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

During the last several years the number of smart devices has been rapidly growing. The impressive improvements in the domains of connectivity, connection speeds, computing power, enable this ever-increasing swarm of machines (IoT, edge-devices, mobiles, etc.) to generate a huge amount of data. Combined with the exponential growth of AI/ML this data is a driving factor for further improvements. Diverse and complex challenges arise from this rapidly evolving landscape.

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 that store and handle sensitive private data. In Classical Machine Learning data gets send from client devices to a server where for training the ML model or performing inference serving. This enables direct access to this sensitive data as well as the power to trace it back. Thus creating a breach of privacy.

Governments and organisations have put laws and regulations in place to prohibit such potential abuse.

Federated Learning (FL) is one possible solution to still be able to utilize sensitive data. Instead of collecting the data on a server and training ML models there the training occurs directly on the client devices. Afterwards the individually trained models get send to the server who combines the collected models into a single global one. This global model can be then distributed to the clients again for further training cycles. Therefore enabling a way to train a shared global model on sensitive data while keeping the sensitive data secure on the local client devices.

Because FL is a rather modern technique it lacks in regards of being easy to set up and utilize. It lacks sophisticated tooling to increase ease of access and use. There is a lot of research being done in the realm of FL. The majority of researchers in this field focus on improving existing FL components, strategies, and algorithms or inventing better alternatives. Is is very rare to see work that focuses on the actual initial setup, deployment, or improvment of usability of FL. As a result contributing in the field of FL or simply reproducing findings is a task that ranges from non-trivial to completely undocumented blackbox, thus not reproducable. Instead of using common tooling to catalyze one's work to be able to quicker get started with doing new novel work. One needs to reinvent the wheel of how to setup and manage one's FL. I.e. There is a big lack of unity and automation in the realm of FL.

So FL is lacking the bare-bone tools to easily work on it - not even to mention more advanced techniques to increase productivity and effectiveness that other domains are already using for ages.

1.2 Motivation

Let's imagine the mentioned problems could be resolved. What difference would it make? If we invision a universally trusted and used standard tool or set of tools for doing FL, similar to what Docker or Kubernetes have become. This tool would allow researchers, developers, and end-users to easily set up, perform, reproduce, experiment everything FL related. This would not only reduce the necessary level of expert knowledge in this field and related ones but also accelerate existing workflows. FL could be adapted and used by more people in more areas. By the power of automating tedies and error-prone tasks the current and future progress of FL connected work could be accelerated.

1.3 Contribution

In this thesis we introduce FLOps, a novel open-source foundational work and proof-of-concept that enables real (not simulated) FL to be used, developed and evaluated. FLOps enriches FL with best SOTA practices from the realms of automation, DevOps/MLOps, and orchestration. FLOps is intended to be easy to use for people without previous expert knowlege in FL, MLOps or orchestration. Users can simply provide their ML code in form of a git repository (e.g. in GitHub) as long as this code satisfies some simple structural prerequisites. This repo code gets automatically augmented by FLOps to support FL. FLOps creates a containerized Image with all necessary dependencies to do FL training. These images get automatically build and follow best practices to be as fast and light-weight as possible. These images can be build for multiple different target platforms. Thus enabling FL component images to run e.g. on arm devices like PIs or Nvidia Jetsons. With the help of Oakestra deployes and orchestrates these FL components. FLOps automatically performs FL training. This training process can be observed during runtime via the use of SOTA MLOps tools like MLflow, which offers a sophisticated GUI where users can observe, compare, store, export, share, and organize training runs, metrics, and trained models.

FLOps uses Flower as a FL framework which so far does not support Hierarchical FL. As far as we know, FLOps is the first work that combines Flower and MLflow and allows HFL.

As far as we know, the term FLOps (besides the unit of how powerful something can compute things) has not been used, thus this work is called FLOps but should also open the door into future exiting developments for dedicated ML/Dev-Ops practices specifically for FL.

FLOps also allows to automatically build an inference server/service based on the trained model. This image can then be pulled like any other image by the user and used for arbitrary purposes. If the user requests it FLOps can also directly deploy this trained-model image as an inference service in Oakestra.

Besides the end-user perspective FLOps is intended to be a foundational piece of software that should be easily modifiable and extendable for developers and researchers. We put a lot of effort into writing high quality code, using SOTA libraries and frameworks. We also added many developent friendly features into FLOps. E.g. enforced proper styling and typing via formatters and linters, including CI. We createad ready-made static images and services that can be used and extended to automate development and evaluation workflows. E.g. we have a Mock-Data-Provider image/service that can act as a data provider to populate the data used for FL training. Or the Inference Service that can be deployed with the base case FLOps project to verify that the trained model works as expected and is able to properly do inference serving. All these images support arm and amd architectures.

We also add base-images with optinal development flags to speed up 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 from scratch that is used to interact with Oakestra's and FLOps API. Besides that this configurable CLI tool also is capable of visualizing current proceses in a human friendly way in real time as well as trigger evaluation runs and other automated tasks like installing necessary dependencies onto the local machine.

1.4 Thesis Structure

2 Background

2.1 Federated Learning

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

Citation test [Lam94].

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

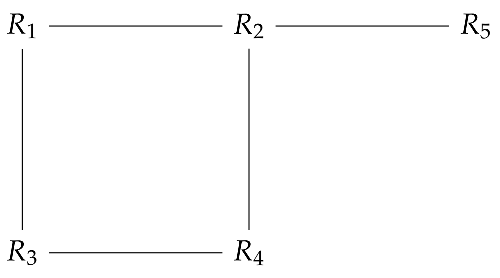
For more details, see the documentation of the `acronym` package[1].

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



$$\begin{tikzcd} R_1 \arrow[r, dash] \arrow[d, dash] & R_2 \arrow[r, dash] \arrow[d, dash] & R_5 \\ R_3 \arrow[r, dash] & R_4 & \end{tikzcd}$$

Figure 8.1: An example for a simple drawing.

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

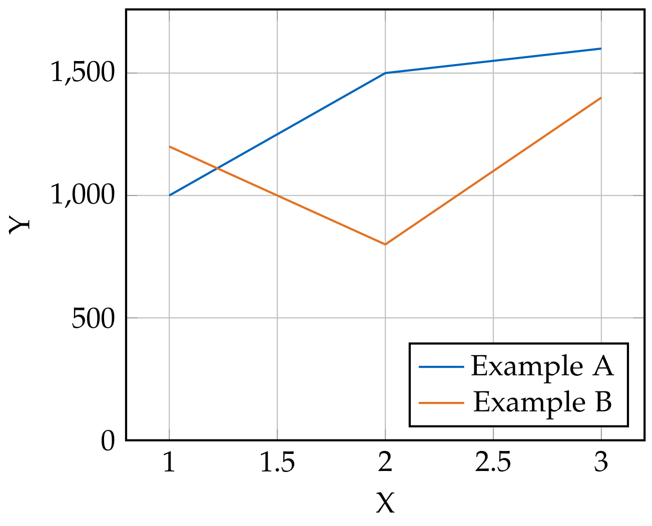


Figure 8.2: An example for a simple plot.

```
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

[Lam94] L. Lamport. *LaTeX : A Documentation Preparation System User's Guide and Reference Manual*. Addison-Wesley Professional, 1994.