# MLOps and Deployment Documentation: Customer Churn Prediction

**Project Overview**  
This document outlines the MLOps and deployment strategy implemented for the Customer Churn Prediction project. The project follows MLOps best practices to ensure **reproducible**, **scalable**, and **maintainable** machine learning operations. The Customer Churn Prediction system uses advanced machine learning techniques to predict customer churn behavior. By analyzing patterns in customer data, businesses can identify at-risk customers and take proactive steps to improve retention.

## Model Selection and Artifacts

**Champion Model**

* **Algorithm:** XGBoost with Important Features
* **Performance Metrics:**
  + Accuracy: 95.09%
  + Precision: 95.19%
  + Recall: 94.94%
  + F1 Score: 95.04%
  + AUC: 98.50%

**Serialization**

* **Method:** Python's pickle library
* **Serialized Files:**
  + Model: best\_xgb\_model.pkl
  + Scaler: scaler.pkl

## Deployment Architecture

The project implements a dual deployment strategy to serve different use cases:

### Streamlit Application

* **Purpose:** Interactive UI for business users
* **File:** streamlit\_app.py
* **Features:**
  + User-friendly interface for inputting customer data
  + Real-time prediction visualization
  + Interactive elements for data exploration
* **Monitoring:** Logs stored in streamlit\_app.log

### FastAPI Application

* **Purpose:** Production API endpoint and web interface
* **File:** fastapi\_app.py
* **Features:**
  + RESTful API for programmatic access
  + Swagger UI documentation at /docs endpoint
  + Web interface for JSON input through HTML/CSS templates
  + Input validation using Pydantic schema
* **Web Assets:**
  + HTML templates: index.html
  + CSS styling: style.css
* **Monitoring:** Logs stored in fastapi\_app.log

## Containerization

Docker is used to containerize the FastAPI application for consistent deployment across environments:

* **Dockerfile:** Contains all dependencies and configuration
* **Base Image:** Python-based
* **Exposed Port:** 9000
* **Entry Point:** uvicorn server running the FastAPI app

## Deployment Steps

### Local Deployment

1. **Streamlit App Deployment:**  
   streamlit run streamlit\_app.py
2. **FastAPI App Deployment:**  
   uvicorn fastapi\_app:app --reload --port 9000
3. **Access at:**
   * Web interface: http://127.0.0.1:9000/
   * API documentation: http://127.0.0.1:9000/docs

### Containerized Deployment

1. **Build Docker Image:**  
   docker build -t churn-fastapi-app .
2. **Run Docker Container:**  
   docker run -d -p 9000:9000 churn-fastapi-app
3. **Access:** The same endpoints as in local deployment.

## 6. Monitoring and Logging

Key components of monitoring and logging include:

* **Logging Implementation:** Python's built-in logging module
* **Log Level:** INFO and above
* **Log Format:** Timestamp, level, and message
* **Log Files:**
  + streamlit\_app.log: All Streamlit app activities
  + fastapi\_app.log: All FastAPI requests and responses

## 7. Model Inference Pipeline

Both applications implement the same inference pipeline to ensure consistency:

1. Receive input data (form or API)
2. Apply the same preprocessing steps used during training
3. Transform features using the serialized scaler
4. Generate predictions using the serialized model
5. Return formatted results to the user

## 8. Future MLOps Enhancements

The current implementation provides a solid foundation for further MLOps capabilities:

1. Model versioning and experiment tracking
2. Automated retraining and deployment pipelines
3. Performance monitoring and drift detection
4. Cloud deployment with auto-scaling
5. A/B testing infrastructure for model comparison

## 9. Technologies Used

The following technologies are utilized:

* **Data Processing:** Pandas, NumPy, Scipy
* **Visualization:** Matplotlib, Seaborn, Plotly
* **Machine Learning:** Scikit-learn, XGBoost, Random Forest, Logistic Regression
* **Statistical Analysis:** T-Tests, Chi-Squared Tests
* **Deployment:** Streamlit, FastAPI, Pickle, Docker, Logging