and trained on the CIFAR-10 dataset for 50 rounds; and (v) the same MobileNetV2 model but implemented using TensorFlow and again trained for 50 rounds using the CIFAR-10 dataset. For reference, MNIST and CIFAR-10 are open-source image classification datasets that are widely used to evaluate FL algorithms and testbeds. In all cases, Client nodes are assigned 25,000 randomly selected data points for training and 5,000 data points for evaluation, ensuring consistency across the experiments.

Figures 3, 4, and 5 depict an aggregated overview of the traces that show trends on 3 of the monitored performance axes. The left plot highlights the overall training duration for each FL experiment. The workload involving 200 training rounds required approximately 8,000 seconds (just over 2 hours). This was followed by the MobileNetV2 model trained for 50 rounds in both TensorFlow and PyTorch, each taking around 6,000 seconds (just over 1.5 hours). The 50-round PyTorch-MNIST workload completed in approximately 2,500 seconds (slightly more than 40 minutes), while the TensorFlow-MNIST model demonstrated the fastest performance, finishing in under 1,000 seconds (less than 30 minutes). These results underscore the varying computational demands of different models and ML backends.

The middle plot highlights the energy consumed by each workload, measured in Joules, specifically for the compute nodes involved in training. Interestingly, the PyTorch-CIFAR10 workload consumed the highest amount of energy, although it required less time to complete compared to PyTorch-MNIST. This is attributed to the complexity of the MobileNetV2 model, which demands significantly more computational resources than the simpler neural network used for the MNIST dataset. This increased complexity results in higher CPU utilization (as shown in the right plot), leading to higher power consumption per second.

The second most energy-intensive workload was the PyTorch-MNIST model with 200 rounds, primarily due to its extended duration. This was followed by the TensorFlow-CIFAR10 workload, which also involved training the complex MobileNetV2 model for an extended period. Notably, the PyTorch-MNIST workload consumed more energy than the TensorFlow-MNIST workload, even though the latter exhibited higher CPU utilization. This observation underscores the significant role that execution duration plays in determining a workload's total energy consumption.

Finally, Figure 5 depicts the mean cpu utilization per training round for each of the 5 FL configurations. In turn, Figure 6 depicts the loss per training round. For brevity, we show only plots for the CNN model trained for 50 rounds embracing TensorFlow (top) and PyTorch (bottom) as the ML backend. Interestingly, one can observe

[3] https://www.intel.com/content/www/us/en/developer/articles/tool/powertop-primer.html

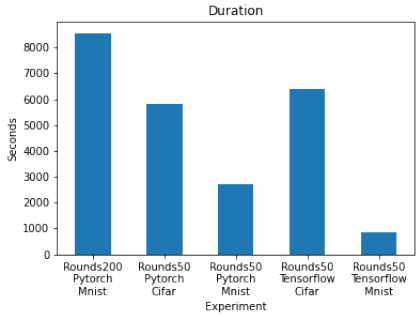


Figure 3. Training Duration

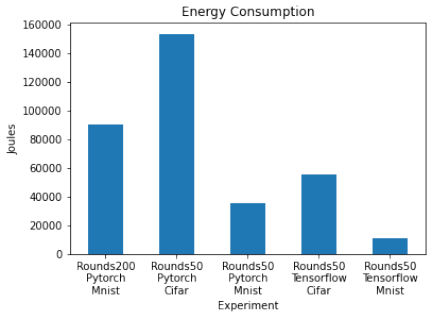


Figure 4. Energy Consumption

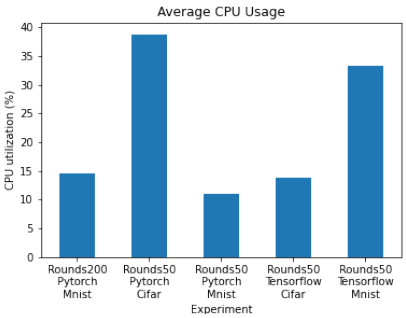


Figure 5. CPU Utilization

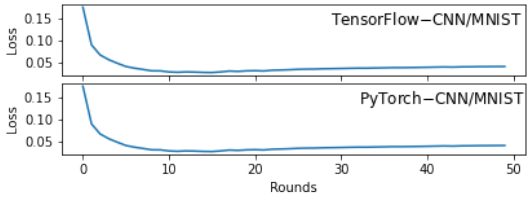


Figure 6. Example of TF models loss for 50 Rounds of FL execution

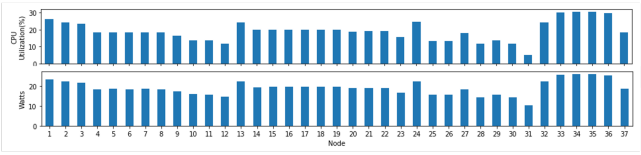


Figure 7. FL Client Mean CPU and Power for Local Training

that the loss minimizes in less rounds when adopting PyTorch as the ML backend.

2.3 Energy Grid Carbon Sources & RES Data

For AtlasFL to provide carbon footprint estimations, a similar workflow with the FL trace collection must be adopted. To support this, AtlasFL provides integration endpoints to access energy grid data from the country or region where the deployment is provisioned and acknowledge if on-site Renewable Energy Sources (RES) are directly available during computations.

The current version of AtlasFL, is compatible with ElectricityMaps[4]. ElectricityMaps is a service that enables the extraction of real-time data (and forecasts) of energy grid production for various countries and local regions. In turn, for RES data, AtlasFL provides another integration endpoint where datapoints for PV rack energy production can be made available to AtlasFL at an hourly-based granularity and following a CSV based data format.

Similar to the benchmarking process, users that do not want to use the real-time offerings can take advantage of the pre-compiled data made readily available in the AtlasFL repository. Specifically, energy source data are available for both Cyprus and Poland that have been collected across several different time periods and weather conditions. Moreover, RES data is available and embraces ML models developed for self-hosted photovoltaic (PV) racks located at the University of Cyprus, as described in [15]. Specifically, these models were created using a dataset that collected hourly metrics over 135 days (3,240 data points) from 3 PV panel racks, namely, PV1 (107.1 m2), PV2 (95.2 m2), and PV3 (57.8 m2) with the best-performing models achieving error rates of 7.3%, 9.0%, and 8.9% for PV1, PV2, and PV3, respectively. With these models, we can input weather data from various regions to simulate and generate corresponding results, as if the PV panels were deployed in those specific locations.

[4] https://app.electricitymaps.com/map/

2.4 Workload Modeling & Simulation

To model FL workloads, we need to consider what are the various types of tasks an FL system performs. Through our (Python) programming abstractions, one can define when an FL workload will be deployed, represented by the `start_at` property, as well as the properties of the workload, such as the `dataset`, underlying `ml_library`, `model`, and other relevant attributes. The workload also specifies the number of respective `tasks`, including the FL server task and a set of FL client tasks, along with a starting `tolerance` indicating how tolerant the user is regarding the execution time of the workload. Then, each task is characterized by a unique `id` and a set of properties, such as `execution_duration`, compute requirements (`CPU` and `memory`), and `power_consumption`. It is important to note that these properties depend on both the requested FL implementation as well as the underlying infrastructure. Consequently, the next entity in our workload modeling is the `compute node`, which is identified by its unique `id`. A compute node is characterized by a node `type`, which indicates its `total` resources and characteristics, its `location`, and includes runtime properties such as `available resources` (e.g., `CPU` and `memory`) and a set of `running tasks`, which is initially empty upon initialization. Lastly, users may specify renewable energy sources (`RES`) in their model, which only requires the relative `path` and `column` of the respective CSV dataset.

When the description of the FL workloads and the infrastructure is ready, then the *AtlasFL Simulation Executor* is ready to generate the respective workload traces. The current AtlasFL implementation embraces the key metrics collected during the benchmarking process along with the compute note features and are fed into a regression model to infer the runtime load per node. For this, specifically, the simulator invokes the `schedule_service` function, which takes as input the FL `implementation`, desired `configurations` for the requested what-if scenarios, a list of `nodes`, and the simulation `duration` (in seconds).

Before the simulator executes its pipeline, it invokes the *update_start_time* method for each service. By default this method does not influence the starting point of a workload. However, users can easily update this method in order to postpone the execution of a service based on the available renewable energy or carbon

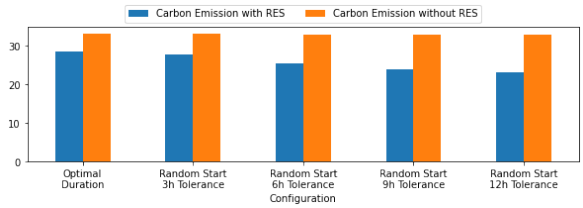


Figure 8. Total CO_2 Emissions (kg) per Scheduling Strategy

intensity. Then, for each second, it observes if a task is finished and release the respective node's resources. When the finished tasks release the occupied resources, the simulator evaluates the starting time of the undeployed workloads, and, if the starting time is more than the current second, the simulator calls the `schedule_service`, in order to place the tasks of the workload on available nodes. At this point, the system considers different task requirements and execution duration based on the candidate nodes. Metrics are extracted automatically from the FL Workload Traces, and if the algorithm selects the respective node, the properties of the deployed task will be updated respectively. Currently, our implementation follows a greedy approach, deploying a task on a node that is available, but users can easily extend this approach to materialize their placement strategies. Lastly, the simulator keeps statistics of the execution creating utilization and energy consumption dataset for the respective scenario, and combines it with renewable sources and energy grid carbon intensity data.

Finally, it is worth highlighting that users can define a set of *what-if scenarios* encompassing multiple FL workloads, pluggable algorithms, varying infrastructures, and geographically distributed regions. The system automatically executes these scenarios sequentially, generating the corresponding simulated datasets. This functionality enables users to perform A/B testing and conduct an in-depth analysis of their workloads with ease.

3 Use-Case: Time Shifting Federating Learning

This section introduces a use case scenario that aims to demonstrate the utility of AtlasFL.

Suppose that an organisation wants to employ FL to harness the power of distributed ML but wants to do so by examining if the introduction of time shifting can reduce its carbon footprint by enabling the system to suggest delaying model training to take advantage of accessible on-site RES. In an attempt to understand the gains of time shifting, the organisation employs AtlasFL and its ready-to-use modeling to generate data for a 48h horizon and several different FL configurations to evaluate the outcome of different scheduling strategies.

Let us assume the organization is interested in simulating a setting with 1 server and 37 clients and generate FL experiments with the following configurations: (i) a different number of training rounds (50, 100, 200, 400); (ii) the use of 2 different ML backends (TensorFlow, PyTorch); and (iii) the adoption of 2 different datasets (MNIST, CIFAR-10). With these configurations, a total of 15 different FL experiment scenarios can be established and run together. As a country of preference, impacting estimation of carbon emissions, the organisation opts to use data from Lodz, Poland. The scheduling strategies will examine executing the model training process immediately aiming at minimizing execution time without considering any carbon efficiency gains, versus the time shifting

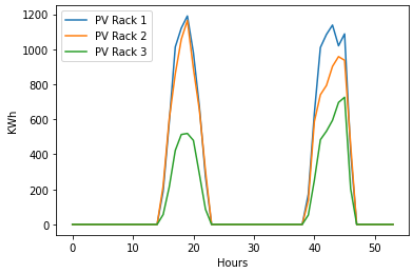


Figure 9. PV Panel Production Estimation for 48h in Lodz

adoption where the training process can be delayed for either 3, 6, 9 or 12 hours. For time shifting, a greedy algorithm is employed to decide when to start the FL task. Moreover, in reference to client data, we reiterate that each client receives a random sample of 25K datapoints for training and 5K for test-based validation, all originating from the utilized dataset (MNIST, CIFAR).

The following summarizes key findings from the evaluation of the organisation after embracing AtlasFL. Figure 7 depicts the mean compute usage and power drawn by each FL client node during local training after the simulated workload is produced. In turn, Figure 9 depicts the estimated production for an on-site PV panel rack connected to the edge-microDC based on the rack capacity, configuration, and weather conditions. Moreover, Figure 8 illustrates CO_2 emissions over the 48h period for the various scenarios. The blue bars represent deployments incorporating RES, while the orange bars depict carbon intensity without RES integration in Lodz. The results clearly demonstrate that utilizing RES significantly reduces the carbon footprint of the workloads. Specifically, this reduction averages around 21% and, in some cases, reaches up to 29.7%, depending on the strategy employed. The impact of FL time shifting is notable. For instance, introducing a 6-hour postponement tolerance reduced carbon emissions by up to 10.2% in RES-enabled deployments but only 1% in deployments without RES. Similar results are observed for the rest of the time-shifting strategies, the best being the 12 hour delay with a reduction of about 18% from the immediate execution strategy. When both RES integration and FL time shifting are combined, our methodology demonstrated a significant carbon reduction of approximately 29.9% compared to the baseline.

Finally, to visually illustrate our findings, we present the results of the 12h FL time shifting greedy algorithm with and without RES integration. As shown in Figure 10, the carbon intensity remains identical between the two deployments until 6 a.m., when PV energy generation begins. From this point until 5 p.m., the RES-enabled deployment (depicted by the blue line) demonstrates a significant reduction in carbon intensity, particularly around midday, when carbon emissions drop to zero. A similar pattern is observed on the second day, with an extended period of zero carbon emissions, attributed to the absence of scheduled workload during this time.

4 Related Work

Flower [1] and OpenFL [10] are tools enabling the design of FL implementations by abstracting the challenges introduced by dealing with a distributed environment (i.e., coordination) while also providing ready-to-use FL algorithms and client selection strategies. In turn, FedBed [13] and FedLab [18] provide both configurable and automated deployment of FL implementations across physical

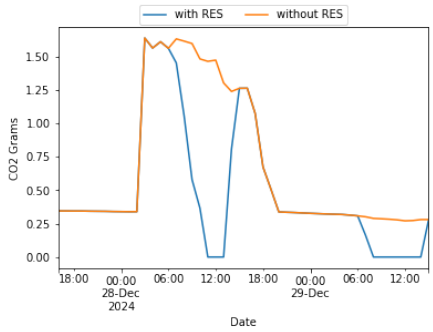


Figure 10. 12h Time-Shifting w/o RES Integration

and virtual testbeds, while also extracting during execution several monitoring metrics.

In regards to benchmarking, LEAF [2] is a popular suite that features a set of 5 reference FL implementations that focus on text and image processing. LEAF provides a set of metrics to evaluate model accuracy and running time, and also outputs at the end of each experiment run the amount of computing resources needed from each client in FLOPS and number of bytes downloaded/uploaded. In turn, FedScale [5] presents a similar bechmarking framework but interestingly, also incorporates a runtime that enables the testing of different FL experiments in a controlled environment.

Notable FL workload simulation tools are the following. FLSim by Facebook Research [11] is a framework written in PyTorch that simulates FL experiments in controlled environments (i.e., laptop, single-board device) by allowing users to configure the number of clients, as well as client-server communication and data exchange. FLSim features 3 workloads from the domains of computer vision and text processing. In terms of metrics, only model accuracy/loss is reported at a per training basis. In turn, FedJAX [12] is an open-source python library released by Google Research with the intent to provide simple primitives for implementing FL algorithms and testing them in a controlled environment without dealing with the complexities introduced by distributed system deployment. After a test is complete, FedJAX provides evaluation metrics but these are limited to only model quality and training duration.

5 Conclusions and Future Work

This work tackles the complex endeavor of benchmarking and workload generation for experiments featuring Federated Learning due to the uncertainties introduced by several different configuration knobs in FL settings. Key innovations of our work include (i) the ability to run offline benchmarking with several different FL implementations to collect execution traces,se covering different ML backends, models, experiment duration, and datasets; (ii) the collection of traces beyond model QoS, expanding coverage to include computational resources, memory footprint, and energy consumption. (iii) integrating energy grid data and data from renewable energy sources to provide carbon footprint estimations; and (iv) for users in need of avoiding the time-consuming benchmarking process, the direct availability of the simulation models for workload generation.

Our future work directions are three-fold. First, we aim at further enriching the configuration selectivity of the workload generator. Second, we aim at extending the carbon estimation module to collect real-time data and cover countries and regions beyond the two that are currently made available. Third, we aim to conduct a scalability evaluation for AtlasFL.

Acknowledgment. This work is part of GreenAnalyzer and AdaptoFlow, which have indirectly received funding from the European Union's Horizon Europe research and innovation action programme, via the aerOS Open Call issued and executed under the aerOS project (Grant Agreement no. 101069732) and the TRIALSNET Open Call issued and executed under the TrialsNet project (Grant Agreement no. 101017141), respectively.

References

[1] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. 2022. Flower: A Friendly Federated Learning Research Framework. arXiv:2007.14390 [cs.LG]

[2] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097* (2018).

[3] Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, and David Owen. 2024. The rising costs of training frontier AI models. arXiv:2405.21015 [cs.CY] https://arxiv.org/abs/2405.21015

[4] Enmao Diao, Jie Ding, and Vahid Tarokh. 2020. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264* (2020).

[5] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. 2022. FedScale: Benchmarking Model and System Performance of Federated Learning at Scale. In *Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.). PMLR, 11814–11827.

[6] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th AISTATS*, Vol. 54. PMLR, 1273–1282.

[7] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. 2021. Federated Learning for Internet of Things: A Comprehensive Survey. *IEEE Communications Surveys and Tutorials* 23, 3 (2021), 1622–1658. doi:10.1109/comst.2021.3075439

[8] Fotis Nikolaidis, Moysis Symeonides, and Demetris Trihinas. 2023. Towards Efficient Resource Allocation for Federated Learning in Virtualized Managed Environments. *Future Internet* 15, 8 (2023), 25 pages.

[9] Xinchi Qiu, Titouan Parcollet, Javier Fernandez-Marques, Pedro P. B. Gusmao, Yan Gao, Daniel J. Beutel, Taner Topal, Akhil Mathur, and Nicholas D. Lane. 2024. A first look into the carbon footprint of federated learning. *J. Mach. Learn. Res.* 24, 1, Article 129 (March 2024), 23 pages.

[10] G Anthony Reina, Alexey Gruzdev, Patrick Foley, Olga Perepelkina, Mansi Sharma, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, et al. 2021. OpenFL: An open-source framework for Federated Learning. *arXiv preprint arXiv:2105.06413* (2021).

[11] Facebook Research. [n. d.]. Federated Learning Simulator (FLSim). https://github.com/facebookresearch/FLSim

[12] Jae Hun Ro, Ananda Theertha Suresh, and Ke Wu. 2021. FedJAX: Federated learning simulation with JAX. *arXiv preprint arXiv:2108.02117* (2021).

[13] Moysis Symeonides, Fotis Nikolaidis, Demetris Trihinas, George Pallis, Marios D Dikaiakos, and Angelos Bilas. 2023. FedBed: Benchmarking Federated Learning over Virtualized Edge Testbeds. In *2023 IEEE/ACM 16th International Conference on Utility and Cloud Computing (UCC)*. IEEE.

[14] Moysis Symeonides, Demetris Trihinas, and Fotis Nikolaidis. 2024. FedMon: A Federated Learning Monitoring Toolkit. *IoT* 5, 2 (2024), 227–249. doi:10.3390/iot5020012

[15] Moysis Symeonides, Nicoletta Tsiopani, Georgios Maouris, Demetris Trihinas, George Pallis, and Marios D. Dikaiakos. 2024. CarbonOracle: Automating Energy Mix & Renewable Energy Source Forecast Modeling for Carbon-Aware Micro Data Centers. In *Proceedings of the 17th IEEE International Conference on Utility and Cloud Computing (UCC)*.

[16] Nguyen Truong, Kai Sun, Siyao Wang, Florian Guitton, and YiKe Guo. 2021. Privacy preservation in federated learning: An insightful survey from the GDPR perspective. *Computers & Security* 110 (2021), 102402. doi:10.1016/j.cose.2021.102402

[17] Philipp Wiesner, Ramin Khalili, Dennis Grinwald, Pratik Agrawal, Lauritz Thamsen, and Odej Kao. 2024. FedZero: Leveraging Renewable Excess Energy in Federated Learning. In *Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems (e-Energy '24)*. Association for Computing Machinery, New York, NY, USA, 373–385. doi:10.1145/3632775.3639589

[18] Dun Zeng, Siqi Liang, Xiangjing Hu, Hui Wang, and Zenglin Xu. 2023. FedLab: A Flexible Federated Learning Framework. *Journal of Machine Learning Research* 24, 100 (2023), 1–7.