**MLOps Housing Price Predictor - Architecture Summary**

**Project**: California Housing Price Prediction MLOps Pipeline  
**Author**: Pravien Madhavan  
**Date**: July 2025  
**Assignment**: MLOps Best Practices Implementation

**Executive Summary**

This MLOps pipeline implements a complete machine learning lifecycle for predicting California housing prices. The system demonstrates industry best practices including automated data processing, experiment tracking, model versioning, containerized deployment, CI/CD automation, and production monitoring.

**Architecture Overview**

The system follows a microservices architecture with clear separation of concerns across data, model, API, and deployment layers. The pipeline is designed for scalability, maintainability, and reproducibility.

**Core Components**

**1. Data Layer**

* **Data Source**: California Housing Dataset (scikit-learn)
* **Processing**: Automated feature engineering, scaling, and train/test splitting
* **Storage**: Processed datasets saved as NumPy arrays for reproducibility
* **Versioning**: Data processing pipeline ensures consistent preprocessing

**2. Model Development & Tracking**

* **Training**: Multiple algorithms (Random Forest, Linear Regression) compared systematically
* **Experiment Tracking**: MLflow tracks all parameters, metrics (R², MSE, MAE), and model artifacts
* **Model Registry**: Best performing models automatically registered with version control
* **Evaluation**: Cross-validation and holdout testing for reliable performance assessment

**3. API Service Layer**

* **Framework**: FastAPI providing high-performance REST API with automatic documentation
* **Endpoints**: /predict for inference, /health for monitoring, /prediction-history for analytics
* **Validation**: Pydantic models ensure robust input validation and error handling
* **Logging**: Comprehensive request/response logging to SQLite database

**4. Containerization & Deployment**

* **Docker**: Multi-stage builds for production-optimized containers
* **Scalability**: Stateless design allows horizontal scaling
* **Health Checks**: Built-in container health monitoring
* **Volume Mounting**: Persistent storage for models and logs

**5. CI/CD Pipeline**

* **GitHub Actions**: Automated testing, building, and deployment
* **Quality Gates**: Code linting, testing, and model validation before deployment
* **Docker Registry**: Automated image building and pushing to Docker Hub
* **Deployment Automation**: Streamlined production deployments

**Data Flow Architecture**

[Raw Data] → [Preprocessing] → [Feature Engineering] → [Model Training]  
 ↓ ↓ ↓ ↓  
[Data Storage] → [MLflow Tracking] → [Model Registry] → [API Serving]  
 ↓ ↓ ↓ ↓  
[Persistence] → [Experiment History] → [Version Control] → [Production Inference]

**Technology Stack**

|  |  |  |
| --- | --- | --- |
| Layer | Technology | Purpose |
| **Data Processing** | Python, pandas, scikit-learn | Feature engineering and preprocessing |
| **Model Training** | scikit-learn, NumPy | ML algorithms and evaluation |
| **Experiment Tracking** | MLflow | Parameter tracking, model versioning |
| **API Framework** | FastAPI, Pydantic | REST API with validation |
| **Containerization** | Docker | Consistent deployment environments |
| **CI/CD** | GitHub Actions | Automated testing and deployment |
| **Monitoring** | SQLite, Prometheus | Request logging and metrics |
| **Development** | Git, Python venv | Version control and environment management |

**Key Features & Benefits**

**Reproducibility**

* Versioned datasets and consistent preprocessing pipelines
* MLflow experiment tracking ensures reproducible model training
* Docker containers provide consistent runtime environments

**Scalability**

* Stateless API design enables horizontal scaling
* Containerized deployment supports orchestration platforms
* Modular architecture allows independent component scaling

**Monitoring & Observability**

* Comprehensive request/response logging with SQLite persistence
* Health check endpoints for service monitoring
* MLflow UI for experiment visualization and model comparison

**Quality Assurance**

* Automated testing pipeline with pytest
* Code quality checks with linting and formatting
* Model validation gates in CI/CD pipeline

**Developer Experience**

* Interactive API documentation with FastAPI/Swagger
* Local development environment with hot reloading
* Clear project structure following Python best practices

**Deployment Architecture**

The system supports multiple deployment patterns:

**Local Development**: Direct Python execution with MLflow UI  
**Containerized**: Docker containers for consistent environments  
**CI/CD**: Automated GitHub Actions pipeline for production deployment

**Security & Compliance**

* Input validation prevents injection attacks and malformed requests
* Containerized deployment provides process isolation
* Logging captures all prediction requests for audit trails
* Version control ensures code and model traceability

**Future Enhancements**

The current architecture provides a solid foundation for advanced MLOps features:

* Advanced monitoring with Prometheus/Grafana dashboards
* Automated model retraining with data drift detection
* A/B testing capabilities for model comparison
* Enhanced input validation with business rule enforcement

**Success Metrics**

The pipeline successfully demonstrates MLOps best practices:

* **Automation**: Fully automated data processing, training, and deployment
* **Reproducibility**: Consistent results across environments and runs
* **Monitoring**: Comprehensive logging and health monitoring
* **Quality**: Automated testing and validation gates
* **Scalability**: Container-ready architecture for production scaling

This architecture provides a production-ready foundation for machine learning applications while maintaining developer productivity and operational excellence.