# Generic LLM Observability Framework

**Status:** In Progress (Design Complete / Dev Pending)

**Target Delivery:** MVP by Dec 31st, 2025

**Tech Stack:** Python (llmops\_core), Azure OpenAI, Langfuse, AWS Secrets Manager

## 1. Executive Summary

We are developing a generic middleware library that acts as the standardized "Gateway" between our internal applications and LLM providers.

While the immediate pilot is the *Payer Knowledge* app, this framework is explicitly designed to be consumed by the future use cases (e.g., Medical Coding, Enrollment, Auth etc.,). This ensures we avoid vendor lock-in, enforce security compliance (PII redaction), and gain granular cost visibility across tenants.

**Core Objectives (MVP):**

1. **Decoupling:** Applications must not interact with LLM Providers directly.
2. **Safety:** No PII (SSN, Member IDs) enters the logging system.
3. **Resilience:** If the observability layer fails, the application **must not** crash.
4. **Regression:** Automated testing capabilities for model upgrades (GPT-4 vs GPT-5).

## 2. Feasibility & Solution Design

We have defined the technical strategy to support the requirements:

* **Connectivity Strategy:** The solution will connect the internal Azure Environment to Langfuse Cloud (Dev Project) using secure API keys.
* **SDK Pattern:** We have selected the **Wrapper Pattern** to intercept Azure OpenAI Python SDK calls.
* **Package Structure:** We have defined the llmops\_core package structure to ensure the Middleware pattern is enforceable.

**Immediate Validation Required (First Dev Task):**

* [ ] Run a "Hello World" connectivity test to Langfuse Cloud.
* [ ] Verify AWS Secrets Manager retrieval from the application runtime.

## 3. Architectural Flow

The framework operates as a "Wrapper" around the LLM calls. It intercepts requests to inject metadata, sanitize inputs, and manage prompts before executing the call.

*(Insert the Architecture Diagram generated in Draw.io here)*

**The Data Pipeline:**

1. **Request:** App initiates call via Client (Zone 1).
2. **Config:** Framework loads Environment keys (Dev vs Prod) securely.
3. **Prompt:** Framework fetches the latest prompt version from Langfuse (cached locally).
4. **Sanitize:** Regex engine strips PII from inputs.
5. **Execution:** Call is sent to Azure OpenAI (Zone 3).
6. **Telemetry:** Trace data is pushed to an **Async Queue** (Non-blocking).
7. **Response:** Clean response returned to the User.

## 4. Key Architectural Principles (The Rules)

|  |  |
| --- | --- |
| **Principle** | **Description** |
| **The "Middle Layer" Mandate** | No application code should import openai directly. All calls must go through our generic framework. |
| **Circuit Breaker Pattern** | **Critical:** Observability is secondary to user experience. If Langfuse times out or errors, the framework must catch the exception, log it locally, and ensure the LLM response is still returned to the user. |
| **Async by Default** | Logging operations must happen in background threads. We cannot add latency to the UI for the sake of telemetry. |
| **Environment Isolation** | Strict separation of data. **Dev** projects for prompt engineering; **Pre-Prod** for Golden Dataset regression; **Prod** for sanitized live traffic. |

## 5. MVP Implementation Plan & Estimates

Estimation Basis: 1 Story Point = 6 Hours of Development.

Target: 17 Points (~102 Hours) by Dec 31st.

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Task** | **Description & Deliverables** | **Est. Points** |
| **1. Architecture** | **Define Framework Specs** | • Define llmops\_core package structure.  • Define metadata schema (payer\_id, session\_id).  • **Deliverable:** Interface Spec. | **3** |
| **2. Infra** | **Langfuse & Secrets Setup** | • Create Langfuse Projects (Dev/Prod).  • Implement AWS Secrets Manager retrieval logic.  • **Deliverable:** Secure Env Config Loader. | **1** |
| **3. Core Dev** | **Develop Middleware Core** | • Initialize Python Package.  • Build Singleton Client with **Circuit Breaker** logic.  • **Deliverable:** client.py and tracing.py. | **3** |
| **4. Security** | **PII Sanitization Engine** | • Build Regex engine to detect SSN, Member IDs, Emails.  • Apply masking *before* logging.  • **Deliverable:** sanitizer.py module. | **2** |
| **5. Integration** | **Integrate Payer App** | • Refactor *Payer Knowledge* to use our generic framework.  • Remove hardcoded OpenAI calls.  • **Crucial:** Inject payer\_id metadata here for cost tracking.  • **Deliverable:** Updated App Code. | **2** |
| **6. Prompts** | **Prompt Migration** | • Extract hardcoded prompts from Python code.  • Upload to Langfuse "Prompt Management".  • Implement local caching in SDK. | **2** |
| **7. Regression** | **Golden Dataset Suite** | • Create "Golden Set" (50-100 samples).  • Build script to run batch tests (GPT-4 vs GPT-5).  • **Deliverable:** Regression Python Script. | **3** |
| **8. Monitoring** | **Dashboards & Alerts** | • Configure Dashboards (Cost per Payer, Latency).  • Set Alert Rules (Error Rate > 5%). | **1** |

## 6. Definition of Done (DoD) for Dec 31st

* [ ] **Zero Hardcoded Prompts:** All system prompts are fetched via the SDK.
* [ ] **Resilience Verified:** Disconnecting the Langfuse API key does *not* crash the application.
* [ ] **Sanitization Active:** No SSN or PII is visible in the Langfuse Dashboard traces.
* [ ] **Cost Visibility:** We can filter costs by payer\_id (Tenant) in the dashboard. (Enabled by Task 5).
* [ ] **Regression Ready:** A single script execution runs the "Golden Dataset" against a new model candidate.

## 7. Out of Scope (Deferred to 2026)

* **Advanced Cost Logic:** Budget enforcement/quotas (Phase 3 in original plan).
* **A/B Testing Framework:** Automated traffic routing.
* **Advanced Evaluators:** "LLM-as-a-Judge" for complex clinical accuracy.
* **CI/CD Automation:** Blocking deployment pipelines automatically based on scores.