MLOps Term Project: Pima Diabetes Prediction System

Development and Evaluation of a Machine Learning Lifecycle Management System using MLflow

Author: Atena Jafari Parsa

Course: AIN3009- Delivering AI Applications with MLOps

Table of Contents

Abstract

This project aims to develop a complete machine learning lifecycle management system using **MLflow**, an open-source platform. Focusing on a domain-specific dataset (AI in Healthcare), I will implement a predictive ML model (classification task for diabetes) and manage its end-to-end lifecycle, including training, hyperparameter tuning, deployment, performance monitoring, and model versioning. The system will offer transparency, reproducibility, and modularity aiming for scalable and maintainable machine learning applications.

Introduction

The lifecycle of a machine learning model involves multiple stages: data preparation, model training, parameter tuning, deployment, monitoring, and version control. Managing these stages manually is often prone to errors and inefficient during production.

MLflow is a powerful open-source platform that handles this process by offering tools for experiment tracking, model packaging, serving, and registry.

In this project, I will leverage MLflow to implement a robust and reproducible ML pipeline tailored to a specific domain (AI in healthcare). This report documents the setup, execution, and evaluation of my ML system, demonstrating how MLflow can be integrated into real-world machine learning workflows.

Project Objectives

1. **Experiment Tracking:** The developed system will Implement and demonstrate how MLflow can be used to track different experiments, including logging parameters, metrics, and outputs.
2. **Model Training and Tuning**: I'll develop ML models and use MLflow to log different training sessions
   with varying parameters and hyperparameter tuning processes.
3. **Model Deployment:** The trained model will be packaged using MLflow's model packaging tools and deployed as a service for real-time or batch predictions.
4. **Performance Monitoring:** The deployed model's performance will be monitored overtime, utilizing MLflow to track drifts in model metrics.
5. **Model Registry:** I will use MLflow's Model Registry to manage model versions and lifecycle including
   stage transitions like staging and production.

Dataset & Domain Selection

The selected domain is AI in Healthcare as diverse AI and ML-related datasets are available in that domain. I found The Pima Indians Diabetes Database from kaggle. It is well-structureed, interpretable and suitable for applying **experiment tracking**, **model training**, **hyperparameter tuning**, and **deployment**. The target variable is "Outcome" (0 = No diabetes, 1 = Diabetes). The features include Glucose, BMI, Age, Blood Pressure, Insulin, etc.

There are ~768 rows and 9 columns in total which is good for fast experimentation.

Project Structure

```
TermProject_MLOps/
│
├── data/
│   └── diabetes.csv                # The dataset used for training and evaluation
│
├── notebooks/
│   ├── eda.ipynb                   # Exploratory Data Analysis notebook
│   └── mlflow_pipeline.ipynb       # Full ML pipeline: training, tracking, registering
│
├── scripts/
│   ├── deploy_model.py             # Flask API to deploy the best model for
predictions
│   ├── monitor_model.py            # Script to simulate API requests and log
predictions
│   ├── register_best_model.py      # Script to register the best model to MLflow
registry
│   ├── train_model.py              # Train a single baseline model (Logistic
Regression)
│   ├── train_multiple_models.py    # Train and compare multiple ML models (RF,
SVM, etc.)
│   └── tune_hyperparams.py         # Tune Random Forest hyperparameters and
log results
│
├── mlruns/                         # MLflow artifacts and experiment tracking directory
```

(auto-generated)
│
├──
README.md # Project overview, setup instructions, usage guide
├── requirements.txt # Python dependencies needed to run the project

Data & eda.ipynb Notebook

“diabetes.csv” is the raw data and **eda.ipynb** is meant to explore the general structure of the dataset and detect missing values and anomalies as well as understanding the features’ relationships with the target variable.

After importing the necessary libraries, the dataset is loaded and the first 5 rows are shown.

```
# Load dataset
df = pd.read_csv("../data/diabetes.csv")
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

As shown above, the first feature is the number of pregnancies which demonstrates that the data belongs to Indian Women. In fact, it is written on kaggle that: “...all patients here are females at least 21 years old of Pima Indian heritage.” More details on the exploratory data analysis is available in eda.ipynb.

ML Model Development & Experiment Tracking with MLflow

Training the Model

In “train_model.py”, after importing the necessary libraries (also written in requirements.txt), the dataset has to be loaded. Then, just to be more careful, 0s

are replaced with NaN for specific columns and then filled with the median value of that column.

- **Split**

To prepare the data for training the machine learning model, the “Outcome” column has to be dropped and saved to “X” which will hold all the input features. The “Outcome” feature (whether the patient has diabetes or not) will be saved to “y” as it would be what the model will try to predict (the dependent variable). Then, the actual splitting process takes place here:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

This part splits the data into two parts:

- X_train, y_train: for **training** the model
- X_test, y_test: for **testing** the model afterward

test_size=0.2 means that 20% of the data is for testing and 80% is for training.

random_state=42 ensures that the split is the same every time you run it (reproducibility).

- **MLflow**

After starting MLflow, and starting the MLflow experiment “Pima Indians Diabetes Prediction”, a folder with the same name as the experiment will be created on MLflow where all the training results will be saved.

```
with mlflow.start_run():
```

By starting a new “run”, everything inside the block below will be tracked: parameters, scores, models, etc.

```
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train, y_train)
```

This part of the code creates a Logistic Regression model, which is great for binary classifications and yes/no predictions like diabetes. “max_iter=1000” tells the model to take more time to try and find the best answer if needed.

“.fit()” is where the model learns from the training data.

```
mlflow.log_metric("accuracy", acc)
```

This part saves a score/result of how accurate and well the model did. It is a metric or something the model achieved.

```
mlflow.sklearn.log_model(model, "model")
```

This part saves the actual trained model into MLflow and anyone else will be able to download, reuse or deploy it later.

```
print(f"Model accuracy: {acc}")
```

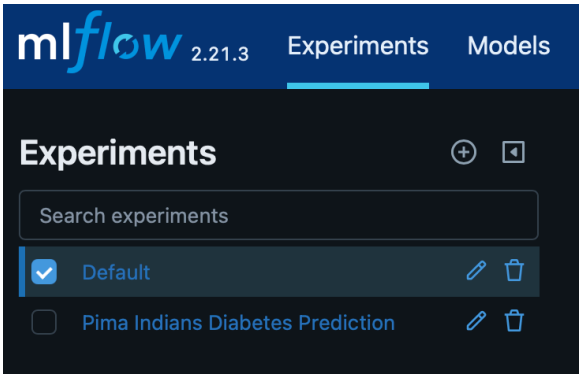
This line prints the accuracy in the terminal.

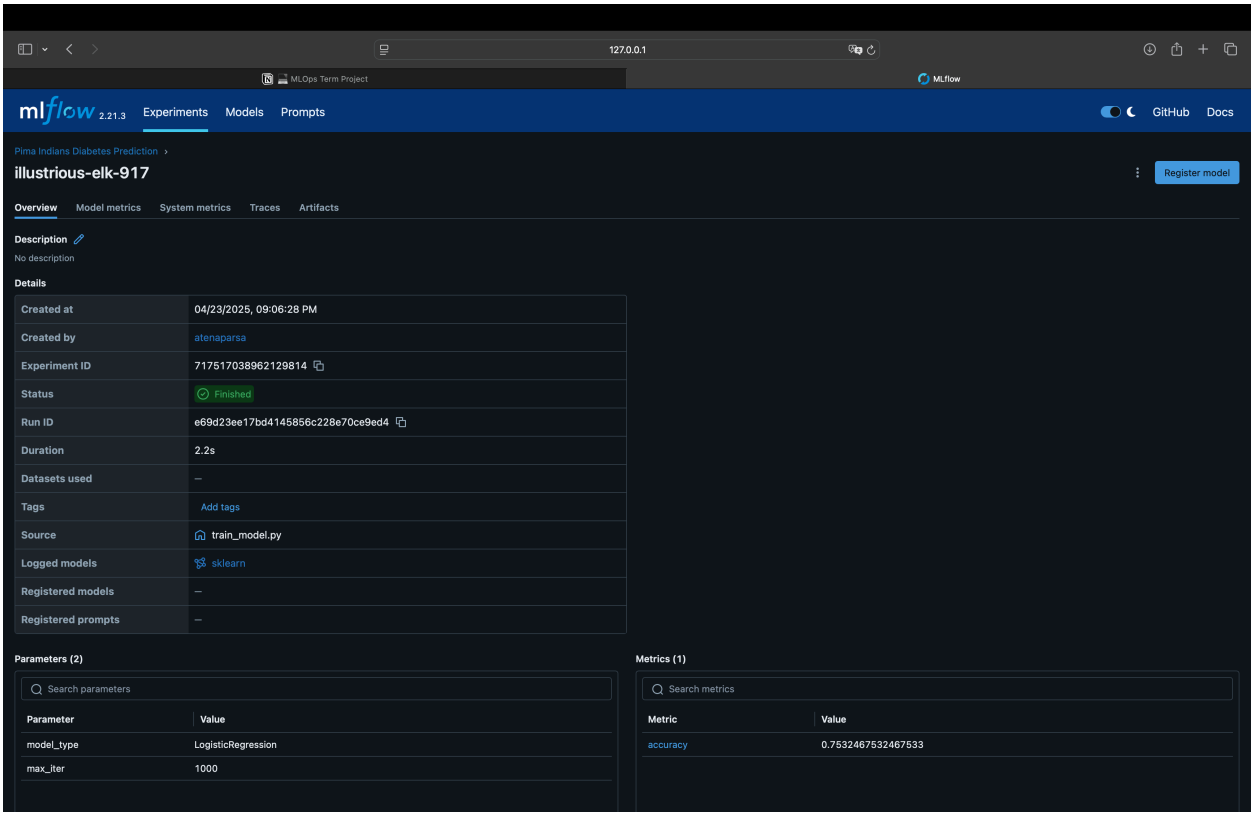
In other words, MLflow gives a project history which is very useful when comparing models or explaining how they work.

The script “train_model.py” was successfully run, meaning that a new MLflow experiment was created, the model was trained, the parameters, metrics and the model itself were logged, and a baseline accuracy of ~75.3% was achieved as shown in the screenshot below.

```
(myenv) (base) atenaparsa@Atenas-MacBook-Pro TermProject_MLOps % /opt/anaconda3/envs/myenv/b
in/python /Users/a
tenaparsa/TermProject_MLOps/scripts/train_model.py
2025/04/23 21:06:28 INFO mlflow.tracking.fluent: Experiment with name 'Pima Indians Diabetes
 Prediction' does not exist. Creating a new experiment.
2025/04/23 21:06:30 WARNING mlflow.models.model: Model logged without a signature and input
example. Please set `input_example` parameter when logging the model to auto infer the model
 signature.
Model accuracy: 0.7532467532467533
```

I started the MLflow UI on port 5001 and here is the Pima Indians Diabetes Prediction:





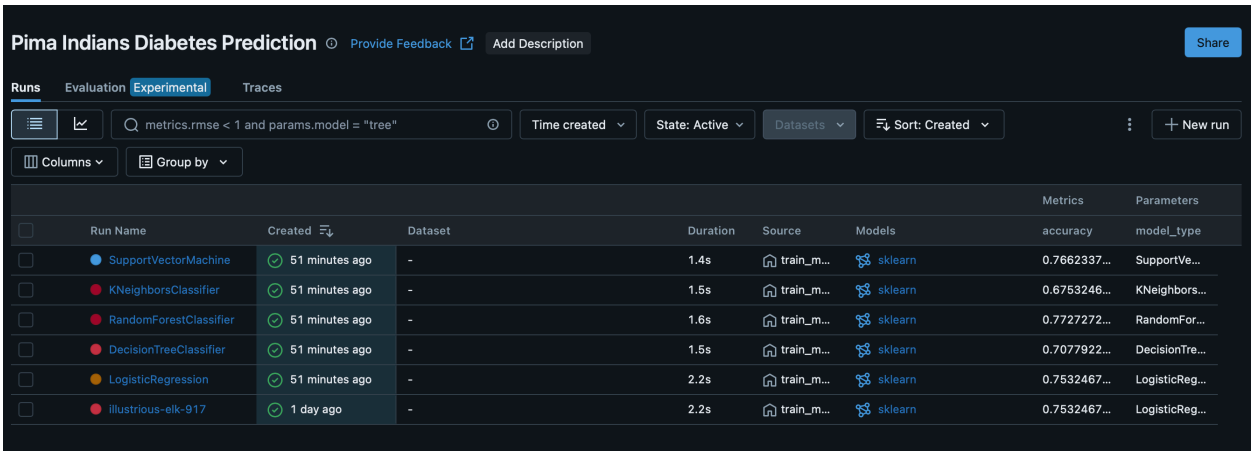
All the information about this experiment including the accuracy, source, model type (logistic regression) and experiment status are logged!

The first full MLflow lifecycle step is now officially completed. Now, let's train the second model for comparison. For more simplicity, I created a new script called "train_multiple_models.py". The process is similar to the one explained earlier. However, multiple models are defined in this case:

```
# Define models
models = {
    "LogisticRegression": LogisticRegression(max_iter=1000),
    "DecisionTreeClassifier": DecisionTreeClassifier(),
    "RandomForestClassifier": RandomForestClassifier(),
    "KNeighborsClassifier": KNeighborsClassifier(),
    "SupportVectorMachine": SVC()
}
```

5 models are defined this time, a logistic regression model (it will be the same as the one explained before), a decision tree classifier, a random forest classifier, a KNeighbors classifier and a support vector machine (SVC). Then, the MLflow experiment that was created before will be started again and there will be a loop through the models so that all five models are trained (with .fit()), tested (model.predict(X_test)), evaluated by the accuracy score, and lastly logged.

After running the script, here are all the models in the experiment Pima Indians Diabetes Prediction on the MLflow UI:



As shown above, the highest accuracy belongs to the random forest classifier. The models are ordered like this based on accuracy:

Random Forest Classifier > SVM > Logistic Regression > Decision Tree Classifier > KNeighbor Classifier

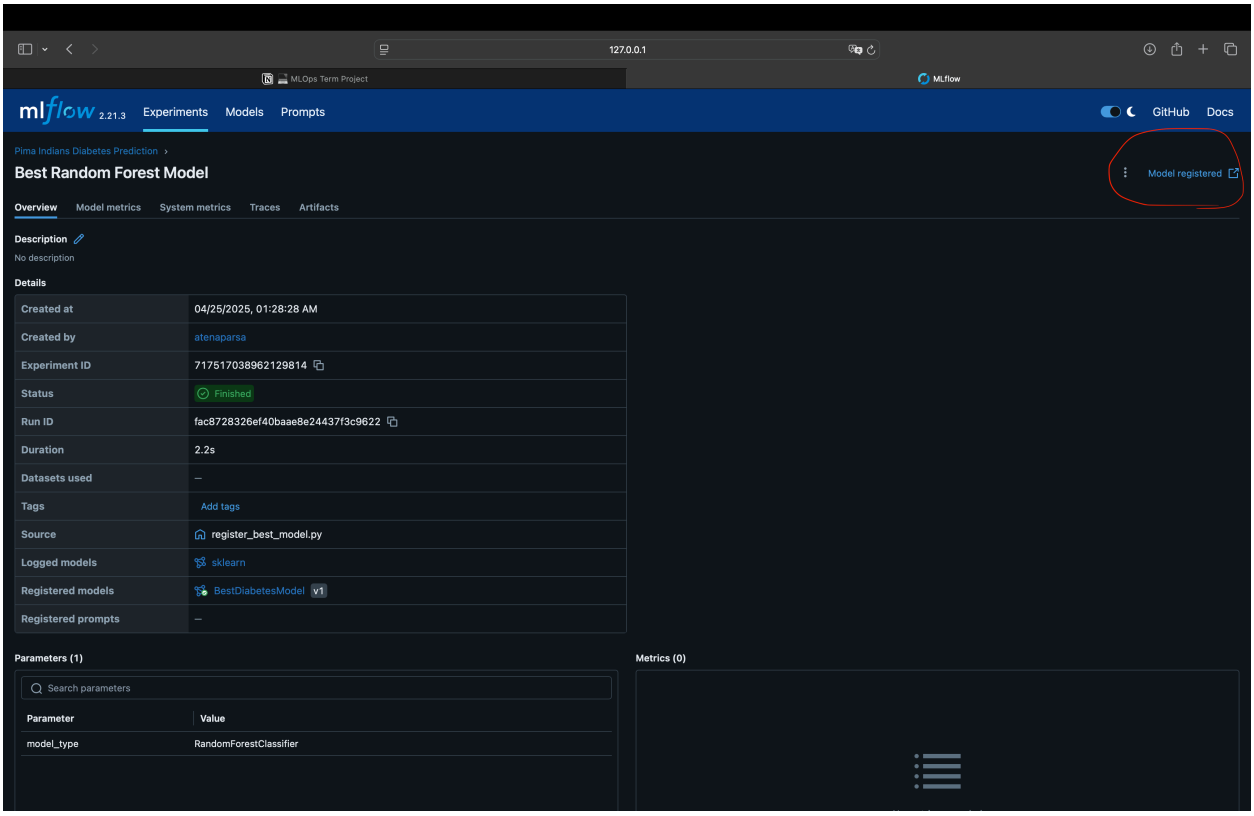
Thus, the random forest classifier will be promoted as the primary model. I will take it through the next stages of the MLOps lifecycle.

Registering the Best Model on MLflow

To register the best model in the model registry, I created a new script called register_best_model.py. Each time I try to run a new script, because it tries starting the MLflow experiment all over again, I have to restart the MLflow UI through the terminal with this command:

```
mlflow ui --port 5001
```

It would be easier to both train all models and register the best model in the same script. But I am trying to keep this project simple, explanatory and modular as I am still learning to work with MLflow. Plus, I am aiming for reusability so that I can re-train or re-load best models separately.



As you can see, best random forest model has been registered.

At the end of this stage, the best model was trained, logged to MLflow, and registered it under **BestDiabetesModel.**

mlflow_pipeline.ipynb

This notebook was created for the presentation. It includes the basic steps, setup, loading the data, train/test split, MLflow experiment set up, and lastly train and logging the models. These tasks are thoroughly done in the scripts. Then, tune_hyperparams.py, register_best_model.py, and deploy_model.py will be the only ones left to explain.

Tuning the Hyperparameters

In tune_params.py, after the basic steps of loading the dataset, cleaning the data and splitting into training and testing, firstly, a hyperparameter space is defined as

shown below:

```
# Define hyperparameter space
param_dist = {
    'n_estimators': [50, 100, 150, 200, 250],
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}
```

Different options are defined for the number of estimators in the random forest, the maximum depth (that the decision trees go in when trying to divide the data into pure groups by asking “questions”) and minimum samples split and minimum samples leaf.

Of the hyperparameters space, 10 different combination will be randomly tried (n_tier=10) and for each one, the training will be split into 3 parts. 2 parts will be used for training and one will be used for testing. This is executed by 3-fold cross validation. Lastly, the best version with the highest accuracy (the best combination from the hyperparameters space) will be picked and remembered.

```
# Run tuning with RandomizedSearchCV
rf = RandomForestClassifier(random_state=42)

random_search = RandomizedSearchCV(rf, param_distributions=param_dist, n_it
random_search.fit(X_train, y_train)

# Best model found
best_rf = random_search.best_estimator_
```

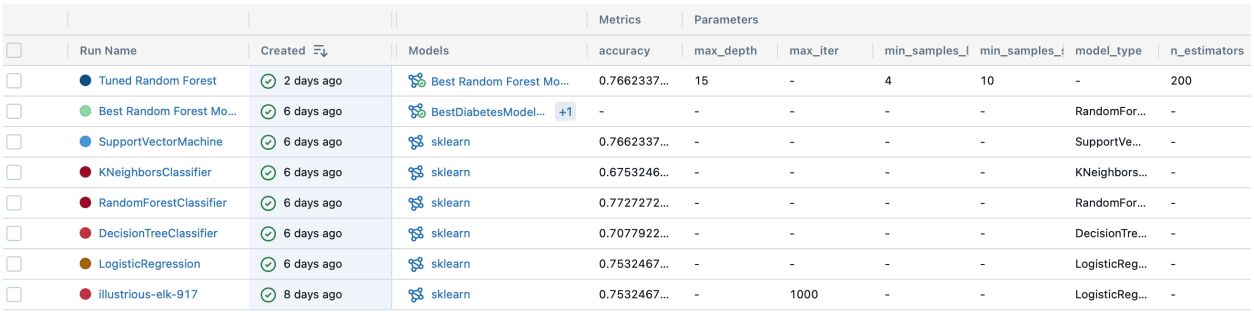
Then, the picked model will be evaluated on test data that it has not seen before:

```
# Test the best model on test set
y_pred = best_rf.predict(X_test)
```

```
acc = accuracy_score(y_test, y_pred)
print(f"Tuned Random Forest Accuracy: {acc:.4f}")
```

Finally, everything will be logged to MLflow, what settings were used, the accuracy, and the trained model itself, which could be deployed later.

As shown in the screenshot below, the first model from above is the Tuned Random Forest. The hyperparameters that were randomly selected are 15 for maximum depth of the decision trees, 4 for minimum sample leaf, 10 for minimum sample split and 200 as the number of estimators. These hyperparameters combination gave the highest accuracy comparing to the other 9 random hyperparameter combinations.



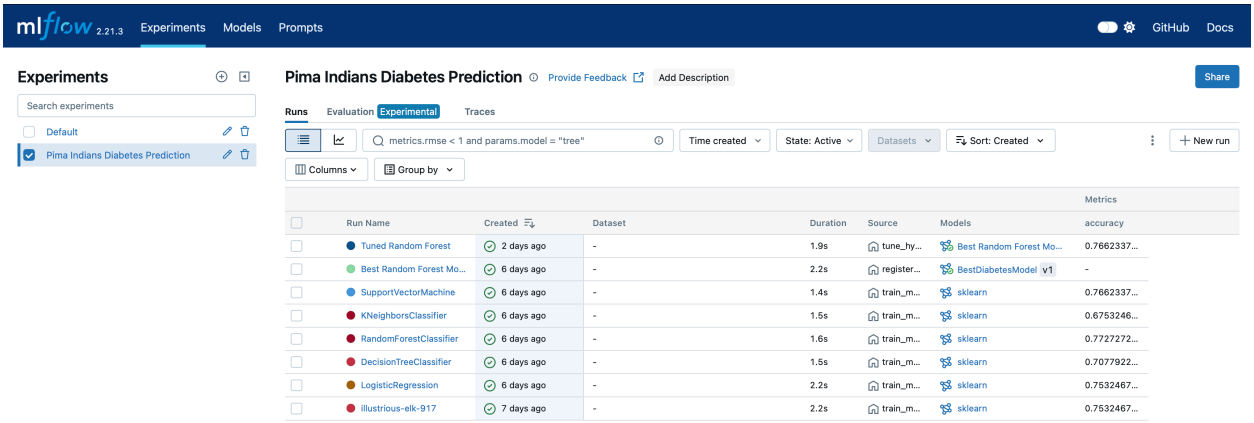
				Metrics	Parameters					
	Run Name	Created	Models	accuracy	max_depth	max_iter	min_samples_l	min_samples_s	model_type	n_estimators
	Tuned Random Forest	2 days ago	Best Random Forest Mo...	0.7662337...	15	-	4	10	-	200
	Best Random Forest Mo...	6 days ago	BestDiabetesModel... +1	-	-	-	-	-	RandomFor...	-
	SupportVectorMachine	6 days ago	sklearn	0.7662337...	-	-	-	-	SupportVe...	-
	KNeighborsClassifier	6 days ago	sklearn	0.6753246...	-	-	-	-	KNeighbors...	-
	RandomForestClassifier	6 days ago	sklearn	0.7727272...	-	-	-	-	RandomFor...	-
	DecisionTreeClassifier	6 days ago	sklearn	0.7077922...	-	-	-	-	DecisionTre...	-
	LogisticRegression	6 days ago	sklearn	0.7532467...	-	-	-	-	LogisticReg...	-
	illustrious-elk-917	8 days ago	sklearn	0.7532467...	-	1000	-	-	LogisticReg...	-

Findings

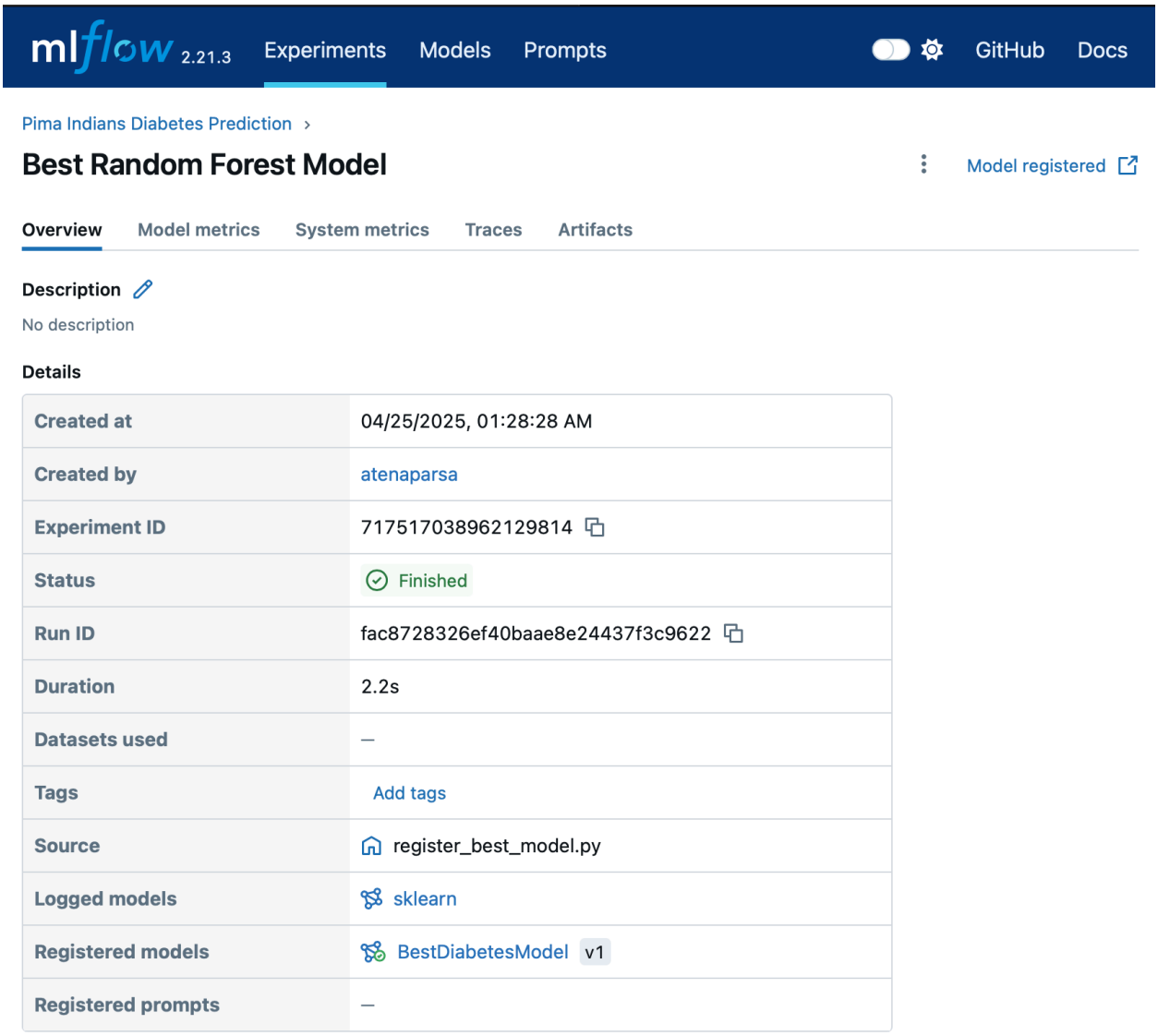
The accuracy of the random forest model was around 0.7727 before hyperparameter tuning. Interestingly, it decreased after hyper parameter tuning to around 0.76! This could be caused by the data not being very large (769 records). Thus, the original random forest model will be deployed instead of the tuned one.

Model Deployment

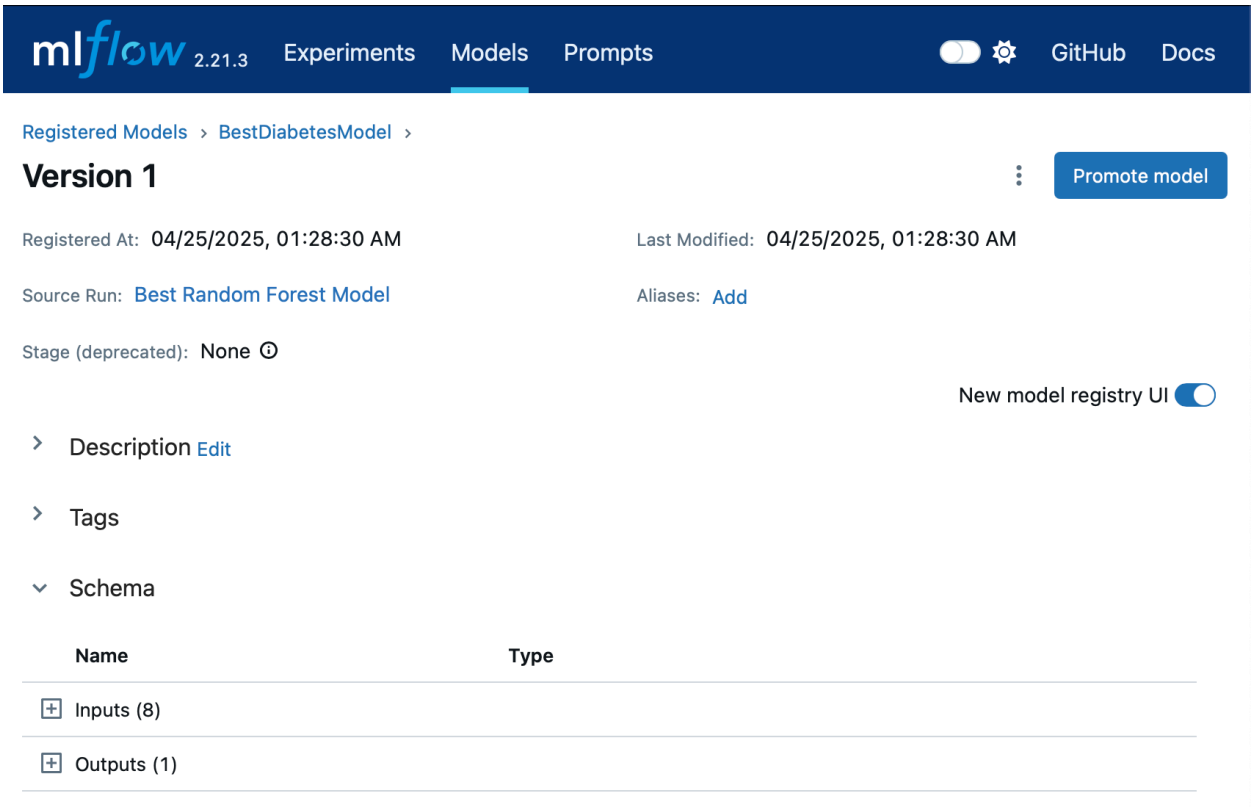
Here are all the models with their accuracy including the tuned one. The best one is still the original random forest model that was already registered as explained in **Registering the Best Model on MLflow** section.



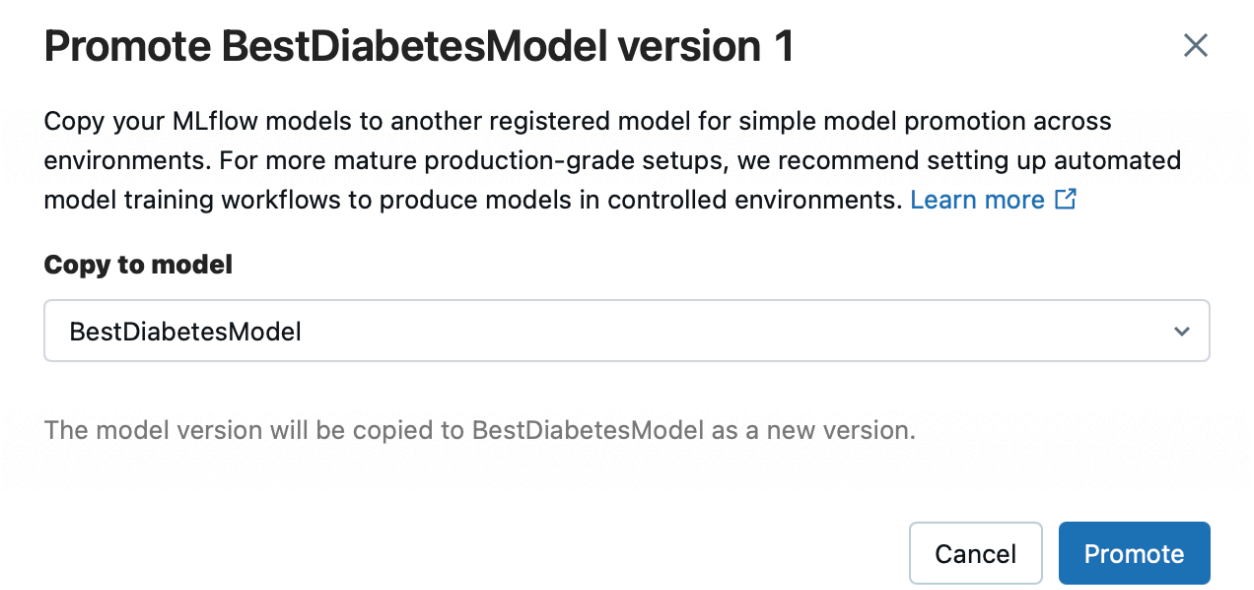
The highest accuracy belongs to the original random forest model below and it has already been registered. Let’s promote this model to the production stage.



To promote this version of the random forest model, it has to be copied either to an already existing model, or a new one has to be created.



The model is promoted as the Best Diabetes Model Version 1 as shown below.



Then, through deploy_model.py, the registered best model which is supposed to be in the production stage now, is first loaded:

```
# Load the registered best model
model = mlflow.sklearn.load_model(model_uri="models:/BestDiabetesModel/Pro
```

Then a Flask app is initialized and input data is requested in json format (pandas dataframe) and the prediction integer is posted in the 5003 port, as written in deploy_model.py .

Performance Monitoring

In order to simulate real-world usage of the deployed modelcand to log how it performs, it is a good idea to send repeated predictions, measure response time, record what was sent and what was predicted an save them all to a csv file for monitoring and analysis.

In monitor_model.py, there is an example test data for this purpose.

```
# Simulated test data
sample = {
    "Pregnancies": 2,
    "Glucose": 130,
    "BloodPressure": 70,
    "SkinThickness": 25,
    "Insulin": 80,
    "BMI": 28.0,
    "DiabetesPedigreeFunction": 0.35,
    "Age": 35
}
```

This test patient input will be sent to the model and it will simulate what a user might input in the real world. To store each prediction’s result along with time and latency, “ logs = [] “ is used.

With a for loop, the number of the requests is simulated which is 5 in this case.

The sample data is sent to the Flask API.

```
response = requests.post("http://localhost:5003/predict", json=sample)
```

Lastly, after grabbing the prediction result (0 or 1), the time the process took and the current time for the log. Also, 2 seconds is paused to mimic real time between users.

After all 5 predictions are done, the list is turned into a table (data frame) and it is saved into a CSV file to track model performance later.

Conclusion

This MLOps project successfully implemented a complete machine learning lifecycle for diabetes prediction using MLflow. The key achievements include:

- Development and comparison of 5 different classification models, with Random Forest achieving the highest accuracy (77.27%)
- Implementation of model tracking, versioning, and registration using MLflow
- Exploration of hyperparameter tuning, though the original model outperformed the tuned version
- Deployment of the best model using Flask API
- Creation of a monitoring system to track model performance and latency

The project demonstrates the practical application of MLOps principles in managing the full lifecycle of a machine learning model, from development through deployment and monitoring.

My Github Repository

https://github.com/AtenaJP22/MLOps-Term-Project

References

Dataset Retrieved from:

https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database