Where is the Testbed for my Federated Learning Research?

Janez Božič[a][b]*, Amândio R. Faustino[b]*, Boris Radovič[a][b]*, Marco Canini[b], Veljko Pejović[a]
[a] University of Ljubljana [b] KAUST

2024 IEEE/ACM Symposium on Edge Computing (SEC) | 979-8-3503-7828-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/SEC62691.2024.00027

***Abstract*—Progressing beyond centralized AI is of paramount importance, yet, distributed AI solutions, in particular various federated learning (FL) algorithms, are often not comprehensively assessed, which prevents the research community from identifying the most promising approaches and practitioners from being convinced that a certain solution is deployment-ready. The largest hurdle towards FL algorithm evaluation is the difficulty of conducting real-world experiments over a variety of FL client devices and different platforms, with different datasets and data distribution, all while assessing various dimensions of algorithm performance, such as inference accuracy, energy consumption, and time to convergence, to name a few. In this paper, we present COLEXT, a real-world testbed for FL research. COLEXT is designed to streamline experimentation with custom FL algorithms in a rich testbed configuration space, with a large number of heterogeneous edge devices, ranging from single-board computers to smartphones, and provides real-time collection and visualization of a variety of metrics through automatic instrumentation. According to our evaluation, porting FL algorithms to COLEXT requires minimal involvement from the developer, and the instrumentation introduces minimal resource usage overhead. Furthermore, through an initial investigation involving popular FL algorithms running on COLEXT, we reveal previously unknown trade-offs, inefficiencies, and programming bugs.**

***Index Terms*—Federated Learning, Testbed, Performance Evaluation**

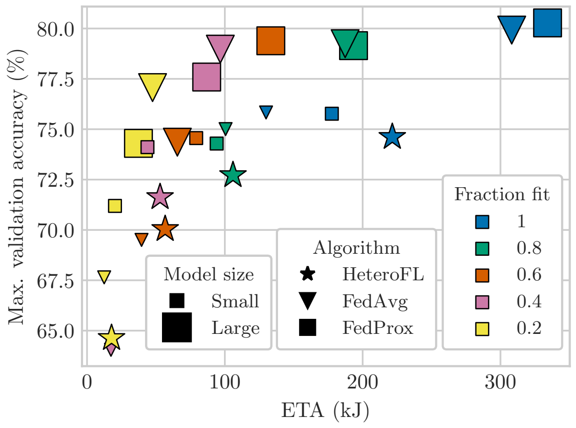


Fig. 1: Max. validation accuracy and energy to accuracy (ETA) for three FL algorithms on the CIFAR-10 dataset. In FedAvg and FedProx, all clients use either a *Small* or a *Large* model, while in HeteroFL, clients use one of the two depending on the computational power. The ETA axis values cannot be (reliably) assessed without real-world experimentation provided by COLEXT.

I. INTRODUCTION

Data is a most precious resource, and the one that, with the growth of privacy awareness, users tend to be less inclined to share with third parties. Centralized AI has made tremendous advances in the last couple of decades. Yet, virtually all of the publicly available data, such as the content of the World Wide Web, are already being used for large foundational models. The next breakthrough in AI will necessarily have to rely on high-quality private data residing on end-user devices.

Harnessing individuals' data while maintaining privacy is a challenging feat, and various research approaches have been proposed to tackle this issue [1], [2]. Federated learning (FL) allows distributed collaborative training of machine learning (ML) models over a group of participating devices, with a centralized server used merely for training orchestration, essentially aggregating clients' model updates and sharing the newly-created model within the group [2]. Its conceptual simplicity makes FL the most popular solution for privacy-preserving distributed AI, especially when deep learning (DL) models are involved.

*Equal contribution. Work done in part while Janez Božič was interning at KAUST.

The original FL algorithm, FedAvg [3] was shown to underperform when different assumptions, such as the uniformity of data distribution or computational capabilities over clients, are lifted [4]–[11]. A large body of theoretical work has followed, and various attempts to alleviate the issue have been made [12]–[17]. Nevertheless, these scientific contributions very rarely trickle down to practice, primarily because they do not provide readily usable implementations and fail to convince that the claimed improvements indeed translate to real-world deployments. Instead, most proposals remain at the level of a simulation running on a single server, and the real-world behavior in terms of the algorithm running time, computational/memory/energy demand, and performance under various real-world constraints, such as with data/network/device heterogeneity, remains unknown.

In this paper, we aim to increase the credibility of FL research by providing COLEXT – a solution for reproducible FL experimentation over real-world devices while also enabling a gamut of relevant performance metrics to be collected. Our solution is tailored to support virtually any existing and future algorithm constructed on top of the currently most popular FL framework, Flower [18], and enables a range of scenario-defining parameters, such as client heterogeneity, metric col-

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

lection configurations, and others, to be set. We implement CoLExT in a federation of 28 single-board computers (SBCs) and 20 Android smartphones and demonstrate its utility for objectively assessing FL algorithms over different dimensions.

As an illustrative example, in Figure 1, we depict the result of CoLExT experimentation over three popular FL algorithms (FedAvg [3], FedProx [12], and HeteroFL [19]), with two different model sizes (small – with 35k and large – with 380k parameters), and a different ratio of clients participating in every training round. Traditional means of simulation-based assessment would "see" only the y-axis in the figure, essentially comparing algorithms based on the highest accuracy achieved, and would identify FedProx over a 380k parameter model with full client participation as the most promising solution. CoLExT, on the other hand, uncovers other metrics that may be relevant, such as the memory, CPU, and energy usage, as well as the training duration, and juxtaposes them with the achieved accuracy. In Figure 1, for instance, CoLExT reveals that the amount of energy needed for reaching a particular level of accuracy (i.e., energy-to-accuracy, ETA) differs drastically among points that achieve very similar accuracy. Thus, the previously identified FedProx configuration reaches the top accuracy while consuming almost 350 kJ (kilo Joules). At the same time, involving only 40% of the clients in each training round, the FedAvg algorithm incurs a 4 p.p. (percentage points) decrease in accuracy while consuming less than a third of the energy (i.e., 100 kJ).

The above is just one of the examples of how CoLExT can uncover nuances related to the performance of FL algorithms in real-world environments and can do this with minimal involvement of the algorithm developer. More broadly, our work brings the following contributions to FL research:

• **We design and implement a framework for experimentation with FL algorithms that readily supports a wide range of existing and future FL solutions.** Experimenters need to make minor changes (three lines of code) to run their Flower-ready algorithms on CoLExT.

• **We instrument algorithms to collect a range of metrics.** We implement low-level acquisition of CPU/GPU utilization, memory consumption, training time, and network usage. We also expose real-world energy measurements through power meters deployed in our setup.

• **We develop an expressive interface for defining the experimentation scenario.** An experimenter can decide on the level of heterogeneity among devices, change the experiment's training parameters, and configure metric collection settings.

• **We deploy CoLExT in a heterogeneous device setting and perform thorough experimentation with a number of popular FL algorithms.** Our CoLExT testbed includes both SBCs and Android devices. Through microbenchmarks, we confirm that CoLExT can support a range of algorithms with negligible impact on the device's resources, while use-case experimentation demonstrates how CoLExT can be harnessed to uncover implementation inefficiencies and trade-off issues related to the real-world use of FL.

While this paper unveils only a few important and interesting revelations (such as the one described in Figure 1), we believe that CoLExT will help practitioners navigate the trade-offs that different FL algorithms avail and will also help researchers identify the most promising directions for future development of FL algorithms. To facilitate this, CoLExT is available to interested researchers and the code is open source. Information on how to access the testbed and the code can be found at https://github.com/sands-lab/colext.

II. Background and Obstacles to Realistic FL Experimentation

A. FL Primer and Algorithm Variations

FL is arguably the most promising means of training AI models in a distributed manner so that the data of individual training participants (clients) remain private. In its simplest form, FL operates in rounds $t = 0, 1, \ldots, T-1$, and lets each client k from a set of clients $\mathcal{K} = \{1, 2, \ldots, K\}$ independently train a model $\mathbf{w}$ over its local dataset $\mathcal{D}_k$ using stochastic gradient descent (SGD) or a variation thereof. After E local training epochs, the updated versions of the model $\mathbf{w}_k$ are sent to the server, which then aggregates them in a new version of the global model, usually weighting the contribution of each client according to the number of data samples n_k in $\mathcal{D}_k$: $\mathbf{w}^{(t+1)} = \frac{1}{\sum_{k \in \mathcal{K}^{(t)}} n_k} \sum_{k \in \mathcal{K}^{(t)}} n_k \mathbf{w}_k^{(t,E)}$. After T rounds, the global model $\mathbf{w}^{(T)}$ is considered trained.

The above algorithm, which is essentially a form of distributed SGD with local steps, is termed FedAvg and represents the de-facto baseline from which numerous other FL algorithms have been developed and have been pitted against. For instance, FedProx [12] extends the FedAvg local training loss function $F_k(\mathbf{w})$ with a proximal term $\frac{\mu}{2}\|\mathbf{w} - \mathbf{w}^{(t)}\|^2$, where μ is a non-negative constant, that penalizes large deviations of the local model from the global model. i.e. the loss function becomes: $F_k^{\text{prox}}(\mathbf{w}) = F_k(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w} - \mathbf{w}^{(t)}\|^2$. Other solutions may introduce other modifications and innovations, for instance, by changing the way in which local weight updates are aggregated on the server [14], enabling knowledge distillation among the global and the local models [20], or, as is the case with HeteroFL (results of which are depicted in Figure 1) by allowing aggregation of models of different sizes. Notably, even without any modifications, FedAvg already enables significant customization, as several hyperparameters, such as the number of local epochs, the number of clients per round, the deadline for receiving an update from a client, and the number of clients that have to report to a server for a round to be considered successful, all can be tuned and have been shown to influence the performance of the algorithm [8], [10].

B. Experimentation Challenges

1) Impact of Heterogeneity on FL: In a centralized setting, SGD is performed on the same computing device, and the data comes from the same distribution in each iteration of the algorithm. When SGD is, through FL, deployed over multiple clients, the convergence of the resulting distributed algorithm may be affected by the heterogeneity of a real-world setting.

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

For practical purposes, critical heterogeneities affecting FL include data, hardware, and platform heterogeneities.

Data heterogeneity. The nature of data distribution among clients plays a pivotal role in FL. In Independent and Identically Distributed (IID) scenarios, each client's data conforms to a uniform distribution, simplifying model aggregation across devices. Conversely, non-IID data presents a more challenging landscape where client dataset distributions vary significantly. This diversity demands nuanced strategies to adaptively reconcile differences between local and global models while preserving data privacy and achieving robust model performance.

Data heterogeneity is the most broadly examined property of realistic FL, and a plethora of algorithms, including previously described FedProx, have been proposed to address the issue [11]–[17], [20]–[23]. At least a part of the reason for intensive research in this direction can be explained by the ease at which one can experiment with FL over non-IID data – the actual clients may remain simulated, while datasets assigned to these clients can be made artificially non-IID.

Hardware heterogeneity. "Stragglers," clients who, usually due to poor computing capabilities, take prohibitively long to complete a round of local training, present a major issue in practical FL [24]. Under the presence of stragglers, FL becomes highly inefficient because either other clients must wait for the stragglers, or the stragglers' local updates must be discarded for the learning to advance [25]. Alternatives, such as asynchronous FL, have been proposed [26], but a common approach in practice is to select clients with uniform hardware specifications [27]. Unfortunately, such an approach severely limits the applicability of FL and may introduce bias in the resulting models, as clients of certain characteristics (and, consequently, data properties) do not feature in the training.

Solutions for learning over clients of heterogeneous capabilities have been proposed [19], yet independently evaluating such solutions remains challenging, as it would require a testbed of sufficiently heterogeneous clients. Furthermore, with heterogeneous clients come heterogeneous processing speeds, power consumption, memory usage, and other metrics, which suddenly expand the dimensionality in which the optimal FL solution should be sought. As seen in Figure 1, introducing just one dimension (energy to accuracy) may alter the way we assess the optimality of an FL algorithm.

Platform heterogeneity. FL is promoted as a solution for edge AI. Yet, "edge" encompasses a wide range of devices, from embedded devices and single board computers (SBCs) common in the Internet of Things (IoT) deployments, to smartwatches, smartphones, and beyond. Nevertheless, FL algorithms are rarely tested in actual deployments, and even more rarely are evaluated on multiple platforms. Mobile devices, in particular, tend to be highly underrepresented when it comes to evaluating FL solutions. FL over different platforms is challenging to implement, and, to the best of our knowledge, only one framework – Flower [18] – enables distributed training over both Linux and Android platforms.[1]

2) Experiment Orchestration and Testbed Implementation: **Defining, scoping, and monitoring experiments.** Thorough examination of FL algorithms should encompass experimentation over different datasets and different data distributions, with a varying number of clients involved and reporting, to name just a few experiment parameters that should be considered. Without an easy-to-use support for such experimentation, researchers either limit the richness of the experimental scenarios or develop their own infrastructure for such experimentation, which is both time consuming and error prone.

Furthermore, a realistic view of algorithm performance necessitates its realistic deployment. Constructing a full-fledged hardware testbed requires significant resources, both in terms of time and money. A high-end mobile device can cost about \$1,000, an SBC can cost up to \$500, while a high-frequency power meter costs about \$1,000. Equipping an FL testbed with a few dozen devices can be prohibitively expensive for many smaller research groups to do. Moreover, hardware failures (especially networking connectivity) and software updates require active DevOps effort to maintain testbed usability.

Performance metric collection and analysis. In Section I, we have demonstrated the need for assessing FL algorithms along different dimensions. Metrics, such as CPU/GPU utilization, energy and power consumption, memory consumption, data transfer sizes, and others, can paint a different picture of an algorithm's performance compared to merely inspecting the inference accuracy on a test set. However, capturing these metrics without requiring changes in the algorithm code and without affecting the execution of the algorithm is challenging. Moreover, the data should be collected throughout the experiment, reliably transferred and stored, and presented to the experimenter in an appropriate manner.

III. Related Work

Realistic experimentation is at the core of computer science research. With the rise of distributed and networked computing, a need for more elaborate testbeds has appeared. Consequently, testbeds tailored to allow multiple researchers to conduct relatively diverse experiments have appeared. Planetlab, for example, was a global testbed for computer networking services research that spawned over more than 1,000 distributed nodes at its peak [28]. Emulab, on the other hand, allows experimentation with various networking topologies that are emulated over a cluster of networking devices [29]. Building upon the Emulab software, the CMU wireless emulator enables remote emulation of wireless propagation conditions [30]. Also on the wireless front, the ORBIT testbed allowed indoor and outdoor evaluation of wireless protocols [31], while massive MIMO testbeds, such as [32], enable experimentation with future 5G and 6G wireless transmission protocols.

The above testbeds have had a significant impact on networking research. Yet, besides the networking aspect, practical

[1]Flower does not allow a mix of different platforms in the same FL setting.

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

FL encompasses mobile systems and ML aspects. Testbeds, such as CityLab [33], facilitate IoT systems research, however, do not cover mobile, especially smartphone-based, computation, and do not readily support distributed ML applications. ML testbeds, on the other hand, focus on cloud computing (e.g., CloudLab [34]) and do not support FL over edge devices.

FL research is supported by several open-source frameworks that have emerged in recent years [18], [35]–[46]. These frameworks typically provide APIs for users to express DL model architectures, data loading, and model training algorithms. Often, the frameworks separate the client-side training logic from the server-side aggregation logic. In several cases, such as with Flower [18], these frameworks include a simulation backend that allows an FL system to be run in a simulated environment without any substantial code changes.

Simulating FL is essential to expedite the prototyping of FL algorithms, system designs and evaluations thereof. Generally speaking, simulation can facilitate studying FL systems in different scenarios where the user can control the number of clients, the conditions in which they operate, the algorithms they execute, and other factors. To aid in simplifying large-scale simulation, several toolkits have been proposed to support simulating FL workloads [6], [47]–[54]. To supplement FL simulation with realistic characteristics of heterogeneous devices, Protea [55] proposes to profile devices to obtain information regarding resource consumption and computation time. Similarly, FedScale [48] builds a simulation infrastructure on top of realistic client device behavior traces in order to account for system-level heterogeneities that might affect FL. Nevertheless, certain aspects of FL performance, for instance, individual devices' energy consumption, cannot be reliably assessed through trace-based simulation due to the use of simplified models [56], [57].

Critical for FL experimentation is the need to take real-world heterogeneities into account. Several studies have analyzed heterogeneity at the level of data [5], [21], [58], [59], system [9], [10], and client availability [6], [8], and have shown that these factors significantly impact the performance of FL. Consequently, several benchmarks that include comprehensive data partitioning strategies to cover the typical non-IID data cases have been introduced [4], [7], [60]–[65].

Nevertheless, to this date, most FL research results have been obtained through simulation, and only relatively few studies (e.g., [66]–[70]) make use of actual experimental testbeds. Wong et al. perform a thorough study of FedAvg on a real-world edge computing testbed [71]. Their study demonstrates the utility of experimentation over heterogeneous hardware (the authors use Raspberry Pi 3/4, Jetson Nano and Jetson TX2), with various experiment settings (e.g., different data distributions), while different metrics (such as CPU utilization) are collected. Grounded in these findings, but not limited to FedAvg, CoLExT enables real-world experimentation on a wide span of hardware platforms, including 28 SBCs across 6 hardware types and 20 Android mobile phones across 8 models and 5 vendors, all while a wide range of metrics, from CPU utilization, to the amount of transferred data, to high-frequency energy measurements are collected.

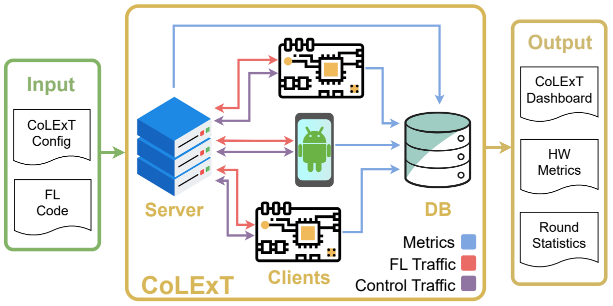


Fig. 2: CoLExT workflow.

IV. CoLExT: Federated Learning Testbed

In this paper, we develop CoLExT, a testbed for comprehensive experimentation with FL over real-world devices. CoLExT is designed as a solution for seamless deployment of arbitrary FL algorithms and supports a range of experimental scenarios. Characteristically, the testbed supports highly heterogeneous deployments and a comprehensive set of performance metrics.

The overview of the testbed is shown in Figure 2. The **CoLExT server** acts as a central entity for both experiment coordination as well as model aggregation within an FL algorithm. The experiment configuration, **CoLExT Config**, is provided by the experimenter. This configuration includes a reference to the code of the FL algorithm under test, **FL Code**, and the selection of **CoLExT clients** that will participate in the experiment. Clients can be selected from a range of devices, which, in the current implementation of CoLExT, include ARM and x86 SBCs (some having CUDA-capable GPUs) and Android smartphones.

The **CoLExT server** instantiates the experiment by packaging experimenter-provided FL code into deployable units and deploying these to the appropriate devices: for SBCs, CoLExT packages the code into containers and deploys them with Kubernetes; for Android smartphones, the code is integrated within the CoLExT Android app code, packed into an APK, and deployed with Android Debug Bridge (ADB). The **CoLExT server** also executes the FL server code provided by the experimenter. Separately, the experiment devices collaborate to collect relevant metrics. These are collected partly on clients and partly on the server and are transferred to the **CoLExT database** for further inspection.

Once the experiment is completed, the researcher will find a rich range of hardware and system metrics and statistics to gain insights into the performance of their FL algorithm, compare different FL algorithms based on the metrics, and even identify the performance difference across various devices. To facilitate the analysis, CoLExT also provides a Grafana-based **CoLExT dashboard** where the metrics collected during an experiment can be visualized in real-time.

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

A. *Using* CoLExT *in a Nutshell*

Interaction with CoLExT is designed to be succinct and in line with the workflow employed when using currently popular FL simulation environments [18], [36], [49], [50]. In a nutshell, the experimenter has to:

1) Access the CoLExT server running Python, and in a local Python environment, install our `colext` package.
2) In the FL code, import the above library and wrap the FL client and strategy code with CoLExT decorators. If used outside of the testbed, these decorators do not modify the program behavior and thus can safely be included in the code in general. An example of decorated client and strategy would be:

```
from colext import MonitorFlwrClient, MonitorFlwrStrategy

@MonitorFlwrClient
class FlowerClient(fl.client.NumPyClient):
[...]
@MonitorFlwrStrategy
class FlowerStrategy(flwr.server.strategy.Strategy):
[...]
```

3) Declare the pip `requirements.txt` file with the dependencies.
4) Declare the `CoLExT_config.yaml` experiment configuration file, which specifies the client and server entry points and the testbed devices on which the experiment will run. An example of such a file would be:

```
code:
  client:
    entrypoint: "client.py"
    args:
        - "--server_addr=${COLEXT_SERVER_ADDRESS}"
        - "--client_id=${COLEXT_CLIENT_ID}"
  server:
    entrypoint: "server.py"
    args: "--n_clients=${COLEXT_N_CLIENTS} --n_rounds=3"
devices:
  - { dev_type: LattePandaDelta3, count: 4 }
  - { dev_type: OrangePi5B,   count: 2 }
  - { dev_type: JetsonOrinNano, count: 4 }
monitoring:
  scrapping_interval: 0.3 # in seconds
  push_to_db_interval: 10 # in seconds
```

5) Deploy the experiment using the `colext_launch_job` command, after which the experiment performance metrics will be available for real-time monitoring on the CoLExT dashboard. Optionally, after the experiment is completed, the experimenter can call `colext_get_metrics` to retrieve the collected data in the form of CSV files from the CoLExT database. In the terminal, it would look like:

```
$ colext_launch_job --config <path-to-config>
# Prints a job-id and a Grafana dashboard link

# After the job finishes, retrieve metrics for job-id
$ colext_get_metrics --job_id <job-id>
```

V. CoLExT Implementation

A. *Underlying FL Framework*

CoLExT is designed to provide a realistic picture of FL algorithm performance in a real-world deployment, yet, at the same time, it aims to minimize the effort one needs to put into the experimentation. Therefore, rather than developing a custom solution for FL, we base CoLExT on an existing framework, offering researchers a convenient way to deploy their existing code with minimal modification.

From the available frameworks, we opted for Flower [18]. This framework offers a simple interface that facilitates FL research and is currently the most popular FL framework, with a large community of developers actively using and developing the framework.[2] Moreover, the Flower community led a substantial effort towards reproducing FL research, which has generated a pool of high quality baselines coded against the Flower API. From the technical side, Flower supports synchronous FL, relying on gRPC [72] protocol, and ensures that on-client training, on-server aggregation, and communication between the FL clients and the server are executed.

While CoLExT relies on Flower, it is by no means locked into using this framework. Indeed, Flower could be substituted with any other framework that supports on-device execution of FL, as long as, besides the FL training control commands, the framework exposes API hooks for indicating the beginning/end of an FL round at the server and the beginning/end of the local training on the client.

B. CoLExT *Client and Server*

The majority of FL research has been developed in Python and evaluated on Linux-based x86 machines due to the ease of use of this platform. Real-world FL deployments, on the other hand, are expected to include other platforms, such as the Tegra architecture of NVIDIA Jetsons, or even different operating systems, such as Android. With CoLExT, we aim to support code execution on different devices with minimal effort from the experimenter's side.

Android and Linux-based clients are handled differently in Flower. Consequently, in CoLExT we devise a separate Linux FL client and an Android FL client.

Linux client leverages containers cross-compiled using Docker Buildx to support multiple architectures, including AMD64 and ARM64. These containers are then stored on our private container registry using Harbor [73]. CoLExT expects that experimenters have their FL code written in Python and

[2]Flower GitHub page has been "starred" 4.4K times, while the TesorFlow Federated page has accumulated 2.3K stars. In addition, the Flower website states that the framework is associated with "The world's largest Federated Learning conference" – Flower AI Summit 2024.

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

will use the supplied pip `requirements.txt` file to install the dependencies within the container.

Due to the unavailability of specific Python packages on the Python Package Index (PyPI) for some of the SBC architectures used in our testbed, we manually pre-compiled multiple versions of certain Python packages (such as PyTorch with GPU support for ARM) and enabled their inclusion during the container packaging process.

In the client code, CoLExT expects that CoLExT decorators have been applied to automatically collect performance metrics from the FL code. Finally, while the common situation of an experimenter providing the Python code with the pip `requirements.txt` file is handled through automatic container packaging (the configuration specification is described in subsection IV-A), other setups, such as those involving custom dependency compilation, are possible with manual container building.

Android client is based on the CoLExT Android app, within which an experimenter copies their Java/Kotlin code defining the client behavior. The provided app exposes the TensorFlow Lite support for on-device DL training, handles communication with the server, and collects performance metrics.

FL server is provided as a Python script, irrespective of whether the clients are SBCs or smartphones, and is expected to run on a Linux x86 machine. Similar to the Linux clients, the FL server code is containerized with all its dependencies installed. CoLExT expects the server code to utilize the server-side CoLExT decorator to collect performance metrics, e.g., round timings. Additionally, the container is configured to have access to the GPU on the host machine in case the server code can benefit from the accelerator.

C. Datasets and Data Partitioning

Regardless of client type, clients must obtain their dataset at the start of each experiment. To avoid repeated downloads, commonly used datasets in FL research, such as CIFAR [74] and MNIST [75], are cached on all devices. Users can include additional datasets to be cached if needed.

Note that since the entire dataset is installed on each device, clients still need to determine which data points comprise their specific subset. To stay general, CoLExT is oblivious to specific data partitioning schemes, and we assume the responsibility of devising data partitions lies with the user.[3]

D. Collecting Performance Metrics

In CoLExT, we aim to seamlessly capture experiment performance metrics with minimal modification to the FL research code and also support intuitive and informative visualization of the captured metrics.

The performance of FL has traditionally been evaluated along a single dimension – inference accuracy of the resulting model. CoLExT naturally supports the collection of this metric by logging the returned accuracy values from Flower's server and client evaluation functions. Nevertheless, as shown in our introductory example in Figure 1, metrics, such as the energy consumption per device, execution time, and others, are necessary for a holistic evaluation of FL algorithms. In CoLExT, such metrics are collected by a background scraper that periodically records them, timestamps them with the local time, and aggregates the data in batches before sending them to CoLExT database. The measurements are unrelated to FL rounds. However, as the devices in the testbed are synchronized using NTP, CoLExT can associate the performance data with rounds by grouping the data according to the round start/finish time.

The way the metrics are collected differs between Linux and Android clients. **In Linux**, standard hardware metrics such as CPU, memory, and network utilization are obtained using the `psutils` Python package from a separate background process. Power consumption, however, is tracked differently for different devices. Thus, for NVIDIA Jetsons, we obtain the power consumption of the entire board using `jetson-stats` monitoring tool. For LattePanda devices, on the other hand, such a tool is not available, and we capture the CPU power consumption using Intel "Running Average Power Limit" (RAPL) through the `pyRAPL` Python package. Finally, for OrangePi devices, no suitable software solution exists; thus, we resort to physical power consumption measurement using the Monsoon power meter [76].

Regarding GPUs, even though all SBCs have one, we were only able to train ML models on NVIDIA Jetsons' GPUs. The other GPUs, the Intel HD Graphics GPU on the LattePanda and the Mali GPU on the Orange Pi, lack support from major ML frameworks, as these GPUs primarily focus on graphics processing, thus have limited memory and lack support for precision formats required for machine learning. NVIDIA Jetson integrated GPUs, on the other hand, support CUDA and, for the purpose of deep learning, can be treated as discrete GPUs. Metrics for NVIDIA Jetson GPUs were collected using the `jetson-stats` package.

In Android, accessing performance metrics is more challenging than in Linux. While in the past, parsing `/proc/stat` allowed one to retrieve device utilization metrics, in newer versions of Android (as of Android 12), this is not available anymore. We therefore develop a set of techniques for accessing various metrics on Android devices. First, we assign special privileges to our app in Android OS, where we register the app as the device owner. Then, we use the official memory utilization API to obtain the memory usage and `/sys/devices/system/cpu` scraping for CPU utilization statistics. Obtaining GPU usage statistics is highly challenging, as there is no official API for GPU statistics on Android, nor do the usage statistics files necessarily exist on the file system. Our investigation with multiple phone makers, and models (Google Pixel 7, Xiaomi 12, Samsung Galaxy S21 FE, M54, XCover 6 Pro, ROG Phone 6, OnePlus Nord 2T 5G, and Xiaomi Poco X5 Pro) finds that only Samsung devices reliably expose the GPU utilization files, and only in case the

[3]For instance, one option is to assign dataset subsets to clients by generating a CSV file for every client, listing the data point indices to be used, as shown in https://github.com/sands-lab/flower_dcml_algorithms.

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

TABLE I: COLEXT environment variables.

Environment Variable	Description
COLEXT_SERVER_ADDRESS	Server address (host:port)
COLEXT_N_CLIENTS	Number of clients
COLEXT_CLIENT_ID	Client ID (0...n_clients)
COLEXT_CLIENT_DEV_TYPE	Client device type

devices are rooted. We, thus, collect GPU statics for Samsung devices within the COLEXT testbed. Finally, when it comes to power consumption, the official Android `BatteryManager` API allows us to collect power consumption attributed to different apps and the system as a whole, all from our app that was previously given device owner privileges.

E. *Experiment Orchestration*

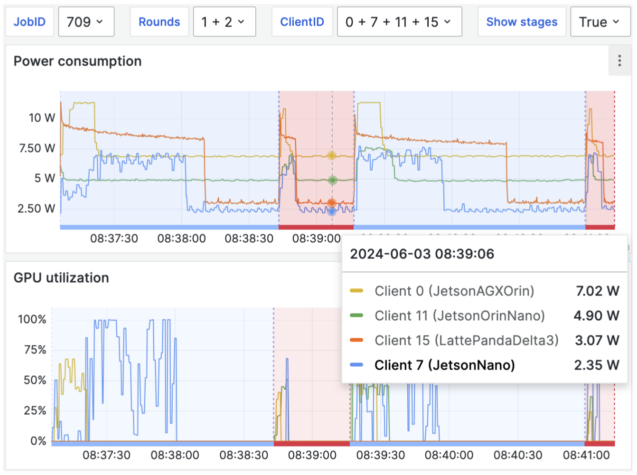
COLEXT automates the deployment and running of FL experiments on real-world clients. Since the underlying FL framework handles Android and Linux clients separately (see also Section V-B), the experiment orchestration varies between SBCs and smartphones.

For SBCs, we create a Kubernetes cluster containing Linux clients and the server. We use the `microk8s` orchestration tool as it is geared towards edge devices and conveniently available as a `snap` package, which isolates the Kubernetes installation from the host filesystem while natively running the entire Kubernetes stack without the need for containers. Container deployment in Kubernetes requires configuring pods, the deployment unit in Kubernetes. COLEXT prepares FL client and server pod configurations using `Jinja2` template files that can be configured with the required device type for the client, entry point, mounted directories for dataset caching, and added COLEXT related environment variables (listed in Table I), including a client identifier and device type.

Exposing and utilizing GPU computation through a Kubernetes container is done through `microk8s`, yet the support is limited to discrete GPUs and is incompatible with integrated GPUs of NVIDIA Jetsons. To overcome this, we configured the underlying container runtime on those devices to use the NVIDIA container runtime as the default runtime, which exposes the GPU to any containers running on the device.

Finally, we also had issues with the default `microk8s` Container Network Interface (CNI), Calico, because the `ipset` kernel module is missing from Jetsons. To avoid this issue, we switched to another CNI, Flannel [77].

For smartphones, a Kubernetes-based solution is not an option. Instead, we build our own deployment system using a combination of Python scripts, Bash scripts, and ADB. ADB allows us to administer Android devices and issue commands for installing and running our applications, while Python scripts, along with Bash commands, provide an interface between COLEXT server (also written in Python) and deployment scripts. The smartphones are connected to the server in Debug mode, which allows us to transfer files, install the refreshed application (if needed), and run the application for the clients through ADB commands.


JobID 709 · Rounds 1 + 2 · ClientID 0 + 7 + 11 + 15 · Show stages True

2024-06-03 08:39:06

Client 0 (JetsonAGXOrin)	7.02 W
Client 11 (JetsonOrinNano)	4.90 W
Client 15 (LattePandaDelta3)	3.07 W
Client 7 (JetsonNano)	2.35 W

Fig. 3: Example capture of COLEXT Dashboard.

F. COLEXT *Dashboard*

COLEXT Dashboard provides a visual depiction of the collected performance metrics. To avoid cluttering the dashboard, an experimenter can limit the metrics to a subset of rounds and clients. A cropped screenshot of the dashboard can be seen in Figure 3. Hovering over the graph provides information across different clients, while the x-axis denotes time.

The dashboard also allows highlighting periods the FL algorithm spends on different training stages (“Show stages”): training in blue, and evaluation in red. By clearly distinguishing between these two phases, the dashboard helps identify patterns of each phase and reveals the cause of metric spikes (power and GPU utilization) that are present in the example depicted in Figure 3.

In addition to per-client metrics, the dashboard also contains a section with aggregated metrics over device types to assist with cross-device comparison. Finally, to simplify debugging, COLEXT can also collect logs and display them directly in the dashboard while the experiment is running.

VI. COLEXT TESTBED DEPLOYMENT

We deployed COLEXT testbed in a dedicated server room at our institution premises. The testbed comprises of heterogeneous edge devices, including SBCs with integrated GPUs, such as the NVIDIA Jetsons, x86 SBCs, such as LattePandas, ARM-based SBCs, OrangePis, and Nvidia Jetsons, and Android devices, selected based on their AI task performance scores from the benchmark AI-Benchmark [78], so to cover low-, middle-, and high-end phones. In total, 48 devices, of which 28 are SBCs of 6 model types, and 20 Android smartphones of 8 models (5 vendors), are included in the testbed. More details, including the quantity of each device type, are present in Table II, while a picture of the testbed can be seen in Figure 4. Furthermore, a server-grade machine equipped with an Intel(R) Xeon(R) Gold 6442Y CPU (48 cores @ 2.6GHz), 256GB of RAM, and an NVIDIA RTX A6000 GPU acts as the COLEXT server. Finally, the testbed

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

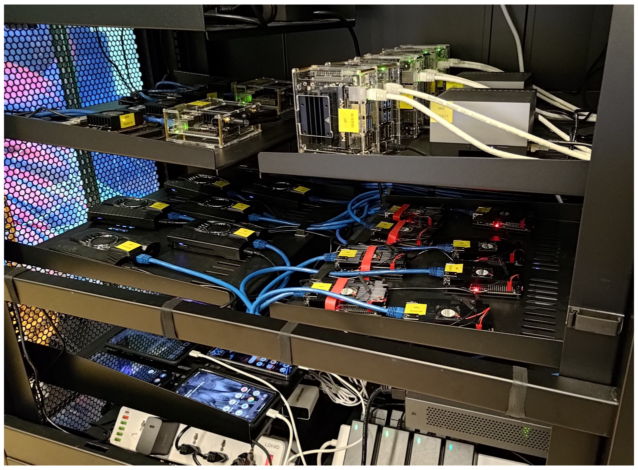


Fig. 4: **COLEXT testbed devices.**

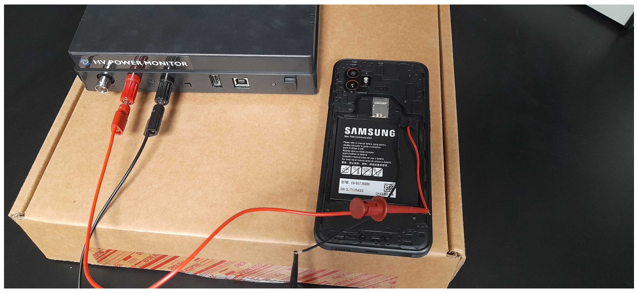


Fig. 5: COLEXT Samsung Galaxy XCover 6 Pro powered by a Monsoon PM.

also includes a Monsoon High Voltage Power Monitor (PM), which acts as a power source, while simultaneously monitoring the amount of power supplied to the device. PM measures current and voltage at a sampling rate of 300 kHz. PM can power SBCs directly through appropriate pins. For Android devices, the battery must be removed and the PM's wires need to be connected to the battery pins. The Samsung Galaxy XCover series, with its removable batteries, offers the most straightforward implementation of the above, depicted in Figure 5.

The testbed operates over its dedicated network. COLEXT devices are connected through a switch and a WiFi access point (AP) – the 28 SBCs, the FL server, and the WiFi AP are connected to a switch using Ethernet cables, while the 20 smartphones connect to the AP via WiFi. The FL server runs a DHCP service, configured with statically assigned IPs, and is the gateway to the Internet.

The testbed devices are pre-configured with the appropriate operating system and software environment. For SBCs, we opt for Ansible playbooks to automate the configuration of the devices, including the configuration of the NTP server and the installation and configuration of microk8s, which also adds the node to the Kubernetes node pool. For Android, some device configuration is possible through ADB, including the NTP server setup, but other configurations, like the connection to ADB, require manual intervention. Due to hardware differences, the OS and the environment vary across SBCs. The OS also varies for Android devices, depending on the vendor. The OS environment for every device type is described in Table II. The testbed devices are pre-configured with the appropriate operating system and software environment. For SBCs, we opt for Ansible playbooks to automate the configuration of the devices, including the configuration of the NTP server and the installation and configuration of microk8s, which also adds the node to the Kubernetes node pool. For Android, some device configuration is possible through ADB, including the NTP server setup, but other configurations, like the connection to ADB, require manual intervention. Due to hardware differences, the OS and the environment vary across SBCs. The OS also varies for Android devices, depending on the vendor. The OS environment for every device type is described in Table II.

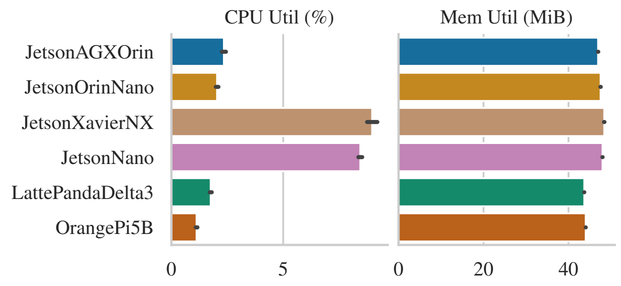


Fig. 6: The overhead of performance metric collection on SBCs. The results show average CPU and memory utilization over 1,000 metric collection events, with error bars indicating the 95th percentile. On each platform the CPU utilization remains below 9%, while the memory utilization remains very low (< 50MiB).

VII. VALIDATING COLEXT

We validate COLEXT for its ability to provide the metrics of interest without significant overhead and its ability to support a range of FL algorithms out-of-the-box.

Throughout this section, we use the CIFAR-10 dataset [74], which is widely employed in FL research and hence allows us to validate the results we obtain. We also use relatively small models (35k to 380k parameters). Nevertheless, COLEXT is agnostic to both the models and the datasets the clients use. Lastly, devices use their default performance configurations, with the exception of NVIDIA Jetsons, whose power mode is set to the highest setting to optimize performance.

A. Quantifying Metric Collection Overhead

Frequent sampling of performance metrics may require significant resources of the host machine, which, in turn, could affect the execution of the FL task. To ensure this is not the case in COLEXT, we profile the metric scrapper on SBCs. The profiling ran for 5 minutes, scrapping every 0.3 seconds and pushing metrics to the database every 10 seconds. In total,

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

TABLE II: Testbed device specifications. SBC's AI Performance was retrieved from NVIDIA Jetson Benchmarks [79]. Score values for Android devices were obtained from AI-Benchmark [78]. Unavailable scores are shown with a dash.

Single Board Computer (SBC)					
Device	Qty	CPU (Core@GHz)	Mem (GB@GHz)	AI Perf (TOPS)	OS environment
Jetson AGX Orin	2	12@2.2	64@3.2	275	Jetpack 5.1.2 + Linux for Tegra 35.4.1
Jetson Orin Nano	4	6@1.5	8@2.1	40	Jetpack 5.1.2 + Linux for Tegra 35.4.1
Jetson Xavier NX	2	6@1.9	8@1.8	21	Jetpack 5.1.2 + Linux for Tegra 35.4.1
Jetson Nano	6	4@1.5	4@1.6	0.472	Jetpack 5.1.2 + Linux for Tegra 32.7.4
Latte Panda Delta 3	6	4@2.9	8@2.9	–	Ubuntu server 22.04
Orange Pi 5B	8	8@2.3	16@2.3	–	Ubuntu server 22.04 [80]
Android Mobile Phone					
Device	Qty	SoC	Mem (GB)	Phone score / SoC score	OS environment
Asus ROG 6	2	Snapdragon 8+ Gen 1	16	1447 / 1000	Android 13
Xiaomi 12	2	Snapdragon 8 Gen 1	12	1355 / 1046	Android 12
Google Pixel 7	5	Google Tensor G2	8	720 / 525	Android 13
Samsung XCover 6 Pro	3	Snapdragon 778G	6	– / 257	Android 13
Xiaomi Poco X5 Pro	2	Snapdragon 778G	8	– / 257	Android 12
Samsung Galaxy S21 FE	2	Exynos 2100	8	262 / 196	Android 13
OnePlus Nord 2T 5G	2	Dimensity 1300	8	240 / 177	Android 12
Samsung Galaxy M54	2	Exynos 1380	8	– / 75	Android 13

we collected 1,000 sample points. We assess the excess CPU and memory usage caused by such scraping in Figure 6.

For most devices, we see that CPU usage stays below 2.5%, i.e., remains rather insignificant. The lowest CPU utilization is associated with the OrangePi, which only collects metrics from the `psutils` package. The second lowest utilization belongs to the LattePanda devices, where, on top of `psutils`, CoLExT resorts to `PyRAPL` to collect power measurements, hence requiring additional CPU utilization. Jetson AGX Orin and Orin Nano experience increased CPU usage, as these devices call `psutils` and then `jetson-stats` for power measurements and GPU utilization. Finally, we see a rather unexpected spike in CPU usage for Jetson XavierNX and Jetson Nano, where the CPU utilization is over 3 times higher than for the other Jetson devices, despite using the same data collection method. After profiling the code and benchmarking the CPUs with the `sysbench` program, we discovered that the CPUs of these devices are approximately twice as slow as the CPUs of other Jetson devices. However, the CPU utilization of XavierNX can be reduced by using a power mode that favors CPU speed by using fewer cores (6@1.4GHz or 2@1.9GHz). Nevertheless, we believe that with less than 10% CPU overhead, the burden imposed by fine-grain metric collection remains acceptable irrespective of the device type, while the rate of metric collection can be reduced in case lowering the overhead is necessary.

In Figure 6, we also depict the memory usage, which is shown to stay consistently low across devices when metrics are periodically pushed to the DB. However, if metrics are not periodically pushed, they are stored in memory, causing memory utilization to increase by about 10 KiB for every 1k samples collected, as each sample (of all the metrics) requires an average of 10 bytes. Related to this, the network usage is also determined by how frequently metrics are pushed to the DB. Given the small size of the metrics, only minimal bandwidth is required. Moreover, potential interference of data transmission and the operation of the FL experiment can be entirely avoided by sending the metrics to the DB only after the experiment has finished.

TABLE III: FL algorithms tested on CoLExT SBCs.

Algorithm	Deployed	Issues
FedAVGm [11]	✓	Aarch64 & TensorFlow on LattePanda
FedProx [12]	✓	-
Moon [13]	✓	GPU only
FedNova [16]	✓	-
FedPara [81]	✓	Dataloader segfault LattePanda + OrangePi
HeteroFL [19]	X	Code decoupling needed
FjORD [82]	X	Unsupported serialization

B. Supporting Different FL Algorithms

1) Porting FL algorithms on SBCs: To confirm CoLExT's ease of use and compatibility with different FL algorithms, we execute a collection of FL algorithms on our CoLExT testbed. We select these algorithms from open-source implementations developed within the Flower "Summer of Reproducibility" initiative [83], during which a cash reward was provided for the Flower-based implementations of published FL algorithms. Currently, this collection includes 20 baselines, of which seven (listed in Table III) were used for our experiments.

These baselines are written so that their folder structure and code organization are uniform, which makes comparison across algorithms straightforward. Nevertheless, due to varying authorship, the coding style, code efficiency, and package dependencies varied noticeably among the algorithms. Thus, we believe that with this set of algorithms, we can comprehensively test the CoLExT's ability to support different FL algorithms. Note, however, that all of the baselines were initially implemented with a simulation environment in mind, and consequently, certain modifications are necessary to get them working in a real-world client-server deployment. In case the algorithms do not assume shared data between clients and the server, and when the data communicated with the server can be serialized by the client, the required changes are minor. In all cases, the entry script needs to be updated to support the separate start-up of client and server as follows:

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

```
# Before: Simulation
history = fl.simulation.start_simulation(
    client_fn=client_fn,
    num_clients=cfg.num_clients,
    config=fl.server.ServerConfig(cfg.num_rounds),
    strategy=strategy,
)

# After: Client - Server
#   is_client, is_server, server_addr, num_rounds, client_id:
#   passed as arguments via the configuration file
if is_client:
    fl.client.start_numpy_client(
        server_address=cfg.server_addr,
        client=client_fn(cfg.client_id),
    )
elif is_server:
    fl.server.start_server(
        server_address="0.0.0.0:8080",
        config=fl.server.ServerConfig(cfg.num_rounds),
        strategy=strategy,
    )
```

As we identify in Table III, we successfully complete the experiments with five out of seven selected algorithms – two baselines do not comply with the above requirements and hence cannot be deployed on the testbed without corrections to their codebase. The process was relatively straightforward, and besides the expected augmentation of the code (i.e., 3 lines of code for decorating the FL client and strategy, detailed in Section IV-A, and the above-shown changes to the entry script), we had to take the following additional steps: 1) Flower uses Poetry [84] to specify dependencies, thus we had to convert the list of dependencies into the format CoLExT supports, i.e., the pip `requirements.txt` file, using the `poetry export` command; 2) package dependencies for aarch64 sometimes needed adjustments. From our analysis, all baselines were tested on an x86 machine. However, when we try to use the same (identical) dependencies, we sometimes find that minor versions of packages are not available on aarch64. When we encounter these issues, we bump the minor version for one that supports aarch64; 3) we had to replace the use of dependencies explicitly targeting x86 architectures. In Flower baselines, torch and torchvision dependencies are specified by prebuilt wheels targeting x86. We had to comment out these dependencies so that aarch64 wheels could be used for our aarch64 devices.

We now describe the challenges encountered when porting the baselines. For FedAVGm, we had to degrade TensorFlow from 2.11.1 to 2.11.0 because a dependency required by the original version was not available for aarch64. Additionally, the PyPI TensorFlow wheel for x86 CPUs assumes support for the AVX instruction, which is not available on the LattePandas, so they cannot use those wheels. Certain parts of the Moon baseline code assume that a GPU is available for the movement of data to/from the GPU. This prevents the experiment from running on CPU-only devices. FedPara is deployable, but we encountered issues with the PyTorch DataLoader module, causing segmentation faults on OrangePis and LattePandas. The exact cause remains unclear, but it could be related to the data partitioning strategy not supporting a reduced number of clients. The HeteroFL baseline requires some code decoupling to work in a client-server setup. In the simulated environment code, the client information is assumed to be available to the server before the experiment starts, but in a client-server setup, it needs to be shared by the client. FjORD's client implementation attempts to send nested dictionaries and model weights, whereas Flower supports only string-to-scalar dictionaries, thus, the algorithm was not portable.

To confirm that the experimentation in CoLExT was not only successful but also produced the expected results, we executed the Moon [13] baseline and compared it with the FedProx [12] algorithm with 10 clients using the CIFAR-10 dataset as instructed in the README for the Moon baseline. The accuracy achieved for the final models was slightly lower than the values reported in the README, more specifically 0.1% and 1.4% lower for Moon and FedProx, respectively. This discrepancy can be explained by the alteration of the random number generator sequences when switching from a simulated environment to a client-server setup.

In conclusion, the experiments conducted in CoLExT demonstrate that the FL code can be successfully deployed in the testbed environment with only minor potential variations in accuracy compared to the simulated environment.

2) Porting algorithms to Android devices: On Android devices, Flower's support limits us to use TFLite. One significant constraint of TFLite is its lack of support for stateful FL algorithms. To confirm the support of different algorithms in CoLExT, we needed to port stateless FL algorithms to Android.

Given the limitations, we developed an application that supports stateless algorithms, and we implemented and tested the algorithms presented in Table IV. The algorithms were ported from the Flower code repository. FedYogi and FedAdam modify the adaptive momentum estimation (Adam) optimizer to stabilize the learning process and improve convergence in heterogeneous federated settings. FedAdagrad adapts the Adagrad optimizer for FL by using per-parameter learning rates that adjust based on the history of gradients, accommodating non-IID data distributions. FedOpt serves as a generalized framework for federated optimization, encompassing various adaptive optimizers providing flexibility and robustness.

All of the stateless algorithms only require changes on the server side. Currently, the implementations in the Flower repository for all algorithms support serialization designed for Python implementations. To make the algorithms work with Android, we had to make certain changes to the code. The initial code from Flower relies on NumPy for the serialization of weights transferred to the clients and back. As we are running these clients on Android, we do not have access to NumPy, so we harnessed the existing implementation of a suitable serialization that is present in the FedAvgAndroid code from the Flower repository. Furthermore, the above algorithms require a random starting model. We added a mechanism similar to what FedAvg uses, whereby if there is

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

TABLE IV: FL algorithms tested on COLEXT smartphones.

Algorithm	Deployed
FedAvg [3]	✓
FedAdam [85]	✓
FedYogi [85]	✓
FedAdagrad [85]	✓
FedOpt [86]	✓

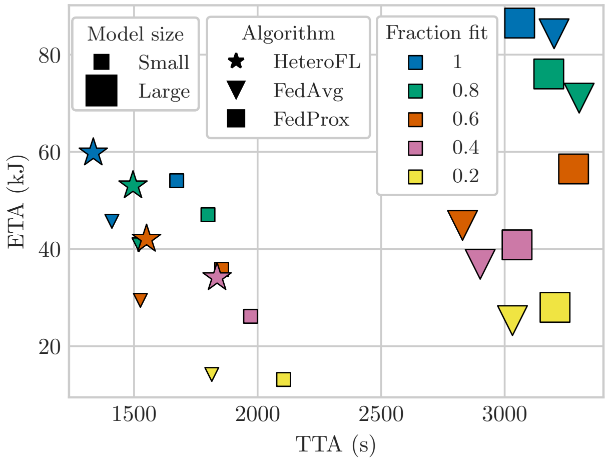


Fig. 7: TTA and ETA consumed to reach 64% accuracy on CIFAR-10 with 20 clients and Dirichlet distribution ($\alpha = 1.0$). Missing configurations failed to reach the target accuracy for three consecutive evaluation rounds.

no initial model present at the server, such a model is pulled from one randomly chosen client and broadcast to the others.

VIII. PROFILING WITH COLEXT

Heterogeneity of both SBC and smartphone devices included in our testbed, support for a broad span of algorithms and datasets, together with a range of performance metrics produced by the framework, ensure that COLEXT provides a rich experimentation space for FL research In this section, through a small number of use cases, we show how COLEXT experiments can be used to improve our understanding of FL.

A. Revealing Accuracy vs. Resource Usage Trade-off

The final model inference accuracy is the most reported metric in FL research. Yet, the accuracy is seldom juxtaposed against the resources needed to achieve it. In Figure 1 of section I, we reveal that substantial energy costs can be incurred to achieve a very modest gain in accuracy. We now expand this investigation and assess how both the time and the energy vary as we aim to achieve a certain inference accuracy through different FL algorithms and settings.

We focus on the SBC clients in our testbed and assign both a training and a validation dataset sampled from the same distribution to each client, whereas distributions differ among the clients (i.e., non-IID data) according to the Dirichlet distribution with parameter $\alpha = 1.0$. We employ FL with different algorithms (FedAvg, FedProx, and HeteroFL), using different model sizes (*Small* – 35k and *Large* – 380k parameters) and a different number of clients harnessed in each round. After each training round, each client evaluates the global model on its validation dataset and reports the resulting accuracy to the server. The experiment continues until the average validation accuracy across all clients achieves a pre-defined target value for three consecutive evaluation rounds.

We are interested in resources spent for training, thus we measure the wall clock (i.e., time-to-accuracy – TTA) and the energy (i.e., energy-to-accuracy – ETA) required to reach the target accuracy. To determine the energy spent on training, we first measure the average idle power consumption of the devices and subtract this "average idle power" from the power measurements during training. For instance, our experiments indicate that Jetson XavierNX consumes $2.9W$ (Watt) at rest. Hence, when such a device consumes, on average, $5W$ during the experiment, we consider that $2.1W$ is the "average active power" used for training. We then compute the energy used for training by multiplying the "average active power" by the time when the device is actively learning.

In Figure 7, we compare TTA and ETA for the three algorithms when the target accuracy is set to 64%. We observe that using the *Large* model (denoted with larger symbols) significantly increases the training time. A close look reveals that the cause for this lies in stragglers, as the server waits until it receives the updated models from all the sampled clients. Note that the FedAvg (square symbols) and the FedProx (inverse triangle symbols) algorithms require all the clients to train the same model architecture, while the HeteroFL algorithm (star symbols) goes beyond this limitation, allowing clients to train a model with a size proportional to their computational power. Consequently, the TTA for HeteroFL is comparable to the algorithms in which all clients train the *Small* model (denoted with smaller symbols). For the cluster of points on the left of the figure, which reflect the training of the *Small* model with the FedAvg and FedProx algorithms and heterogeneous model sizes with HeteroFL, we see a trend: increasing the percentage of clients sampled for training in each training round ("fraction fit") decreases the TTA, as each update to the server model is obtained with more data. However, this causes an increase in ETA as more devices are used for training.

B. Revealing (in)Efficiency of On-device Training

COLEXT can also be used to assess the efficiency of different device types when handling the same workload.

We focus on the Moon and FedProx implementations that we experimented with in subsection VII-B. To profile efficiency, we measure the time and the energy required to process one batch of data, calculated by dividing the measured time and energy required to finish a training round by the number of batches in the round. We deploy and run the algorithms on GPU-enabled SBCs in our testbed.

The results, shown in Figure 8 (top-left), reveal that the fastest device is, unsurprisingly, AGXOrin, which is also advertised as the most powerful SBC in our setup. However,

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

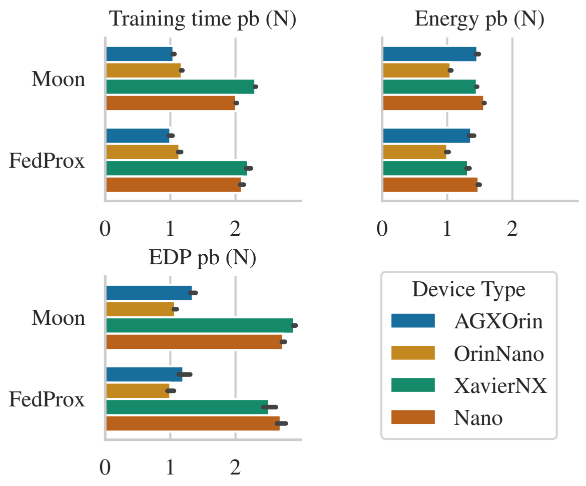


Fig. 8: Comparing per batch (pb) efficiency of four NVIDIA Jetson models using the Moon and FedProx algorithms. All metrics have been Normalized (N) to the smallest value. The plotted data represents averages over 100 round samples, with error bars indicating the 95th percentiles.

for both algorithms, AGXOrin is not significantly faster than OrinNano, yet it uses noticeably more energy (top-right plot). Thus, we ask: How can we quantify whether the decrease in training time justifies the additional energy cost?

This led us to consider another metric common in digital electronics, the *Energy Delay Product* (EDP). This metric multiplies the time taken to complete a task with the energy required for the task completion. Thus, a (preferred) low EDP indicates that a solution is both fast and energy-efficient. In Figure 8 (bottom-left), we observe that, according to EDP, OrinNano is the most efficient device in both algorithms. In other words, considering our subset of devices, if we only use OrinNano devices for this algorithm, we would minimize the time taken to train a batch for a given energy budget and vice-versa. The data also shows that XavierNX appears to be the slowest device, despite being a more recent model and having better hardware compared to the oldest model, the Jetson Nano (Table II for device specifications). However, when FedProx is employed, despite XavierNX being the slowest device, it is still more efficient than a faster device, as is shown by the lower EDP value compared to Jetson Nano.

Note that our comparisons were done without fine-tuning the boards (e.g., the CPU operating frequency, etc.) for optimal efficiency;[4] it is possible that running AGXOrin in a lower power mode could make it more efficient than OrinNano. Nevertheless, these examples demonstrate how EDP helps identify which devices are more efficient for specific algorithms and how device efficiency changes across algorithms. Another interesting use of EDP is to evaluate algorithm modifications to determine efficiency improvements.

[4]Users can update the power mode and CPU frequency of the devices; however, currently this cannot be done via the CoLExT configuration file.

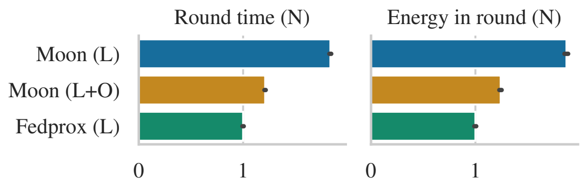


Fig. 9: Normalized (N) round level metrics for Moon and FedProx using an 11.6M parameter model (L), with the addition of a small optimization (O) to the original Moon baseline code. Results show average round data statistics over 30 rounds, with error bars indicating the 95th percentiles.

With these insights, CoLExT provides users with a valuable tool to compare device efficiency, enabling them to observe and address potential performance issues with FL on real-world devices.

C. Revealing Algorithm Implementation Issues

The testbed also allows us to analyze the algorithms' performance at the FL round level. We once again turn back to the SBC-based experiment we performed with FedProx and Moon in the previous section. The algorithms are rather similar, yet we expect that results will reveal the cost of extra forward passes conducted by Moon – compared to FedProx, which completes one forward pass, Moon performs three (one for the client model, two for the server model) in each round. However, the results of the experimentation (depicted in Figure 8 left) show that both algorithms take a very similar amount of time per batch (and, consequently, per round). To further unpack the issue, instead of using the *Small* model, which might prevent the expected differences from being noticed, we tasked the algorithms with training the *Large* model. This larger model exceeds the 4 GB memory of the Jetson Nano devices, so this experiment was only done with other Jetson device types. The results, shown in Figure 9 (with L indicating the *Large* model training), indicate that the difference in time between the two algorithms is obvious, if not unexpectedly large.

Surprised by such a large difference in the execution time (Moon $1.75\times$ slower), we further inspected the code and identified unnecessary data movements from the GPU to the CPU memory in the Moon codebase. By addressing this and optimizing the code (denoted with L+O in Figure 9), the results finally get in line with the theoretical expectations regarding the two algorithms.

This experiment highlights how code optimizations and model size modifications can drastically change how two algorithms compare in terms of the training time, making their implementation and real-world performance analysis critical.

D. Identifying Causes of Stragglers in Real-world Mobiles

We perform extensive experimentation with Android devices in CoLExT and identify two main reasons for devices becoming stragglers, both due to hardware heterogeneity:

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

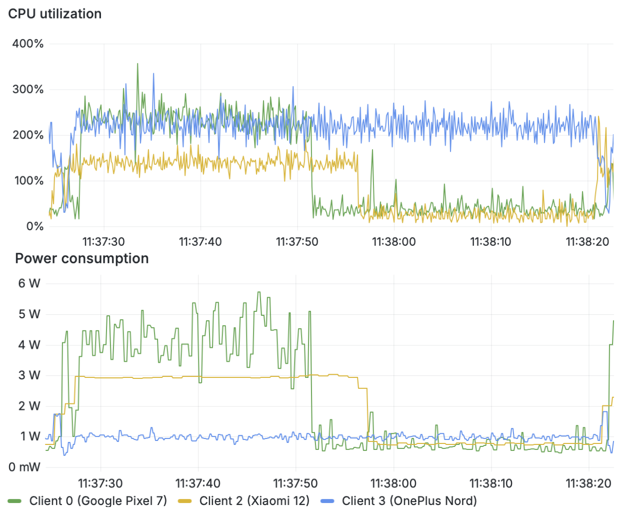


Fig. 10: CPU utilization and power consumption on heterogeneous mobile hardware. Data collected over 1 round of training on CIFAR-10 dataset.

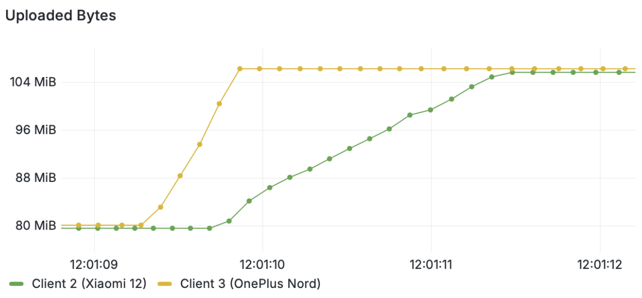


Fig. 11: Uploaded bytes over time after a round of training on FEMNIST dataset [87] using CNN model.

Compute speed. Hardware heterogeneity introduces varying compute capabilities among devices, which is particularly noticeable during model training. The top graph of Figure 10 shows the CPU utilization and power consumption during one round of training on three devices: Google Pixel 7, Xiaomi 12, and OnePlus Nord 2 5G. The start and end of local training can be identified by shifts in CPU utilization, from near idle (close to 0%) to high utilization (over 100% – indicating that multiple CPU cores are active) and then back to idle. We observe that OnePlus Nord takes the longest to complete a training round, with almost 2.3× longer round time than the fastest participating device in this training (Google Pixel 7). In the bottom graph of Figure 10, significant power consumption spikes are evident for two devices, contrasting with the OnePlus as a result of it being severely underclocked. Despite having similar CPUs, the default underclocking of the CPU in OnePlus makes a substantial difference in a real-world deployment of FL.

Data reception and transmission. Even with similar computing capabilities, devices can vary significantly in other hardware components, notably the wireless communication module (WiFi/cellular chip). With larger models requiring the transfer of many weights, differences in data transmission speeds become more evident. Figure 11 illustrates this, showing the Xiaomi 12, with the default transmission parameters, as a straggler, taking 3.2× longer transmission time for the same amount of data compared to the OnePlus 2T 5G.

IX. Limitations and Future Work

Android & SBC deployments. Deep learning has evolved separately on desktop and server environments, where it is often implemented in Python, harnessing PyTorch or Keras libraries, and on mobile devices, where deep learning is usually based on the TensorFlow Lite (TFLite) library and written in Kotlin or Java. Consequently, FL is also supported in different manners on the two groups of devices present in the CoLExT testbed – SBCs and Android phones. While we rely on Flower, due to its support for both of these device groups, we still face the barrier of different serialization formats used by PyTorch and TFLite, which hampers FL over a group of clients where Android and SBC devices are intermixed. A potential solution could be a compatibility-providing mapping between the PyTorch and TFLite serialization formats, something that ONNX [88] promises to provide. However, our extensive experiments with such a mapping failed to produce a reliable solution. Alternatively, deploying TFLite on single-board computers (SBCs) could be considered. However, this approach is not widely adopted among FL researchers, and TFLite may not offer the same level of usability as PyTorch for certain applications. In future versions of CoLExT, we aim to address this issue and enable a combination of models trained with SBC and Android devices to be used within the same experiment.

Network conditions. Network variability is a common property of edge environments where wireless networks are the norm. CoLExT currently does not support controlling network conditions, but preliminary work on that front has already been conducted. For SBCs, we are adding support for fixed latency and bandwidth settings on clients, which could be configured using the Linux `tc` tool. Due to the difficulty of configuring Android clients in the same manner, for smartphones, we plan on supporting network parameter modifications on the server side only.

Concurrent users. Given the size of our device pool, 28 SBCs and 20 smartphones, allowing multiple users on each platform would restrict FL experimentation with just a handful of devices, defeating the purpose of CoLExT, which aims to support the investigation of FL over heterogeneous hardware. Due to this, access to CoLExT is currently limited to two users at a time for a set period (e.g., one week), with one user assigned to the Android portion, and the other user to the SBCs. Once the number of devices is significantly increased, a more scalable access mechanism will be introduced.

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

X. Conclusion

Reproducible and realistic experimentation is necessary for the future growth of distributed AI. Despite its research popularity, FL remains mostly confined to simulations, as the effort of deploying FL solutions over heterogeneous edge devices and collecting performance metrics remains insurmountable to most research groups. We presented CoLExT, an FL experimentation framework and testbed that allows an arbitrary FL algorithm to be run in a heterogeneous environment and assessed from various performance aspects. Our testbed already employs over 40 devices and collects a range of metrics, including inference accuracy, detailed energy usage, and CPU/GPU/memory utilization information. Nevertheless, we believe that by making CoLExT publicly available, we will support the organic growth of our testbed so that it addresses the up-to-date needs of FL researchers and practitioners.

Acknowledgments

This publication is based upon work supported by the King Abdullah University of Science and Technology (KAUST) Office of Research Administration (ORA) under Award No. ORA-CRG2021-4699, and partly supported by the Slovenian Research Agency (research projects J2-3047 and P2-0098). We are thankful to Abdullah Alamoudi, Suliman Alharbi, Rasheed Alhaddad for their helpful contributions to a first prototype of our system. Finally, we wish to thank the reviewers and our shepherd for their thoughtful comments and suggestions that helped improving our paper.

References

[1] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection," *IEEE Transactions on Knowledge & Data Engineering*, vol. 35, no. 4, 2023.

[2] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, H. Eichner, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, H. Qi, D. Ramage, R. Raskar, M. Raykova, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and Open Problems in Federated Learning," *Foundations and Trends® in Machine Learning*, vol. 14, 2021.

[3] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *AISTATS*, 2017.

[4] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "LEAF: A Benchmark for Federated Settings," arXiv:1812.01097, 2019.

[5] K. Hsieh, A. Phanishayee, O. Mutlu, and P. Gibbons, "The Non-IID Data Quagmire of Decentralized Machine Learning," in *ICML*, 2020.

[6] C. Yang, Q. Wang, M. Xu, Z. Chen, K. Bian, Y. Liu, and X. Liu, "Characterizing Impacts of Heterogeneity in Federated Learning upon Large-Scale Smartphone Data," in *WWW*, 2021.

[7] S. Wu, T. Li, Z. Charles, Y. Xiao, Z. Liu, Z. Xu, and V. Smith, "Motley: Benchmarking Heterogeneity and Personalization in Federated Learning," arXiv:2206.09262, 2022.

[8] A. M. Abdelmoniem, C.-Y. Ho, P. Papageorgiou, and M. Canini, "A Comprehensive Empirical Study of Heterogeneity in Federated Learning," *IEEE Internet of Things Journal*, 2023.

[9] G. A. Baumgart, J. Shin, A. Payani, M. Lee, and R. R. Kompella, "Not All Federated Learning Algorithms Are Created Equal: A Performance Evaluation Study," arXiv:2403.17287, 2024.

[10] J. Zhang, S. Zeng, M. Zhang, R. Wang, F. Wang, Y. Zhou, P. P. Liang, and L. Qu, "FLHetBench: Benchmarking Device and State Heterogeneity in Federated Learning," in *CVPR*, 2024.

[11] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification," arXiv:1909.06335, 2019.

[12] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks," in *MLSys*, 2020.

[13] Q. Li, B. He, and D. Song, "Model-Contrastive Federated Learning," in *CVPR*, 2021.

[14] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble Distillation for Robust Model Fusion in Federated Learning," in *NeurIPS*, 2020.

[15] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic Controlled Averaging for Federated Learning," in *ICML*, 2020.

[16] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization," in *NeurIPS*, 2020.

[17] F. Fourati, S. Kharrat, V. Aggarwal, M.-S. Alouini, and M. Canini, "FilFL: Client Filtering for Optimized Client Participation in Federated Learning," arXiv:2302.06599, 2023.

[18] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, "Flower: A Friendly Federated Learning Research Framework," arXiv:2007.14390, 2022.

[19] E. Diao, J. Ding, and V. Tarokh, "HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients," in *ICLR*, 2021.

[20] M. Aljahdali, A. M. Abdelmoniem, M. Canini, and S. Horváth, "Flashback: Understanding and Mitigating Forgetting in Federated Learning," arXiv:2402.05558, 2024.

[21] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive Federated Optimization," in *ICML*, 2021.

[22] F. Haddadpour and M. Mahdavi, "On the Convergence of Local Descent Methods in Federated Learning," arXiv:1910.14425, 2019.

[23] X. Zhang, M. Hong, S. Dhople, W. Yin, and Y. Liu, "FedPD: A Federated Learning Framework With Adaptivity to Non-IID Data," *IEEE Transactions on Signal Processing*, vol. 69, 2021.

[24] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards Federated Learning at Scale: System Design," in *MLSys*, 2019.

[25] A. M. Abdelmoniem, A. N. Sahu, M. Canini, and S. A. Fahmy, "REFL: Resource-Efficient Federated Learning," in *EuroSys*, 2023.

[26] T. Ortega and H. Jafarkhani, "Asynchronous Federated Learning with Bidirectional Quantized Communications and Buffered Aggregation," arXiv:2308.00263, 2023.

[27] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Efficient Federated Learning via Guided Participant Selection," in *OSDI*, 2021.

[28] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, 2003.

[29] B. White, L. Stoller, R. Ricci, S. Guruprasad, M. N. bold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," in *OSDI*, 2002.

[30] G. Judd and P. Steenkiste, "Using Emulation to Understand and Improve Wireless Networks and Applications," in *NSDI*, 2005.

[31] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols," in *WCNC*, 2005.

[32] A. Sakhnini, S. De Bast, M. Guenach, A. Bourdoux, H. Sahli, and S. Pollin, "Near-Field Coherent Radar Sensing Using a Massive MIMO Communication Testbed," *IEEE Transactions on Wireless Communications*, vol. 21, no. 8, 2022.

[33] J. Struye, B. Braem, S. Latré, and J. Marquez-Barja, "The CityLab Testbed – Large-scale Multi-technology Wireless Experimentation in a City Environment: Neural Network-based Interference Prediction in a Smart City," in *INFOCOM Workshops*, 2018.

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

[34] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The Design and Operation of CloudLab," in *USENIX ATC*, 2019.

[35] P. Foley, M. J. Sheller, B. Edwards, S. Pati, W. Riviera, M. Sharma, P. N. Moorthy, S. han Wang, J. Martin, P. Mirhaji, P. Shah, and S. Bakas, "OpenFL: the open federated learning library," *Physics in Medicine & Biology*, vol. 67, no. 21, 2022.

[36] H. R. Roth, Y. Cheng, Y. Wen, I. Yang, Z. Xu, Y.-T. Hsieh, K. Kersten, A. Harouni, C. Zhao, K. Lu, Z. Zhang, W. Li, A. Myronenko, D. Yang, S. Yang, N. Rieke, A. Quraini, C. Chen, D. Xu, N. Ma, P. Dogra, M. Flores, and A. Feng, "NVIDIA FLARE: Federated Learning from Simulation to Real-World," arXiv:2210.13291, 2022.

[37] Y. Xie, Z. Wang, D. Gao, D. Chen, L. Yao, W. Kuang, Y. Li, B. Ding, and J. Zhou, "FederatedScope: A Flexible Federated Learning Platform for Heterogeneity," *Proc. VLDB Endow.*, vol. 16, no. 5, 2023.

[38] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, X. Zhu, J. Wang, L. Shen, P. Zhao, Y. Kang, Y. Liu, R. Raskar, Q. Yang, M. Annavaram, and S. Avestimehr, "FedML: A Research Library and Benchmark for Federated Machine Learning," arXiv:2007.13518, 2020.

[39] Y. Liu, T. Fan, T. Chen, Q. Xu, and Q. Yang, "FATE: An Industrial Grade Platform for Collaborative Learning With Data Protection," *Journal of Machine Learning Research*, vol. 22, no. 226, 2021.

[40] A. Ziller, A. Trask, A. Lopardo, B. Szymkow, B. Wagner, E. Bluemke, J.-M. Nounahon, J. Passerat-Palmbach, K. Prakash, N. Rose, T. Ryffel, Z. N. Reza, and G. Kaissis, *PySyft: A Library for Easy Federated Learning*. Springer International Publishing, 2021.

[41] H. Daga, J. Shin, D. Garg, A. Gavrilovska, M. Lee, and R. R. Kompella, "Flame: Simplifying Topology Extension in Federated Learning," in *SoCC*, 2023.

[42] M. N. Galtier and C. Marini, "Substra: a framework for privacy-preserving, traceable and collaborative Machine Learning," arXiv:1910.11567, 2019.

[43] N. Kourtellis, K. Katevas, and D. Perino, "FLaaS: Federated Learning as a Service," in *DistributedML*, 2020.

[44] D. Chen, D. Gao, Y. Xie, X. Pan, Z. Li, Y. Li, B. Ding, and J. Zhou, "FS-REAL: Towards Real-World Cross-Device Federated Learning," in *KDD*, 2023.

[45] TensorFlow Team, "TensorFlow Federated," 2024. [Online]. Available: https://github.com/tensorflow/federated

[46] H. Ludwig, N. Baracaldo, G. Thomas, Y. Zhou, A. Anwar, S. Rajamoni, Y. Ong, J. Radhakrishnan, A. Verma, M. Sinn, M. Purcell, A. Rawat, T. Minh, N. Holohan, S. Chakraborty, S. Whitherspoon, D. Steuer, L. Wynter, H. Hassan, S. Laguna, M. Yurochkin, M. Agarwal, E. Chuba, and A. Abay, "IBM Federated Learning: an Enterprise Framework White Paper V0.1," arXiv:2007.10987, 2020.

[47] M. Zhang, F. Yu, Y. Yu, M. Zhang, A. Li, and X. Chen, "FedHC: A Scalable Federated Learning Framework for Heterogeneous and Resource-Constrained Clients," arXiv:2305.15668, 2023.

[48] F. Lai, Y. Dai, S. S. Singapuram, J. Liu, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "FedScale: Benchmarking Model and System Performance of Federated Learning at Scale," in *ICML*, 2022.

[49] M. H. Garcia, A. Manoel, D. M. Diaz, F. Mireshghallah, R. Sim, and D. Dimitriadis, "FLUTE: A Scalable, Extensible Framework for High-Performance Federated Learning Simulations," arXiv:2203.13789, 2022.

[50] L. Li, J. Wang, and C. Xu, "FLSim: An Extensible and Reusable Simulation Framework for Federated Learning," in *SIMUtools*, 2021.

[51] D. Zeng, S. Liang, X. Hu, H. Wang, and Z. Xu, "FedLab: A Flexible Federated Learning Framework," arXiv:2107.11621, 2022.

[52] J. H. Ro, A. T. Suresh, and K. Wu, "FedJAX: Federated learning simulation with JAX," arXiv:2108.02117, 2021.

[53] V. Mugunthan, A. Peraire-Bueno, and L. Kagal, "PrivacyFL: A Simulator for Privacy-Preserving and Secure Federated Learning," in *CIKM*, 2020.

[54] W. Zhuang, X. Gan, Y. Wen, and S. Zhang, "EasyFL: A Low-Code Federated Learning Platform for Dummies," *IEEE Internet of Things Journal*, vol. 9, no. 15, 2022.

[55] W. Zhao, X. Qiu, J. Fernandez-Marques, P. P. B. de Gusmão, and N. D. Lane, "Protea: Client Profiling within Federated Systems using Flower," in *FedEdge*, 2022.

[56] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shojafar, A. I. A. Ahmed, S. A. Madani, K. Saleem, and J. J. Rodrigues, "A survey on energy estimation and power modeling schemes for smartphone applications," *International Journal of Communication Systems*, vol. 30, no. 11, 2017.

[57] C. Guo, S. Ci, Y. Zhou, and Y. Yang, "A Survey of Energy Consumption Measurement in Embedded Systems," *IEEE Access*, vol. 9, 2021.

[58] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, and M. Jirstrand, "A Performance Evaluation of Federated Learning Algorithms," in *DIDL*, 2018.

[59] K. Selialia, Y. Chandio, and F. M. Anwar, "Federated Learning Biases in Heterogeneous Edge-Devices: A Case-Study," in *SenSys*, 2022.

[60] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated Learning on Non-IID Data Silos: An Experimental Study," in *ICDE*, 2022.

[61] J. O. du Terrail, S.-S. Ayed, E. Cyffers, F. Grimberg, C. He, R. Loeb, P. Mangold, T. Marchand, O. Marfoq, E. Mushtaq, B. Muzellec, C. Philippenko, S. Silva, M. Teleńczuk, S. Albarqouni, S. Avestimehr, A. Bellet, A. Dieuleveut, M. Jaggi, S. P. Karimireddy, M. Lorenzi, G. Neglia, M. Tommasi, and M. Andreux, "FLamby: Datasets and Benchmarks for Cross-Silo Federated Learning in Realistic Healthcare Settings," arXiv:2210.04620, 2023.

[62] Z. Zhang, X. Hu, J. Zhang, Y. Zhang, H. Wang, L. Qu, and Z. Xu, "FEDLEGAL: The First Real-World Federated Learning Benchmark for Legal NLP," in *ACL*, 2023.

[63] S. Hu, Y. Li, X. Liu, Q. Li, Z. Wu, and B. He, "The OARF Benchmark Suite: Characterization and Implications for Federated Learning Systems," *ACM Trans. Intell. Syst. Technol.*, vol. 13, no. 4, 2022.

[64] C. Song, F. Granqvist, and K. Talwar, "FLAIR: Federated Learning Annotated Image Repository," in *NeurIPS*, 2022.

[65] T.-M. H. Hsu, H. Qi, and M. Brown, "Federated Visual Classification with Real-World Data Distribution," arXiv:2003.08082, 2020.

[66] R. Sun, Y. Li, T. Shah, R. W. H. Sham, T. Szydlo, B. Qian, D. Thakker, and R. Ranjan, "FedMSA: A Model Selection and Adaptation System for Federated Learning," *Sensors*, vol. 22, no. 19, 2022.

[67] J. Shin, Y. Li, Y. Liu, and S.-J. Lee, "FedBalancer: Data and Pace Control for Efficient Federated Learning on Heterogeneous Clients," in *MobiSys*, 2022.

[68] X. Ouyang, Z. Xie, J. Zhou, J. Huang, and G. Xing, "ClusterFL: A Similarity-Aware Federated Learning System for Human Activity Recognition," in *MobiSys*, 2021.

[69] J. Mills, J. Hu, and G. Min, "Communication-Efficient Federated Learning for Wireless Edge Intelligence in IoT," *IEEE Internet of Things Journal*, vol. 7, no. 7, 2020.

[70] H. Woisetschläger, A. Isenko, R. Mayer, and H.-A. Jacobsen, "FLEdge: Benchmarking Federated Machine Learning Applications in Edge Computing Systems," arXiv:2306.05172, 2023.

[71] K.-S. Wong, M. Nguyen-Duc, K. Le-Huy, L. Ho-Tuan, C. Do-Danh, and D. Le-Phuoc, "An Empirical Study of Federated Learning on IoT-Edge Devices: Resource Allocation and Heterogeneity," arXiv:2305.19831, 2023.

[72] (2024) gRPC: A high-performance, open source universal RPC framework. [Online]. Available: https://grpc.io

[73] (2024) Harbor: Cloud Native Registry. [Online]. Available: https://goharbor.io

[74] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[75] Y. LeCun, C. Cortes, and C. J. Burges. (1998) The MNIST database of handwritten digits. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[76] (2024) HVPM: High Voltage Power Monitor. [Online]. Available: http://msoon.github.io/powermonitor/HVPM.html

[77] K. Tsakalozos. (2024) MicroK8s Issue #2. [Online]. Available: https://github.com/canonical/microk8s/issues/2

[78] (2024) AI Benchmark Ranking. [Online]. Available: https://ai-benchmark.com/ranking.html

[79] (2024) NVIDIA Jetson for Next-Generation Robotics. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/

[80] J. Riek. (2024) Ubuntu Rockship. [Online]. Available: https://github.com/Joshua-Riek/ubuntu-rockchip

[81] N. Hyeon-Woo, M. Ye-Bin, and T.-H. Oh, "FedPara: Low-Rank Hadamard Product for Communication-Efficient Federated Learning," in *ICLR*, 2022.

[82] S. Horvath, S. Laskaridis, M. Almeida, I. Leontiadis, S. Venieris, and N. Lane, "FjORD: Fair and Accurate Federated Learning under heterogeneous targets with Ordered Dropout," in *NeurIPS*, 2021.

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.

[83] F. L. GmbH. (2024) Flower Summer: A Federated Learning Initiative. [Online]. Available: https://flower.ai/summer/

[84] S. Eustace. (2024) Poetry: Python Dependency Management and Packaging Made Easy. [Online]. Available: https://python-poetry.org/

[85] D. Wu, R. Ullah, P. Harvey, P. Kilpatrick, I. Spence, and B. Varghese, "FedAdapt: Adaptive Offloading for IoT Devices in Federated Learning," *IEEE Internet of Things Journal*, vol. 9, no. 21, 2022.

[86] M. Asad, A. Moustafa, and T. Ito, "FedOpt: Towards Communication Efficiency and Privacy Preservation in Federated Learning," *Applied Sciences*, vol. 10, no. 8, 2020.

[87] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: Extending MNIST to handwritten letters," in *IJCNN*, 2017.

[88] L. Foundation. (2024) ONNX: Open Neural Network Exchange. [Online]. Available: https://onnx.ai

Authorized licensed use limited to: UNIVERSITY OF LJUBLJANA. Downloaded on January 08,2025 at 20:32:53 UTC from IEEE Xplore. Restrictions apply.