SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

FLOps: Practical Federated Learning via Automated Orchestration (on the Edge)

Alexander Malyuk

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

FLOps: Practical Federated Learning via Automated Orchestration (on the Edge)

TODO

Author:	Alexander Malyuk
Supervisor:	Prof. Dr-Ing. Jörg Ott
Advisor:	Dr. Nitinder Mohan, Giovanni Bartolomeo
Submission Date:	15.09.2024

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15.09.2024 Alexander Malyuk

Acknowledgments

Abstract

Kurzfassung

Contents

1 Introduction

The number of smart devices has been rapidly growing in the last several years. Improvements in connectivity (Cloud Computing & Internet of Things—IoT), connection speeds (5G), and computing power enable this increasing fleet of edge/mobile devices to generate enormous amounts of data (BigData). Combined with the expansion of AI/ML, this data is a driving factor for current successful workflows and future advancements. This complementing union of technologies plays a key role in elevating various domains, from agriculture and healthcare to education and the security sector, to Industry 4.0 and beyond. Diverse and complex challenges arise from this swiftly evolving landscape. [3]

1.1 Problem Statement

With great access to data comes great responsibility that can be easily exploited. Many of the aforementioned machines are personal user devices or belong to companies and organizations that handle customer or internal resources. These devices store and handle sensitive private data.

In classical (large-scale) Machine Learning, data gets sent from client devices to a centralized server, which usually resides in the cloud. The collected data is used on the server to train ML models or perform inference serving. This approach provides direct access to this sensitive data and the power to trace back its origin, creating a breach of privacy. [11]

Governments and organizations have established laws and regulations to prohibit potential abuse of sensitive data. Examples include the European Parliament regulation to protect personal data [15] or the California Consumer Privacy Act (CCPA) [10]. These measures aim to support cooperation between organizations and nations while protecting trade secrets. However, some laws and regulations prohibit sharing or moving data to other countries or even off-premises. [11]

Ignoring and no longer using this large amount of data would heavily limit current workflows and further developments for many data-dependent and data-hungry technologies. In 2017, a team of Google researchers introduced Federated Learning (FL) as one possible solution to utilize sensitive data while keeping it private. Instead of collecting the data on a server and training ML models centralized, in FL, the model

training occurs directly on the client devices. Afterward, the individually trained models get sent to the server, which combines the collected models into a single shared one. This so-called global model can then be distributed to the clients again for further training cycles. Therefore, FL enables training a shared model on sensitive data while keeping that data secure on the local client devices. [12]

While many researchers are actively engaged in the field of FL, the majority of them are focused on enhancing existing FL components, strategies, and algorithms or devising better ways of doing FL. However, there is a noticeable scarcity of work that concentrates on the crucial aspects of the initial setup, deployment, and usability of FL. We delve into this issue in greater detail in the dedicated background section (2.1.3).

Because FL is a relatively modern technique, it lacks a sophisticated production-grade ecosystem that includes frameworks and libraries that improve ease of use by automating its setup and execution. As a result, contributing to the field of FL or reproducing findings is a task ranging from non-trivial to improbable due to the lack of documented steps regarding setup, deployment, management, and execution. Instead of using a shared set of tools for bootstrapping to make progress on novel work more efficiently, one needs to set up and manage FL from the ground up. Note that a small set of emerging libraries and frameworks exists for FL. Instead of orchestrating FL on real distributed devices, they focus on executing FL algorithms and processes, often via virtual simulations. Not to mention utilizing more advanced techniques to increase productivity that other domains have already been using for several years, such as modern DevOps practices like continuous deployment. We review existing FL tools in detail in the dedicated section (2.1.5).

1.2 Motivation

Building or contributing to a novel FL framework or library focusing on the previously mentioned challenges could soften or entirely alleviate those problems. We are talking about a tool that sees Docker [6] and Kubernetes [9] as role models and strives to be comparable to them but for the discipline of FL. It should specialize in the setup, deployment, component management, and automation, in short, FL orchestration. Allowing researchers, developers, and end-users to set up, perform, reproduce, and experiment with FL in a more accessible way.

The goal of this tool should be to automate and simplify complex tasks, reducing the required level of expertise in various domains, ranging from ML/FL, dependency management, containerization technologies, and automation to orchestration. Such a tool would empower less experienced individuals to participate and contribute to the field of FL. As a result, FL could be adapted and used by more people in more areas.

This tool would streamline and accelerate existing workflows and future progress by utilizing reliable automation to avoid error-prone manual tasks. With its potential to optimize, standardize and unify processes, our envisioned tool could become a significant part of the emerging FL ecosystem, contributing to the development and progress of the entire field.

1.3 Objectives

The motivation allows us to distill the following key objectives for such a tool.

Improve Accessibility
Making FL more accessible by abstracting away and automating complexities, enables more individuals to engage with it. Expanding FL to more areas will increase its usage and user base, raising general interest and relevance for its field, which should aid its development.

Utilize Automation
Automating tedious, error-prone, and repetitive manual tasks necessary to perform FL will free up time and resources for more critical work, leading to further advancements.

Prioritize Tangible Applicability
As we discuss in (2.1.3), FL struggles with a gap between research/virtual-simulation and practical application in real production environments. This tool should focus on being usable in real physical conditions on distributed devices. It should be feasible to incorporate this tool into existing workflows.

Embrace Plasticity
Because FL is such a young field, it faces constant change. Naturally, our tool should welcome change in the form of extendability and adaptability. This tool should be flexible and applicable to a myriad of use cases and scenarios. It should be easy to modify to accommodate evolving needs. Likewise, this tool should profit from existing technologies to offer a higher level of quality than creating everything from square one.

1.4 Contribution

We introduce FLOps to fulfill the objectives above. It enables individuals to use, develop, and evaluate tangible FL. FLOps enriches FL with modern best practices from automation, DevOps/MLOps, and orchestration. FLOps improves accessibility by enabling users without experience in FL, MLOps, or orchestration to do FL and still benefit from these technologies via automated orchestration.

To do FL, users simply provide a link to their ML git repository. Note that this code needs to satisfy some simple structural prerequisites. This repository code gets automatically augmented by FLOps to support FL. FLOps creates a containerized image with all necessary dependencies to do FL training. These images are automatically built and adhere to best practices, ensuring they are as fast and lightweight as possible. FLOps can build these images for multiple different target platforms. Thus, FL components can run on ARM edge-devices like Raspberry Pis or Nvidia Jetsons. FLOps enables FL on all devices that support containerization technologies like Docker [6] or containerd [5]. This approach eliminates the need for tedious device setup and the struggle to configure heterogeneous dependencies to match the necessary requirements for training, thereby streamlining the process and saving time.

FLOps automatically performs FL training based on the user-requested configuration. Users can, for example, specify resource requirements, the number of training rounds, the FL algorithm, the minimum number of participating client devices, and more. During runtime, users can observe this training process via a sophisticated GUI, which allows users to monitor, compare, store, export, share, and organize training runs, metrics, and trained models. FLOps can automatically build inference servers based on the trained model. This inference server can be pulled as a regular image. FLOps can also directly deploy this trained-model image as an inference server.

A multitude of diverse technologies and areas are necessary for FLOps to provide its services. Instead of reimplementing these complex features in a subpar fashion from scratch, we benefit from combining and extending existing solutions and technologies in unique and novel ways. This includes the use of Anaconda [1] and Buildah [4] to manage dependencies and build images. We utilize a pioneering FL framework called Flower [8] to execute the FL training loop. The mentioned runtime observability features are available via a mature MLOps tool called MLflow [13]. Because FL pushes model training to client devices, especially edge devices, we decided to use an orchestrator native to the edge environment. With the help of Oakestra [2], FLOps can deploy and orchestrate its components. FLOps is implemented as an separate addon for Oakestra. Because their interaction is based on general API endpoints and SLAs, FLOps can be modified to support other Orchestrators.

It is noteworthy that these different tools do not natively support each other. FLOps

combines them in unprecedented ways to achieve its goals. As an example, FLOps supports hierarchical FL (HFL), which is not directly supported or offered by Flower. To the best of our knowledge, FLOps is the first work that combines Flower with MLflow and allows HFL, as well as automatically converts ML code into FL enabled containerized images.

As far as we know, the term FLOps, besides being a measurement unit for computer performance (Floating point operations per second), has not been used or applied to FL unlike MLOps has been used to describe DevOps techniques for ML. The goal of this work is to showcase the benefits of utilizing the mentioned techniques and open the doors for future developments for FL.

Besides the end-user perspective, FLOps is intended to be a foundational piece of software that can be easily modified and extended for developers and researchers. We put a lot of effort into writing high quality code, using state of the art libraries and frameworks. FLOps includes many development-friendly features. We enforce proper styling and typing via formatters and linters, including CI. Ready-made extendable multi-platform images and services automate development and evaluation workflows. These images, as well as the entire code, are made available on GitHub [7]. We also added base images with optional development flags to speed up the build and execution times of FLOps so that developers can verify and check their changes more rapidly.

On top of that we also implemented a new CLI tool for Oakestra and FLOps from the grounds up [14]. It is used to interact with Oakestra's and FLOps APIs. Besides that this configurable CLI tool also is capable of visualizing current processes in a human friendly way in real time as well as trigger evaluation runs and other automated tasks like installing necessary dependencies.

1.5 Thesis Structure

TODO

2 Background

As mentioned in the contributions section, FLOps combines and uses a large set of technologies from different disciplines. To properly understand FLOps as a whole and why it combines these techniques, it is necessary to analyze them individually. This enables us to form a common understanding, including critical background knowledge of their benefits and downsides. Only afterward does it make sense to discuss how FLOps merges them to create something new.

This background chapter provides a general overview for each sector and discuss aspects that are necessary for FLOps in greater detail. We start with exploring the field of federated learning. FL is the core task at hand that FLOps aims to optimize. A thorough understanding of this discipline is required to figure out where it has shortcomings. To improve upon these weaknesses, we study the established set of best practices from DevOps and MLOps. Techniques like automation and CI/CD require infrastructure and resources. Orchestration allows us to provision, manage, and deploy such infrastructure and resources. We review the field of orchestration technologies and provide a short overview of Oakestra [2] as the chosen platform. In the final background section, we take a look at and compare a couple of existing pieces of work that resemble FLOps.

2.1 Federated Learning

TODO

2.1.1 Basics

2.1.2 Architectures

2.1.3 Research

2.1.4 Industry

2.1.5 Frameworks & Libraries

2.1.6 Flower

2.2 Machine Learning Operations

2.2.1 DevOps

2.2.2 MLOps

2.2.3 MLflow

2.3 Orchestration

2.3.1 Fundamentals

2.3.2 ML Containerization & Orchestration

2.3.3 Oakestra

2.4 Related Work

3 Requirements Analysis

4 System Design

5 Object Design

6 Evaluation

6.1 Rationale

6.1.1 Chosen Experiments

6.2 Experimental Setup

6.2.1 Monolith

6.2.2 Multi-Cluster

6.2.3 Evaluation Procedure

6.3 Results

6.3.1 Basics

6.3.2 Image Builder

6.3.3 Different ML Frameworks/Libraries & Datasets

6.3.4 Multi-cluster & HFL

7 Conclusion

7.1 Limitations & Future Work

7.1.1 Federated Learning via FLOps

7.1.2 Complementary Components & Integrations

8 DELME tum example

8.1 Section

Acronyms must be added in `main.tex` and are referenced using macros. The first occurrence is automatically replaced with the long version of the acronym, while all subsequent usages use the abbreviation.

E.g. `\ac{TUM}, \ac{TUM}` ⇒ Technical University of Munich (TUM), TUM

For more details, see the documentation of the `acronym` package[1].

8.1.1 Subsection

See Table 8.1, Figure 8.1, Figure 8.2, Figure 8.3.

Table 8.1: An example for a simple table.

A	B	C	D
1	2	1	2
2	3	2	3

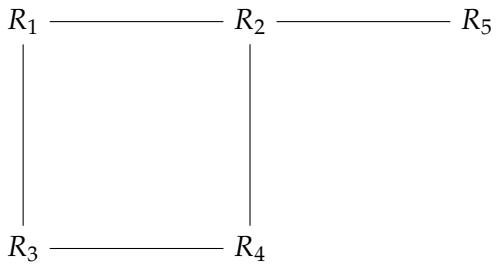

Figure 8.1: An example for a simple drawing.

[1] `https://ctan.org/pkg/acronym`

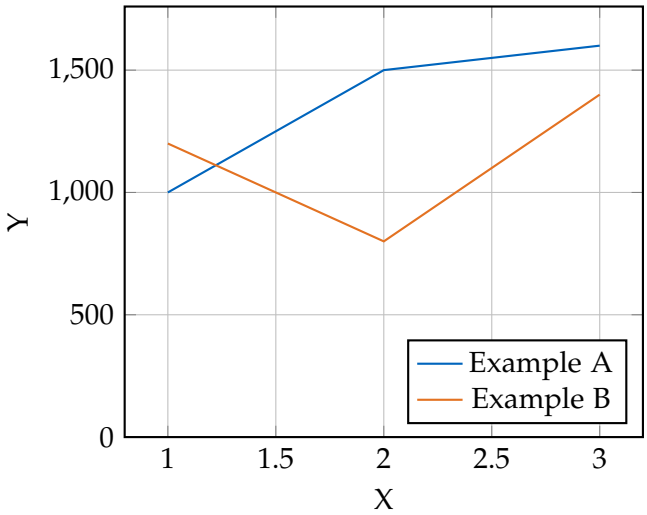

Figure 8.2: An example for a simple plot.

```
SELECT * FROM tbl WHERE tbl.str = "str"
```

Figure 8.3: An example for a source code listing.

Abbreviations

TUM Technical University of Munich

List of Figures

List of Tables

Bibliography

[1] *Anaconda Documentation*. Accessed: 2024-08-12. URL: https://docs.anaconda.com/.

[2] G. Bartolomeo, M. Yosofie, S. Bäurle, O. Haluszczynski, N. Mohan, and J. Ott. "Oakestra: A Lightweight Hierarchical Orchestration Framework for Edge Computing." In: *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, July 2023, pp. 215–231. ISBN: 978-1-939133-35-9.

[3] A. Bourechak, O. Zedadra, M. N. Kouahla, A. Guerrieri, H. Seridi, and G. Fortino. "At the Confluence of Artificial Intelligence and Edge Computing in IoT-Based Applications: A Review and New Perspectives." In: *Sensors* 23.3 (2023). ISSN: 1424-8220. DOI: 10.3390/s23031639.

[4] *Buildah Homepage*. Accessed: 2024-08-12. URL: https://buildah.io/.

[5] *containerd Documentation*. Accessed: 2024-08-12. URL: https://containerd.io/docs/.

[6] *Docker Documentation*. Accessed: 2024-08-12. URL: https://docs.docker.com/.

[7] *FLOps Code Repository*. Accessed: 2024-08-12. URL: https://github.com/oakestra/addon-FLOps.

[8] *Flower Documentation*. Accessed: 2024-08-12. URL: https://flower.ai/docs/.

[9] *Kubernetes Documentation*. Accessed: 2024-08-12. URL: https://kubernetes.io/docs/home/.

[10] C. Legislature. *California Consumer Privacy Act (CCPA)*. Online; accessed August 11, 2024. 2018.

[11] H. Ludwig and N. Baracaldo, eds. *Federated Learning - A Comprehensive Overview of Methods and Applications*. Springer, 2022. ISBN: 978-3-030-96896-0. DOI: 10.1007/978-3-030-96896-0.

[12] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas. "Communication-Efficient Learning of Deep Networks from Decentralized Data." In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by A. Singh and J. Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1273–1282.

[13] *MLflow Documentation*. Accessed: 2024-08-12. URL: https://www.mlflow.org/docs/latest/index.html#.

[14] *Oakestra & FLOps CLI Code Repository*. Accessed: 2024-08-12. URL: https://github.com/oakestra/oakestra-cli.

[15] T. E. Parliament and Council. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. Online; accessed August 11, 2024. 2016.