AI Model Deployment

Hands-on Exercise + Best Practices Recap

Step 2: Train and Save a Model Using Scikit-learn

We'll train a simple machine learning model using **scikit-learn** and save it for deployment.

Copy

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import joblib

# Load dataset
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
labels = data.target

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df, labels, test_size=0.2, ra

# Train model
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)

# Save model
joblib.dump(model, "model.pkl")
print("Model saved as model.pkl")
```

Step 3: Deploy the Model Using MLflow

MLflow is pre-installed in the environment and can be used to serve models easily.

Copy

```
mlflow models serve -m model.pkl --port 5001 --no-conda
```

This will launch a REST API that can be used to make predictions.

Step 4: Make a Prediction Request

You can send a request to the deployed model using Python:

Copy

```
import requests
import json

url = "http://127.0.0.1:5001/invocations"
data = {"instances": [[5.1, 3.5, 1.4, 0.2]]}
response = requests.post(url, json=data)
print("Prediction Response:", response.json())
```

Best Practices Recap

Key Takeaways for AI Model Deployment

- **Use Available Tools:** Leverage **scikit-learn** for training and **MLflow** for easy model deployment.
- **APIs for Model Interaction:** Exposing models via APIs allows seamless integration into applications.
- **Optimize Deployment:** Consider optimizing model size and response times when moving to production.
- **Security & Reliability:** Always consider access control and monitoring for deployed models.

- ## Scalability Strategies: When moving beyond local deployment, consider containerization (e.g., Docker) and cloud-based solutions.

< Previous