DeepMed
MLOps Documentation

April 27, 2025

Contents

1 Introduction **3**

2 End-to-End Machine Learning Lifecycle **4**
2.1 Data Preparation and Processing 4
2.2 Model Development and Training 4
2.2.1 Tabular Model Training 4
2.2.2 Image Model Training 4
2.3 Model Evaluation and Selection 5
2.3.1 Tabular Model Metrics 5
2.3.2 Image Model Metrics 5
2.4 Model Registration and Versioning 5
2.5 Model Deployment 5
2.6 Model Serving and Inference 5
2.7 Monitoring and Maintenance 6

3 MLOps Tooling and Infrastructure **7**
3.1 Docker Containerization 7
3.1.1 Purpose and Benefits 7
3.1.2 Configuration Details 7
3.2 MLflow for Experiment Tracking 7
3.2.1 Purpose and Benefits 7
3.2.2 Configuration Details 7
3.3 Azure Blob Storage 8
3.3.1 Purpose and Benefits 8
3.3.2 Configuration Details 8
3.4 SQLAlchemy Database Integration 8
3.4.1 Purpose and Benefits 8
3.4.2 Configuration Details 8
3.5 GitHub Actions CI/CD Pipeline 8
3.5.1 Purpose and Benefits 8
3.5.2 Configuration Details 8
3.6 Flask Microservices 9
3.6.1 Purpose and Benefits 9
3.6.2 Configuration Details 9
3.7 Image Processing Services 9
3.7.1 Purpose and Benefits 9
3.7.2 Image Classification Service 9
3.7.3 Data Augmentation Service 9
3.7.4 Image Predictor Service 9
3.7.5 Anomaly Detection Service 10
3.8 Tabular Data Services 10
3.8.1 Purpose and Benefits 10
3.8.2 Feature Selection Service 10
3.8.3 Data Cleaner Service 10
3.8.4 Algorithm Training Services 10
3.8.5 Tabular Predictor Service 10

4 Monitoring Systems **11**
4.1 Prometheus and Grafana Monitoring 11
4.2 Health Check System 11

5 MLOps Best Practices Implemented **12**
5.1 Model Training Best Practices 12
5.2 MLOps Process Best Practices 12

6 Troubleshooting and Diagnostics **13**
6.1 Logging System 13
6.2 Metrics and Performance Monitoring 13

7 Conclusion **14**

Chapter 1

Introduction

This document outlines the Machine Learning Operations (MLOps) infrastructure implemented in the DeepMed platform. It details the processes, tools, and practices used for developing, deploying, monitoring, and maintaining machine learning models in production. The infrastructure is designed to ensure reproducibility, scalability, and reliability of machine learning models throughout their lifecycle. The platform supports both tabular and image-based machine learning models, with specialized services for each data type, all containerized using Docker for consistent deployment and execution.

Chapter 2

End-to-End Machine Learning Lifecycle

2.1 Data Preparation and Processing

The data preparation and processing phase encompasses several key activities. Data upload and validation ensure the integrity and quality of incoming data through comprehensive validation checks. Data cleaning processes remove inconsistencies and handle missing values using standardized preprocessing pipelines. Feature selection identifies the most relevant variables for model training through both statistical and model-based selection methods. Data transformation prepares the data in a format suitable for machine learning algorithms, including standardization, normalization, and categorical encoding. The platform supports both structured tabular data and unstructured image data, with specialized preprocessing pipelines for each data type.

2.2 Model Development and Training

The model development and training phase involves careful algorithm selection based on the problem requirements and data characteristics. For tabular data, the platform supports multiple algorithms including Logistic Regression, Decision Trees, Random Forests, Support Vector Machines, K-Nearest Neighbors, and Naive Bayes. Each algorithm is implemented as a separate microservice with standardized training interfaces. Hyperparameter optimization is performed using grid search with cross-validation, typically using 3-5 folds depending on dataset size. Model training implements the selected algorithms with the optimized parameters, tracking all experiments using MLflow for reproducibility and comparison. The training process includes automatic handling of class imbalance using SMOTE for smaller datasets and class weights for larger ones.

2.2.1 Tabular Model Training

Tabular model training follows a structured approach with comprehensive preprocessing and evaluation. The process begins with train-test split to create separate datasets for training and evaluation, with a default test size of 20% but configurable based on requirements. A preprocessing pipeline standardizes data handling through feature scaling and categorical encoding. Hyperparameter grid search systematically explores parameter combinations, with algorithm-specific parameter spaces optimized for each model type. Class imbalance handling techniques address skewed class distributions through SMOTE oversampling for smaller datasets and class weights for larger ones. Metric optimization ensures models meet performance requirements, with comprehensive tracking of accuracy, precision, recall, F1 score, and cross-validation results.

2.2.2 Image Model Training

Image model training requires specialized handling with dedicated services for different aspects of the process. Dataset organization structures image data for efficient processing through standardized directory structures and metadata tracking. Multi-tier training options provide flexibility in model complexity,

from lightweight models for quick experimentation to complex architectures for production deployment. Transfer learning leverages pre-trained models, particularly using EfficientNet B0 as a base architecture, for improved performance and reduced training time. Dataset splitting creates appropriate training, validation, and test sets with configurable ratios. Data augmentation techniques enhance model robustness through a dedicated augmentation service that supports various transformations with configurable intensity levels.

2.3 Model Evaluation and Selection

2.3.1 Tabular Model Metrics

Tabular model evaluation employs comprehensive metrics tracked throughout the training process. Accuracy measures overall prediction correctness across all classes. Precision indicates the proportion of positive predictions that are correct, calculated using weighted averaging for multi-class problems. Recall measures the ability to identify all positive cases, also using weighted averaging for multi-class scenarios. F1 score balances precision and recall to provide a single metric for model comparison. Cross-validation mean provides robust performance estimates through k-fold validation, typically using 3-5 folds. Cross-validation standard deviation indicates model stability across different data splits, helping identify potential overfitting issues.

2.3.2 Image Model Metrics

Image model evaluation tracks multiple performance indicators through dedicated monitoring services. Training accuracy measures model performance on training data, helping identify learning progress. Validation accuracy assesses generalization capability on unseen data during training. Test accuracy provides final performance evaluation on completely held-out data. Training loss monitors optimization progress through the learning curve. Validation loss helps prevent overfitting by tracking performance on validation data. Test loss confirms model effectiveness on the final test set. Class distribution analysis ensures balanced performance across all classes, with special attention to minority classes.

2.4 Model Registration and Versioning

Model registration and versioning maintain organized model management through a combination of MLflow and custom storage solutions. Model storage provides secure and accessible model persistence using Azure Blob Storage for large models and local storage for development. Metadata tracking records essential model information including training parameters, performance metrics, and dataset characteristics. Model registry centralizes model management with version control and staging capabilities. Artifact management handles associated model components such as preprocessing pipelines, label encoders, and feature selection information. All models are stored in joblib format with comprehensive metadata for easy loading and deployment.

2.5 Model Deployment

Model deployment ensures reliable model serving through containerized microservices. Containerization packages models with their dependencies using Docker, ensuring consistent execution environments. CI/CD pipeline automates deployment processes through GitHub Actions, with automated testing and validation. Environment promotion manages model progression through stages from development to production. Deployment verification confirms successful implementation through health checks and performance validation. Each model type (tabular and image) has dedicated deployment services with optimized configurations for their specific requirements.

2.6 Model Serving and Inference

Model serving and inference provide prediction capabilities through RESTful APIs. Prediction services handle model execution with optimized inference pipelines for each model type. Feature consistency ensures input data compatibility through standardized preprocessing and validation. API access enables

model integration with various client applications through well-documented endpoints. Response formatting standardizes prediction outputs with clear error handling and status codes. The platform supports both synchronous and asynchronous prediction modes, with configurable timeouts and batch processing capabilities.

2.7 Monitoring and Maintenance

Monitoring and maintenance ensure ongoing model reliability through comprehensive observability. Health monitoring tracks system status through dedicated health check services for each component. Performance tracking measures model effectiveness through real-time metrics collection and analysis. Resource utilization optimizes system efficiency with automatic scaling based on load. Model updates maintain model relevance through automated retraining triggers and performance degradation detection. The platform uses Prometheus for metrics collection and Grafana for visualization, with custom dashboards for different model types and deployment stages.

Chapter 3

MLOps Tooling and Infrastructure

3.1 Docker Containerization

3.1.1 Purpose and Benefits

Docker containerization provides essential infrastructure benefits for consistent and scalable deployment. Consistency ensures uniform execution environments across development, testing, and production. Isolation prevents interference between services through container-level resource separation. Scalability enables flexible resource allocation with automatic container orchestration. Portability facilitates deployment across environments from local development to cloud production. Each model type and service has its own optimized container configuration with appropriate resource limits and networking settings.

3.1.2 Configuration Details

Docker configuration manages container execution with detailed specifications for each service. Base image selection determines the foundation environment, using lightweight Python images with necessary ML libraries. Dependency management handles required packages through comprehensive requirements files and layer optimization. Environment variables configure container behavior for different deployment scenarios. Volume mounting manages persistent storage for models and data with appropriate access controls. Networking enables service communication through Docker networks with configurable security policies. Each container includes health check endpoints and logging configurations for monitoring.

3.2 MLflow for Experiment Tracking

3.2.1 Purpose and Benefits

MLflow organizes machine learning experiments effectively with comprehensive tracking capabilities. Experiment organization structures research activities through project-based grouping and tagging. Parameter logging records configuration details including hyperparameters and preprocessing steps. Metric tracking monitors performance indicators throughout the training process. Artifact storage manages model components with version control and access control. Run comparison facilitates experiment analysis through visual comparison tools and metric aggregation. The platform uses MLflow for both tabular and image model experiments, with custom tracking for each model type.

3.2.2 Configuration Details

MLflow configuration establishes tracking infrastructure with optimized settings for production use. Tracking URI setup connects to the tracking server with appropriate authentication and access controls. Parameter and metric logging captures experiment details with automatic type conversion and validation. Model registration manages model versions with staging capabilities and metadata tracking. Run context management organizes experiment execution with proper resource cleanup and error handling. The platform uses a combination of local and remote MLflow tracking servers based on deployment environment.

3.3 Azure Blob Storage

3.3.1 Purpose and Benefits

Azure Blob Storage provides robust model storage capabilities for the platform's machine learning artifacts. Model persistence ensures long-term availability with configurable redundancy and backup policies. Scalable storage accommodates growing needs with automatic capacity management. Secure access protects sensitive data through Azure's security features and custom access controls. URL-based retrieval enables efficient access to models and artifacts through optimized endpoints. The platform uses Azure Blob Storage for both model storage and large dataset management, with appropriate lifecycle policies.

3.3.2 Configuration Details

Azure Blob Storage configuration manages data handling with optimized settings for ML workloads. Client initialization establishes storage connections with appropriate retry policies and timeouts. Model upload function handles file transfers with progress tracking and checksum verification. URL generation creates access links with appropriate expiration and access controls. File naming convention maintains organization through standardized prefixes and metadata. The platform implements custom wrappers around Azure Blob Storage for model-specific operations and versioning.

3.4 SQLAlchemy Database Integration

3.4.1 Purpose and Benefits

SQLAlchemy database integration manages structured data for the platform's metadata and tracking needs. User authentication controls system access through role-based permissions and audit logging. Training run tracking records experiment history with comprehensive metadata and relationships. Model metadata storage maintains model information including performance metrics and configuration details. Relationship management organizes data connections between models, experiments, and deployments. The platform uses SQLAlchemy with PostgreSQL for reliable and scalable data management.

3.4.2 Configuration Details

SQLAlchemy configuration establishes database operations with optimized settings for ML workloads. Database models define data structures with appropriate relationships and constraints. Model storage function handles data persistence with transaction management and error handling. The platform implements custom models for tracking training runs, model versions, and deployment configurations.

3.5 GitHub Actions CI/CD Pipeline

3.5.1 Purpose and Benefits

GitHub Actions CI/CD pipeline automates development processes with comprehensive testing and deployment. Automated builds ensure consistent deployment packages through standardized build processes. Continuous deployment streamlines release processes with automated testing and validation. Health verification confirms system functionality through comprehensive test suites. Scheduled monitoring maintains system awareness through automated checks and alerts. The platform uses GitHub Actions for both model training and deployment automation.

3.5.2 Configuration Details

GitHub Actions configuration manages automation workflows with detailed specifications. Build and push workflow handles package creation with dependency management and testing. Health check workflow monitors system status with comprehensive service checks. The platform implements custom GitHub Actions for model-specific operations and deployment scenarios.

3.6 Flask Microservices

3.6.1 Purpose and Benefits

Flask microservices provide flexible API development with optimized performance for ML workloads. Lightweight API development enables rapid deployment with minimal overhead. Modular design supports independent service development and scaling. Flexible routing manages request handling with comprehensive error management. WSGI compliance ensures standard compatibility with various deployment options. The platform uses Flask for all ML services with custom middleware for common functionality.

3.6.2 Configuration Details

Flask configuration establishes service infrastructure with production-ready settings. Application initialization sets up the service environment with appropriate logging and error handling. Health endpoint monitors service status with comprehensive system checks. Training endpoint handles model training with progress tracking and resource management. Prediction endpoint processes inference requests with optimized performance and error handling. Production server manages service deployment using Waitress for production workloads.

3.7 Image Processing Services

3.7.1 Purpose and Benefits

Image processing services provide specialized capabilities for computer vision workloads. Specialized image handling optimizes visual data processing with efficient memory management. Model training implements computer vision algorithms with transfer learning support. Data augmentation enhances training effectiveness through configurable transformations. Isolated prediction ensures reliable inference with proper resource management. Containerization manages service deployment with GPU support when available. The platform implements dedicated services for image classification, augmentation, and prediction.

3.7.2 Image Classification Service

The image classification service implements visual recognition with comprehensive training and inference capabilities. Service implementation provides classification capabilities through RESTful APIs. Training endpoint manages model development with progress tracking and resource optimization. Tiered training levels offer flexibility from quick experimentation to production deployment. EfficientNet B0 architecture provides efficient processing with transfer learning support. Automated dataset splitting organizes training data with configurable ratios. Comprehensive metrics evaluate model performance with detailed logging.

3.7.3 Data Augmentation Service

The data augmentation service enhances training data through configurable transformations. API endpoint handles augmentation requests with progress tracking and resource management. Configurable intensity levels control transformation strength for different training stages. Augmentation implementation applies transformations with efficient memory usage. Automatic scaling manages resource usage based on workload. Memory-efficient processing optimizes performance for large datasets. The service supports various augmentation techniques including rotation, flipping, and color adjustments.

3.7.4 Image Predictor Service

The image predictor service handles model inference with optimized performance and reliability. Service implementation provides prediction capabilities through RESTful APIs. Prediction endpoint processes classification requests with configurable batch sizes. Isolated execution environment ensures reliability through containerization. Dependency isolation prevents conflicts between different model versions. Secure execution protects system integrity with proper access controls. The service includes comprehensive error handling and logging for production use.

3.7.5 Anomaly Detection Service

The anomaly detection service identifies unusual patterns in image data through unsupervised learning. Service implementation provides detection capabilities with configurable sensitivity. Unsupervised approach enables pattern discovery without labeled data. Training endpoint manages model development with automatic feature extraction. Detection endpoint processes anomaly requests with efficient scoring. Tiered training configuration offers flexibility for different use cases. Autoencoder architecture implements detection with reconstruction error analysis. EEP middleware integration enables efficient processing of large image sets.

3.8 Tabular Data Services

3.8.1 Purpose and Benefits

Tabular data services handle structured data processing with comprehensive feature engineering and model training capabilities. Specialized data processing optimizes feature handling through efficient pipelines. Feature engineering creates predictive variables with automatic type detection and transformation. Algorithm selection matches methods to problems through comprehensive evaluation. Hyperparameter optimization fine-tunes models with efficient search strategies. Standardized pipelines ensure consistency across different data types. Containerization manages service deployment with appropriate resource allocation. The platform implements dedicated services for feature selection, data cleaning, and model training.

3.8.2 Feature Selection Service

The feature selection service identifies relevant variables through statistical and model-based methods. Service implementation provides selection capabilities with configurable criteria. Dual selection methods offer flexibility for different data types and problems. API endpoints handle selection requests with progress tracking. Preprocessing capabilities prepare data with automatic type handling. Feature evaluation assesses variable importance through multiple metrics. Comprehensive logging tracks selection processes with detailed metadata. The service supports both filter and wrapper methods for feature selection.

3.8.3 Data Cleaner Service

The data cleaner service maintains data quality through comprehensive preprocessing pipelines. Service implementation provides cleaning capabilities with configurable rules. Cleaning operations standardize data through type-specific transformations. API endpoints handle cleaning requests with validation and error handling. Configuration options customize cleaning processes for different data types. Quality reporting documents data improvements with detailed metrics. The service includes automatic detection of data quality issues and appropriate handling strategies.

3.8.4 Algorithm Training Services

Algorithm training services implement machine learning with comprehensive model management. Service implementation provides training capabilities through RESTful APIs. Algorithms available support various problems with appropriate configurations. Common API structure standardizes interaction across different models. Training process implements learning algorithms with progress tracking. MLflow integration tracks experiments with comprehensive metadata. The platform supports multiple algorithms including Logistic Regression, Decision Trees, Random Forests, SVMs, KNN, and Naive Bayes.

3.8.5 Tabular Predictor Service

The tabular predictor service handles model inference with optimized performance and reliability. Service implementation provides prediction capabilities through RESTful APIs. Prediction workflow processes requests with configurable batch sizes. API endpoints handle prediction requests with validation and error handling. Model loading manages model deployment with version control. Security features protect system integrity with proper access controls. The service includes comprehensive monitoring and logging for production use.

Chapter 4

Monitoring Systems

4.1 Prometheus and Grafana Monitoring

Prometheus and Grafana provide comprehensive monitoring with detailed metrics and visualization. Prometheus time-series database stores metrics with efficient querying capabilities. Grafana visualization presents monitoring data through customizable dashboards. Metrics collection gathers system information from all services. Dashboard access enables system oversight with role-based permissions. The platform implements custom metrics for model performance, resource usage, and service health.

4.2 Health Check System

The health check system monitors service status through comprehensive checks and alerts. Service implementation provides monitoring capabilities with configurable checks. Service health endpoints report status with detailed diagnostics. Reporting interface presents system information through web dashboards. Historical status tracking maintains system history for trend analysis. Alerting capabilities notify of issues through multiple channels. The system includes automatic recovery procedures for common failure scenarios.

Chapter 5

MLOps Best Practices Implemented

5.1 Model Training Best Practices

Model training follows established best practices with comprehensive validation and tracking. Cross-validation ensures robust performance estimates through k-fold validation. Hyperparameter tuning optimizes model configuration through grid search and random search. Imbalanced data handling addresses class distribution through SMOTE and class weights. Metrics selection matches evaluation to objectives with appropriate averaging methods. The platform implements automatic model selection based on validation performance and resource requirements.

5.2 MLOps Process Best Practices

MLOps processes implement industry standards with comprehensive automation and monitoring. Reproducibility ensures consistent results through containerization and dependency management. Versioning tracks system changes with comprehensive metadata and rollback capabilities. Automation streamlines operations through CI/CD pipelines and scheduled tasks. Monitoring maintains system awareness through comprehensive metrics and alerts. The platform implements automated testing and validation at all stages of the ML lifecycle.

Chapter 6

Troubleshooting and Diagnostics

6.1 Logging System

The logging system tracks system activity with comprehensive context and severity levels. Logging configuration manages log collection with appropriate retention policies. Error logging records system issues with detailed stack traces and context. Success logging documents normal operation with relevant metrics. The platform implements structured logging with appropriate aggregation and analysis tools.

6.2 Metrics and Performance Monitoring

Metrics and performance monitoring track system health with comprehensive coverage. Prometheus client integration collects metrics with appropriate sampling rates. Custom metrics gather specific information for different model types. Service-specific metrics monitor individual components with detailed breakdowns. Centralized visualization presents system status through Grafana dashboards. Historical data analysis identifies trends through time-series analysis. Alert configuration manages notifications with appropriate thresholds and cooldowns. Dashboard categories organize monitoring views by service type and importance.

Chapter 7

Conclusion

The DeepMed MLOps infrastructure implements a comprehensive approach to the machine learning lifecycle, ensuring reliable, reproducible, and scalable machine learning operations through Docker, MLflow, Azure Blob Storage, SQLAlchemy, GitHub Actions, Flask microservices, and centralized monitoring. The platform supports both tabular and image-based machine learning with specialized services for each data type, comprehensive monitoring and maintenance capabilities, and automated deployment processes. Through careful implementation of industry best practices and robust infrastructure components, the platform enables efficient development, deployment, and maintenance of machine learning models in production environments.