AIOps Concepts and Components: [Your Project/Repo Name]

1. Introduction: LLMOps with Prompt Flow

alt text

- **What is LLMOps?** According to the AI Engineer's Handbook, LLMOps involves "deploying, monitoring, and maintaining LLMs in production; encompassing model versioning, deployment, monitoring, and maintenance best practices."
- **What is the objective?** This project, we aim to provides an LLMOps (AIOps for Large Language Models) template and guidance for building, experimenting, evaluating, and deploying LLM-infused applications using Prompt Flow.
- **Core Idea:** Bringing engineering rigor and automation to the lifecycle of LLM applications.

1.1 Presentation Agenda

- **Introduction & Overview:** Understanding LLMOps and its importance
- **Challenges & Solutions:** Problems addressed by AIOps template
- **Core Concepts & Components:** Building blocks of the AIOps framework
- **Architecture Design:** Technical implementation details
- **Implementation Workflow:** End-to-end process from development to deployment
- **Use Case Applications:** QCP and CMC implementation examples
- **Best Practices & Future Directions:** Guidelines and roadmap
- **Q&A Session:** Discussion and clarification

1.2 Supported Platforms & Execution

- **Supported Platforms:**
  - Azure AI Studio
  - Azure Machine Learning (AML)
- **Execution Flexibility:**
  - Local execution for development and testing.
  - Azure-based execution for scalability and production.

1.2 Flow Types & Orchestration

- **Supported Flow Types:**
  - Flexible Flows: Python Function-based, Python Class-based.
  - DAG Flows: YAML-based.
  - Automatic detection and execution of flow types.
- **CI/CD Orchestration:**
  - GitHub Actions
  - Azure DevOps
  - Jenkins
- **Focus:**
  - Inner-Loop: Experimentation and Evaluation.
  - Outer-Loop: Deployment and Inferencing.

2. Challenges in LLMOps Addressed

- **Managing Complexity:** Handling multiple LLM flows, each with unique lifecycles from experimentation to production.
- **Experimentation Rigor:** Systematically managing prompt variants, hyperparameters, and evaluating their performance.
- **Deployment Consistency:** Ensuring smooth and reliable deployments across different environments.
- **Data Management:** Bringing discipline to data preparation for training, experimentation, and evaluation (DataOps).
- **Reducing Boilerplate:** Enabling configuration-driven development to focus on core logic.

3. Core AIOps/LLMOps Concepts in this Repository

- **Centralized Code Hosting:** A single repository structure to manage multiple Prompt Flow use cases.
- **Independent Lifecycle Management:** Each flow (use case) has its own lifecycle from local development to production.
- **Variant and Hyperparameter Experimentation:** Robust support for defining and evaluating multiple configurations for flows.

3.1 More Core AIOps/LLMOps Concepts

- **A/B Deployment:** Facilitates comparing different flow versions in real-world settings.
- **Many-to-Many Dataset/Flow Relationships:** Allows using multiple datasets for each standard and evaluation flow.
- **Multiple Deployment Targets:** Configuration-driven deployment to:
  - Kubernetes (including ARC-enabled)
  - Azure Web Apps
  - Azure ML/AI Studio Managed Compute

3.2 Additional Core AIOps/LLMOps Concepts

- **Comprehensive Reporting:** Automated generation of CSV and HTML reports for experiment runs and evaluation metrics.
- **Configuration-Based Development:** Minimizing custom code through declarative configurations (e.g., `experiment.yaml`, `deployment_config.json`).
- **DataOps Integration:** Separating data pipelines from prompt engineering flows, managing datasets as versioned assets in Azure ML.

4. Key Components & Features

- **Prompt Flow:** The core engine for developing, evaluating, and deploying LLM workflows.
- **Standardized Folder Structure:**
  - `.azure-pipelines/`, `.github/`, `.jenkins/`: CI/CD pipeline definitions.
  - `configs/`: Deployment configurations (`deployment_config.json`).
  - `data/`: Raw data files for flows (e.g., `.jsonl`).
  - `environment/`: Dockerfiles and environment specifications (`env.yaml`).

4.1 More Key Components & Features

- **Standardized Folder Structure (continued):**
  - `flows/`: Contains standard and evaluation prompt flows.
  - `tests/`: Unit tests for flows.
  - `data-pipelines/` (Optional): For DataOps implementation.
  - `llmops/`: Core Python modules for flow execution, evaluation, deployment.
  - `dataops/`: Core Python modules for DataOps pipelines.

4.2 Configuration Files

- **`experiment.yaml`:**
  - Central configuration file for each use case.
  - Defines flow paths, connections (e.g., to Azure OpenAI), datasets (sources, mappings), and evaluators.
  - Supports environment-specific overlays (e.g., `experiment.dev.yaml`, `experiment.pr.yaml`).
- **`config.py` (in `llmops/`):**
  - Global setting (`EXECUTION_TYPE`) to switch between `LOCAL` and `AZURE` execution.

4.3 CI/CD & Secrets Management

- **CI/CD Automation:**
    - **PR Validation:** Automated checks on Pull Requests (linting, unit tests, run on minimal data).
    - **CI Pipelines:** Triggered on merges to main/development branches for full build, test, evaluation, and deployment.
    - **Steps:** Registering data assets, running bulk experiments, executing evaluation flows, deploying endpoints, testing endpoints.
- **Secrets Management:**
    - **Local:** `.env` file at the root (gitignored).
    - **Cloud (GitHub):** `ENV_VARS` repository secret.
    - **Cloud (Azure DevOps):** Library variable groups.
    - Placeholders like `${SECRET_NAME}` used in configuration files.

4.4 Data Management & Example Use Cases

- **Data Management (DataOps):**
  - `dataops_config.json`: Configuration for data pipelines.
  - Scripts to process raw data and register it as Azure ML Data Assets.
  - Flows consume data from these registered assets.
- **Example Use Cases Provided:**
  - Web Classification (YAML-based)
  - Named Entity Recognition (YAML-based)
  - Math Coding (YAML-based)
  - Chat with PDF (RAG-based, YAML)
  - Code Generation (Function-based)
  - Chat Application (Class-based)

5. High-Level Workflow

1. **Local Development & Experimentation:**
    - Define/modify flows (standard and evaluation).
    - Configure `experiment.yaml` for the use case.
    - Set up `.env` for local secrets.
    - Run experiments and evaluations locally using provided Python scripts (`llmops.common.prompt_pipeline`, `llmops.common.prompt_eval`).
2. **Source Control & PR Validation:**
    - Commit changes to a feature branch.
    - Create a Pull Request.
    - Automated PR pipeline runs (linting, tests, minimal flow execution).

5.1 CI/CD Pipeline & Post-Deployment

3. **CI/CD Pipeline (on Merge):**
    - **Setup:** Authenticate to Azure, install dependencies.
    - **Data Registration:** Register/update datasets in Azure ML.
    - **Bulk Run (Experimentation):** Execute standard flow(s) with variants against datasets on Azure.
    - **Evaluation:** Execute evaluation flow(s) using the outputs of the bulk run.
    - **Reporting:** Generate and publish metrics reports.
    - **(Optional) Manual Approval Gate:** Human validation of metrics.
    - **Deployment:**
        - Build Docker image (if deploying to Web Apps/AKS).
        - Deploy flow to the configured target (AML Managed Endpoint, AKS, Web App).

6. Benefits for Your Client

- **Accelerated Development:** Faster iteration on LLM features due to streamlined processes and automation.
- **Improved Quality & Reliability:** Rigorous testing, evaluation, and consistent deployments.
- **Scalability:** Easily scale LLM applications using Azure's cloud infrastructure.
- **Cost Efficiency:** Optimized resource usage through managed compute and efficient experimentation.

6.1 More Benefits for Your Client

- **Enhanced Collaboration:** Standardized tools and processes for data scientists, ML engineers, and DevOps.
- **Reproducibility:** Versioned code, data, and configurations ensure experiments and deployments are reproducible.
- **Flexibility:** Supports various flow types and deployment targets to fit diverse needs.

7. Q&A

8. AIOps Design Architecture

- **Architecture Overview:** A comprehensive LLMOps architecture supporting both experimentation and deployment of large language model applications.
- **Development Environment:**
  - Local execution capabilities with Python scripts
  - VS Code integration through extensions
  - Azure AI Studio-compatible workflows
- **Flow Types & Implementation:**
  - YAML-based flows: Traditional DAG-based workflows (e.g., web_classification)
  - Function-based flows: Python functions with prompt flow capabilities
  - Class-based flows: Python classes with more complex state management

8.1 AIOps Design - Key Components

- **Experimentation & Evaluation (Inner Loop):**
  - Variant testing with multiple prompt configurations
  - Comprehensive metrics collection and reporting
  - Multi-dataset support for robust evaluation
  - A/B testing capabilities
- **Deployment Targets (Outer Loop):**
  - Azure ML Compute with managed endpoints
  - Kubernetes deployments (including AKS)
  - Azure Web Apps using Docker containers
  - Support for A/B deployment strategies
- **DataOps Integration:**

8.2 AIOps Design - Architecture Benefits

- **Complete Lifecycle Management:** End-to-end coverage from experimentation to production
- **Flexibility in Development:** Multiple flow types to suit various use cases
- **Robust Evaluation:** Comprehensive evaluation with multiple metrics and datasets
- **Deployment Options:** Multiple Azure-based deployment targets to meet different requirements:
    - Scalability with Azure Kubernetes Service
    - Managed services with Azure ML endpoints
    - Cost-effective options with Azure Web Apps
- **CI/CD Integration:** Automated workflows ensuring quality and accelerating delivery

9. Use Case Application: QCP and CMC

- **Quality Control Platform (QCP):**
  - Streamlines generation of quality control reports by integrating disparate data sources
  - Uses LLMs to analyze data and generate reports based on predefined rules
  - Currently in development stage with Azure DevOps deployment
- **CMC (Content Management and Compliance):**
  - RAG-based solution for finding relevant historical documents and generating summaries
  - Helps users find similar questions/answers from past documents during submission processes
  - Uses Azure AI Search and OpenAI models from the AI marketplace

9.1 QCP and CMC Requirements

- **QCP Requirements:**
  - Data processing pipeline for bacteria samples analysis
  - Manual data ingestion with temporary processing (no permanent storage)
  - High UI availability with low latency for real-time analysis
  - Content moderation services for model safety
  - Evaluation using golden datasets and user feedback
- **CMC Requirements:**
  - Timer-triggered data ingestion pipeline (daily/weekly)
  - Vector search capabilities for document retrieval
  - Confidential data handling within Novo network
  - Document summarization capabilities

9.2 How AIOps Addresses These Requirements

- **For QCP:**
  - Standardized prompt flow development for data analysis logic
  - Evaluation flows for measuring accuracy with golden datasets
  - Automated deployment pipelines through Azure DevOps integration
  - Monitoring capabilities for performance tracking
  - Content filtering and prompt shielding capabilities
- **For CMC:**
  - RAG-based flow templates (e.g., Chat with PDF example)
  - Azure AI integration for vector search functionality
  - Configurable data pipeline structure through DataOps modules
  - Secure deployment within Azure environments

9.3 AIOps Limitations and Future Enhancements

- **Current Limitations:**
  - Limited support for Databricks integration in data pipelines
  - Need for custom connectors to Azure AI Search beyond standard templates
  - Lack of specialized metrics for RAG evaluation in the CMC use case
  - Limited fine-tuning options for both applications' specific domains
  - No built-in content moderation services (relies on Azure's services)
- **Recommended Enhancements:**
  - Develop Databricks-specific DataOps connectors for CMC
  - Implement specialized RAG evaluation metrics for search quality
  - Add domain-specific prompt templates for QCP's bacteria analysis
  - Enhance monitoring dashboards for usability metrics

10. LLMOps Components in Detail

- **Definition:** According to the AI Engineer's Handbook, "LLMOps is a set of practices and tools for deploying, monitoring, and maintaining Large Language Models in production. It extends MLOps principles specifically for LLM applications."

Development to Production Workflow for LLMs

10.1 LLMOps Key Components - Deployment

- **Model Versioning and Deployment:**
  - Managing different model versions
  - Enabling seamless rollbacks and updates
  - Supporting A/B testing for model variants
- **Infrastructure Management:**
  - Setting up necessary hardware and software environments
  - Ensuring efficient model execution
  - Configuring deployment targets (AKS, Azure ML, Web Apps)
- **Scaling and Performance Optimization:**
  - Adjusting resources based on demand
  - Implementing horizontal and vertical scaling strategies

10.2 LLMOps Key Components - Monitoring

- **Response Quality Tracking:**
    - Assessing generated response quality
    - Validating against user expectations
- **Performance Metrics:**
    - Measuring latency and throughput
    - Evaluating model efficiency in real-time
- **Usage Analytics:**
    - Analyzing user interaction patterns
    - Identifying improvement opportunities
- **Error Monitoring:**

10.3 LLMOps Key Components - Maintenance

- **Model Updates and Versioning:**
  - Incorporating new data and improvements
  - Maintaining model relevancy and effectiveness
- **Data Pipeline Management:**
  - Overseeing data flow into and out of models
  - Ensuring clean, relevant, and timely data
- **Fine-tuning Workflows:**
  - Adjusting model parameters based on feedback
  - Retraining with new datasets
- **Security Patches:**

10.4 LLMOps Implementation Steps

- **Step 1: Select a Foundation Model**
    - Choose between proprietary models (OpenAI's GPT) or open-source options
    - Consider performance requirements vs. cost constraints
- **Step 2: Adapt to Downstream Tasks**
    - Implement prompt engineering techniques
    - Apply fine-tuning for specific applications
    - Incorporate external data through RAG
    - Utilize embeddings for search and recommendations
- **Step 3: Deploy and Maintain**
    - Establish version control and governance
    - Implement monitoring and feedback loops

11. LLMOps Security Best Practices

- **Access Control Implementation:**
  - Role-based access control for model APIs
  - Multi-factor authentication for sensitive operations
  - Strict permission boundaries between environments
- **Data Privacy Protections:**
  - Data encryption at rest and in transit
  - Personally Identifiable Information (PII) detection and redaction
  - Compliance with regulatory frameworks (GDPR, HIPAA, etc.)
- **Prompt Injection Defenses:**
  - Input validation and sanitization
  - Context boundary enforcement

12. LLMOps Development Best Practices

- **Version Control for Prompts:**
  - Maintain all prompts in source control
  - Document prompt changes with detailed commit messages
  - Track prompt performance across versions
- **Testing Frameworks:**
  - Implement automated testing for prompts and flows
  - Create comprehensive test suites for different scenarios
  - Test for edge cases and potential vulnerabilities
- **CI/CD Implementation:**
  - Automate flow deployment with quality gates
  - Implement progressive deployment strategies

13. LLMOps Production Best Practices

- **Load Balancing Strategies:**
  - Distribute traffic across multiple model instances
  - Implement auto-scaling based on demand patterns
  - Optimize resource allocation for cost efficiency
- **Failover Mechanisms:**
  - Design redundant systems for high availability
  - Implement graceful degradation patterns
  - Create fallback responses for service interruptions
- **Caching Implementation:**
  - Cache common responses to reduce latency
  - Implement tiered caching strategies

14. Azure-Specific Implementation Guidance

- **Azure OpenAI Integration:**
  - Leverage Azure OpenAI Service for compliance and security
  - Implement Managed Identity for secure authentication
  - Use Private Endpoints for network isolation
- **Deployment Patterns:**
  - Blue-Green deployments for zero-downtime updates
  - Canary releases for controlled feature rollout
  - Shadow deployments for performance testing
- **Monitoring Setup:**
  - Azure Monitor and Application Insights integration
  - Custom dashboards for LLM-specific metrics

15. Summary and Next Steps

- **Key Takeaways:**
  - LLMOps brings engineering rigor to AI development
  - Our AIOps template provides complete lifecycle management
  - Standardized approach improves quality and accelerates delivery
- **Getting Started:**
  - Clone the template repository
  - Follow the documentation for your specific use case
  - Start with the simplest flow type for your needs
- **Future Roadmap:**
  - Enhanced RAG metrics and evaluation capabilities
  - Additional connectors for data sources

16. Q&A

- **Questions?**
- **Demo Available Upon Request**
- **Contact:** [Your Contact Information]