MLOps Final Project Specification Report

HURLIN Pierre
MALAGAD Glenn Pier
RAVICHANDRAN MEHALA Sabarish
SCLANO Francesco
VAN ECCELPOEL Théo

April 2025

Master in Data Sciences and Business Analytics

MLOps

1 Objective

The objective of this project is to develop a modular and lightweight MLOps platform focused on the infrastructure and operations side of machine learning workflows. The platform supports secure data ingestion from a remote Supabase database with role-based access control, automated file handling, caching of expensive computations, and experiment tracking through MLflow. The goal is to provide a reusable and scalable toolset that simplifies data access, enhances reproducibility, and enables efficient experimentation management—without focusing on model performance or algorithm development.

2 Core Functional Components

2.1 Secure Data Ingestion with Supabase

The modular ingestion logic connects Python to a Supabase (PostgreSQL) backend. The SQL-based data retrieval and ingestion are managed using the Supabase API. Files are uploaded to Supabase through authenticated sessions. Authentication and authorization are set up with the help of `admin_setup.py` and `roles_databases.py` files. These implement Role-Based Access Control (RBAC) using Supabase roles.

Users are assigned roles such as `unauthorized`, `read_access`, or `read_write_access`. By default, users can register directly using a registration option, creating an account with their email and password. New users are initially unauthorized to query the database. Only an admin can promote users to query-capable roles using admin credentials.

This workflow mimics real-world data access governance and can be extended to include automated role request notifications. All tokens and sensitive credentials are secured using environment variable files, which are excluded from version control as a rule of thumb.

2.2 File Based Caching

Here the main focus is to improve the performance boost. This is achieved by wrapping the expensive functions like data ingestion, preprocessing to store outputs on disk automatically using the input hashing. This prevents the recomputation when the same inputs are used. This is ideally preferred for iterative workflows.

2.3 Experiment Tracking with MLflow

The entire lifecycle of MLflow is managed by the `mlflow_integration.py` file. It logs all the parameters, tags, artifacts, and metadata during pipeline execution. This eliminates the need for manual logging and enables tracking of performance and configuration changes over time. An optional MLflow UI can be used to visualize and compare experiment runs.

2.4 Modular Design with CLI Integration

The scripts are made CLI-compatible, enabling them to be modular and reproducible using the command line execution. Also the users can run ingestion, logging, caching or registry tasks independently or as in pipeline.

3 Implementation structure

- **Branch 1 (dev_theo)**: Core structure, CLI layout, and initial setup
- **Branch 2 (dev_fra)**: Full ingestion logic, Supabase access, file and table modules, Role-Based Access Control (RBAC)
- **Branch 3 (dev_pier)**: MLflow experiment logging and caching via decorators

4 Technologies Stack

Component	**Tool / Library**
Language	Python
Database	Supabase (*PostgreSQL*)
API Access	Supabase Python SDK
Auth Management	Env vars + Supabase roles
Experiment Logs	MLflow
Caching Layer	Joblib / Pickle via custom wrapper
CLI Interface	Python (`argparse`)

Table 1: Technology Stack Overview

5 Usage and Setup

1. Create a `.env` file with the following variables:
   - `SUPABASE_KEY`
   - `SUPABASE_URL`
   - Any custom roles or tokens required by the Supabase instance
2. Run `admin_setup.py` once to register roles and initialize permissions.
3. Ingest or upload data using either `databases.py` or `files.py` depending on the data source.
4. Use decorators from `simple_cache.py` to cache long-running functions and improve performance.