FastAPI Machine Learning Model Integration Documentation

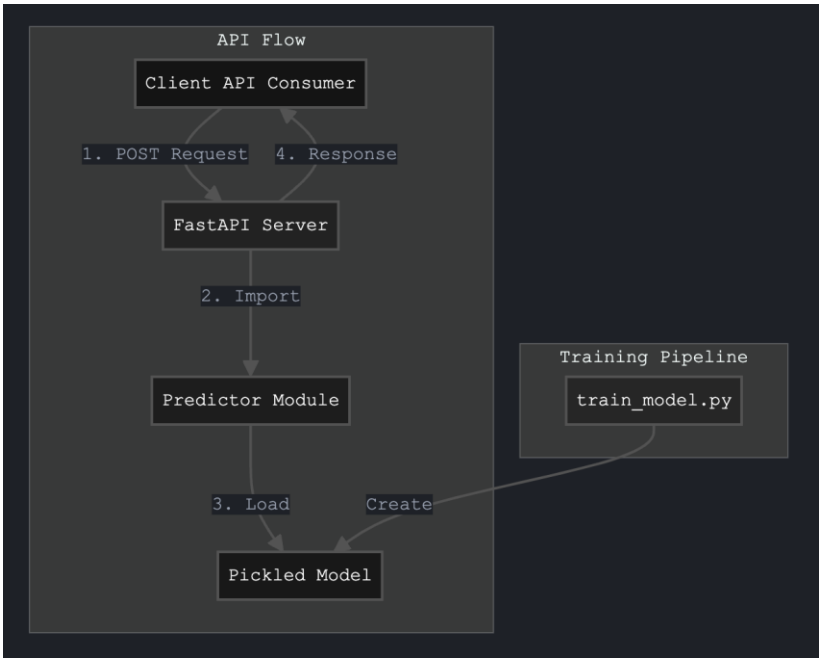
Project Overview

This project demonstrates the integration of `FastAPI` with `Python`, utilizing a serialized pickle file to handle predictions from a trained machine learning model. The system showcases:

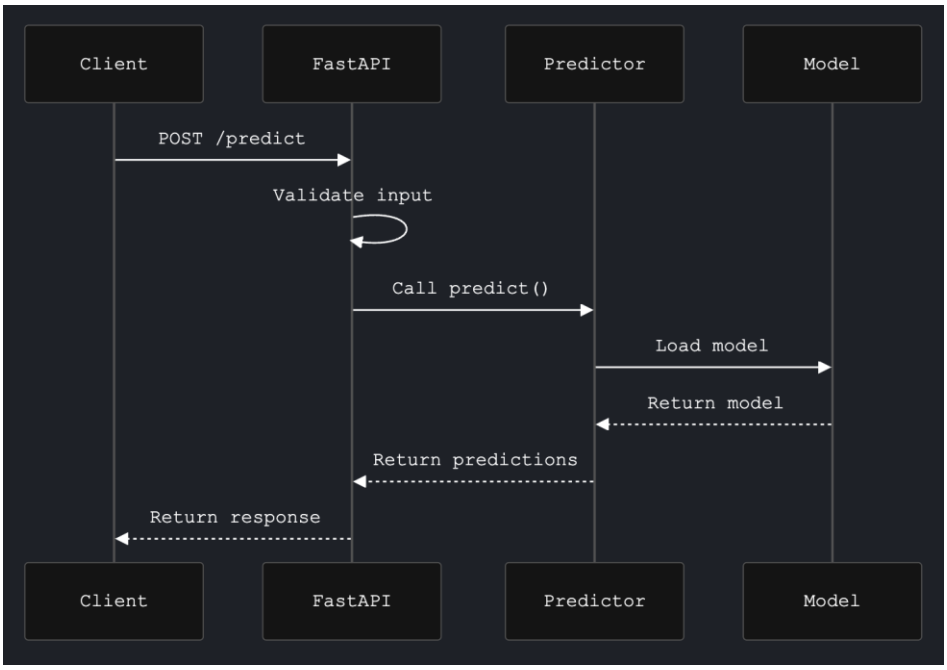
- POST request functionality in `FastAPI`
- Pickle file integration for model persistence
- Modular code structure with separation of concerns
- Error handling and input validation
- Scalable API design

System Architecture

Component Diagram



Data Flow Diagram



Directory Structure

```
#Structure of file
mid_fast_pkl/
│
├── model.pkl           # Pickled Linear Regression model
├── train_model.py      # Script to train and save the model
├── predictor.py        # Module to load model and make predictions
├── main.py            # FastAPI server implementation
└── requirements.txt    # Project dependencies
```

Component Details

1. Model Training (`train_model.py`)

train_model.py

```python
# train_model.py
import pickle
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression

# Create a simple regression dataset
X, y = make_regression(n_samples=100, n_features=1, noise=0.1, random_state=42)

# Train the model
model = LinearRegression()
model.fit(X, y)

# Save the model as a pickle file
with open("model.pkl", "wb") as f:
    pickle.dump(model, f)

print("Model trained and saved as model.pkl")
```

Key Components:

- Synthetic dataset generation using `make_regression`
- Linear Regression model training
- Model serialization using pickle

2. Prediction Module (`predictor.py`)

predictor.py

```python
# predictor.py
import pickle
import numpy as np

# Load the pickled model
with open("./model.pkl", "rb") as f:
    model = pickle.load(f)

def predict(input_data):
    # Ensure the input is a NumPy array
    input_array = np.array(input_data).reshape(-1, 1)
    # Make predictions
    predictions = model.predict(input_array)
    return predictions.tolist()
```

Key Components:

- Model deserialization
- Input data preprocessing

- Prediction generation

3. FastAPI Application (`main.py`)

main.py

```python
# main.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from predictor import predict

appmodel = FastAPI()

# Define a data model for input
class PredictionInput(BaseModel):
    input_data: list[float]  # Accepts a list of floats

@appmodel.post("/predict")
def get_prediction(data: PredictionInput):
    try:
        # Use the predictor module to generate predictions
        output = predict(data.input_data)
        return {"input": data.input_data, "predictions": output}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Key Components:

`FastAPI` server setup
`Pydantic` model for input validation
Error handling implementation

Key Features

Separation of Concerns:

- Training logic isolated in `train_model.py`
- Prediction logic encapsulated in `predictor.py`
- API handling separated in `main.py`

Scalability:

- Modular design allows easy model replacement
- Simple to add new endpoints or features
- Structured for maintainability

Security:

- Input validation using `Pydantic`
- Error handling for all endpoints
- Structured error responses