LLM Observability

Why do we need LLM Observability?

GenAI applications are great, they answer like how a human does. But how do you know if GPT isn't being "too creative" to you when results from the LLM shows "Company finances are facing issues due to insufficient sun coverage"?

As the scope of GenAI apps broaden, the vulnerability expands, and since LLM outputs are non-deterministic, a setup that once worked isn't guaranteed to always work. Here's an example of comparing the reasons why an LLM prompt fails vs why a RAG application fails.

What could go wrong in the configuration?

LLM prompts

- Suboptimal model parameters
    - Temperature too high / tokens too small
- Uninformative System prompts

RAG

- Indexing
    - The data wasn't chunked with the right size, information is sparse yet the window is small.
    - Wrong distance was used. Used Euclidean distance instead of cosine
    - Dimension was too small / too large
- Retrieval
    - Top K too big, too much irrelevant context fetched
    - Top K too small, not enough relevant context to generate result
    - Filter misused
- **And everything in LLM Prompts**

Although observability does not magically solve all problems, it gives us a good chance to figure out what might have gone wrong. LLM Observability provides methodologies to help developers better understand LLM applications, model performances, biases, and can help resolve issues before they reach the end users.

What are common issues and how observability helps?

Observability helps understanding in many ways, from performance bottlenecks to error detection, security and debugging. Here's a list of common questions we might ask ourselves and how observability may come in handy.

1. How long does it take to generate an answer?
    - Monitor LLM response times and database query times helps identify potential bottlenecks of the application.
2. Is the context retrieved from the Vector Database relevant?
    - Logging database query and results retrieved helps identify better performing queries.
    - Can assist on chunk size configuration based on retrieved results.
3. How many tokens are used in a call?
    - Monitor token usage can help determine the cost of each LLM call.
4. How much better/worse is my new configuration setup doing?
    - Parameter monitoring and response logging helps compare the performance of different models and model configurations.
5. How is the GenAI application performing overall?
    - Tracing stages of the application and evaluation helps identify the performance of the application
6. What are users asking?
    - Logging and analyzing user prompts help understand user needs and can help evaluate if optimizations can be introduced to reduce costs.
    - Helps identify security vulnerabilities by monitoring malicious attempts and help proactively respond to mitigate threats.

What should be tracked?

GenAI applications involve components chained together. Depending on the use case, there are events and input/output parameters that we want to capture and analyze.

A list of components to consider:

Vector Database metadata

- **Vector dimension**: The vector dimension used to in the vector database
- **Distance function**: The way two vectors are compared in the vector database

Vector Indexing parameters

- **Chunk configuration**: How a chunk is configured, including the size of the chunk, the unit of chunks, etc. This affects information density in a chunk.

Vector Query parameters

- **Query**: The query used to retrieve context from the Vector Database
- **Top K**: The maximum number of vectors to retrieve from the Vector Database

Prompt templates

- **System prompt**: The prompt to be used throughout the application
- **Template**: The template used to construct a prompt. Prompts work differently in different models and LLM providers

LLM request metadata

- **Prompt**: The input sent to the LLM model from each end-user, combined with the template
- **Model name**: The LLM model used for generation, which affects the capability of the application
- **Tokens**: The number of tokens limit for a single request
- **Temperature**: The parameter for setting the creativity and randomness of the model
- **Top P**: The range of selection of words, the smaller the value the narrower the word selection is sampled from.

LLM response metadata

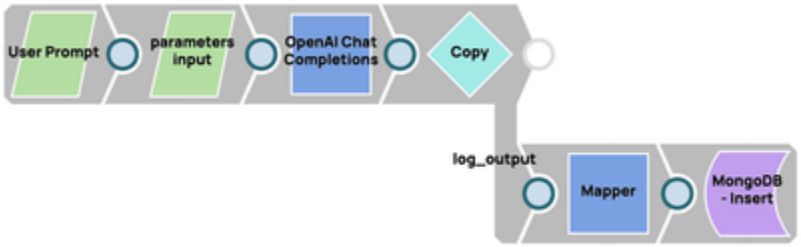
- **Tokens**: The number of tokens used in input and output generation, affects costs
- **Request details**: May include information such as guardrails, id of the request, etc.

Execution Metrics

- **Execution time**: Time taken to process individual requests

Pipeline examples

Logging a Chat completions pipeline



We're using MongoDB to store model parameters and LLM responses as JSON documents for easy processing.

Logging a RAG pipeline

In this case, we're storing parameters to the RAG system (Agent Retrieve in this case) and the model. We're using JSON Generator Snaps to parameterize all input parameters to the RAG system and the LLM models. We then concat the response from the Vector Database, LLM model, and the parameters we provided for the requests.