Observability for Large Language Models

By Javier C.A, Adrián M.F, Sergio M.A, and Saúl M.F

Observability is a new and upcoming concept, especially with the rise of LLMs in our day-to-day lives. Therefore, we can expect to encounter it at some point in our careers, and thus we have the need to understand the concept and how to make use of it in our productivity frameworks. This document/presentation aims to present what observability is, give a brief introduction into LLMs, how the two are related and introduce a set of tools, as well as show the idea of observability-driven development.

Observability

First, we must define the concept itself. However, due to its recent nature, its definition varies massively depending on each company who sells a product about it. However, as Phillip Carter did on the podcast, a great way to understand it is by rooting the explanation on the problem it solves:

Imagine you are notified your users have an issue with your application. To solve this, you typically would debug, find in the code the point/s where the issue stems from and then decide on the approach to solve it, generally modifying the source code. But what if you are in a situation where the source code is either too costly to change or simply unmodifiable?

In these cases, like for example when using closed off systems, you can use observability to spot the point/s of failure. Take for example, a more complex application where you have several services which you do not own, but rather use, and the issue arises from the interaction between the systems you provide and those you have hired, ie a latency issue. However, when you contact the services providers, you are told their latency is just fine. Observability tools take telemetry at the different stages of your system, to “observe” and analyze the process runtime. With this, they can help spot that, maybe you are calling at four different points the same service, thus multiplying by four the latency.

Observability helps to identify the state of the system via external outputs like logs and metrics and therefore reproduce and fix issues that cannot simply be debugged.

Large Language Models

Large Language Models are computer programs that understand and generate human language. They are trained on huge amounts of text data from books, articles, and websites.

This training helps them learn patterns in language so they can complete sentences, answer questions, or even write paragraphs that sound natural.

What is Behind LLMs?

At the heart of most LLMs is a neural network architecture called the Transformer. The Transformer uses a mechanism known as "attention." This means that the model can focus on the most important parts of a sentence or paragraph when trying to understand or generate text. It is one of the main reasons why these models work so well with long texts.

These models are built using deep learning techniques. This means that they have many layers of neurons, which help them learn complex representations of language. Each layer processes the input data in a way that builds up a deeper understanding of the text.

Some of the most famous LLMs include GPT-4 from OpenAI or LLaMA from Meta.

Training of LLMs

LLMs are trained in two main phases: pre-training and fine-tuning.

Pre-training

During pre-training, the model is exposed to high volumes of text to learn general linguistic patterns. This process allows to develop a basic understanding of language, ranging from syntax to semantic context.

This learning is based on the probability and context of the words in the language.

It relies on large amounts of unstructured data, such as books and websites. However, it does not specialise it for specific tasks.

Fine-Tuning

Fine-tuning is the process of taking a pre-trained model and adjusting it to perform a specific task in a more precise manner.

By using fine-tuning, companies can improve the performance of LLMs in specific areas, which makes the models more useful and relevant for different applications.

Where are LLMs Hosted?

Because LLMs are very large and need a lot of computing power, they are hosted in the cloud. Companies like Google, Microsoft, and Amazon have cloud services that run these models. They also use specialized hardware like GPUs and TPUs (tensor processing units) that are designed to handle the heavy computations required by these models.

Prompt Engineering

Finally, an important part of working with LLMs is prompt engineering. This is the process of designing clear and precise questions or instructions, called prompts, that guide the model to give the best answer.

- For example, asking “What are the main challenges of training LLMs?” is clearer than just saying “Tell me about LLMs.” The better the prompt, the better the response.

A well-designed prompt reduces ambiguity and ensures that the response is focused on the intended topic. By refining your prompts, you can tailor the output of an LLM to better suit specific tasks, whether that's generating creative content, answering technical questions, or summarizing complex information.

Observability with LLMs

Introduction

When we think about debugging software, we usually imagine stepping through code, setting breakpoints, and testing different inputs. But with Large Language Models, things work very differently. We can't just 'reproduce' an issue like in traditional systems. So, if the model gives an incorrect answer, how do we even know why it happened? Well, that's where observability comes in.

1. Why Observability Matters for LLMs

Observability helps us understand how a system behaves without changing it. This is crucial for LLMs because they are unpredictable. By giving the same prompt twice, you will most likely get two different answers. Unlike traditional UI-based applications, where user interactions are limited to buttons and forms, in LLMs, users can input anything they want, making traditional testing impossible.

And the biggest challenge? You usually can't modify the model itself. Fine-tuning is expensive and time-consuming. So, if we want to improve an LLM-based system, we need observability to figure out what's going wrong without touching the model.

2. Key Challenges in Observability for LLMs

Now you may wonder what makes observability for LLMs difficult? Well, there are three key challenges.

Complex Decision Chains

When you ask an LLM a question, the answer doesn't just come from the model itself. There are a lot of upstream (or decisions) like retrieving relevant context, structuring the prompt, and applying business rules. The bad thing is that If any of those steps go wrong, the final output will be incorrect, even if the model is technically working fine.

Mostly Correct" Responses

Another issue is that LLMs often produce outputs that are mostly correct but contain small, critical errors. Let's say you ask a chatbot for an address, and it gets everything right except the street number. How do we track and fix these partial errors without manually reviewing every response?

Latency Issues

And then there's performance. LLMs are known to be slow, but is the delay really coming from the model? A great real-world example is what my partner said

before, a company that thought OpenAI's API was slow, but after adding observability, they found out they were making too many unnecessary network calls before even hitting the LLM. They fixed that and response times got much better.

3. How Observability Helps

So how do we solve these problems? Observability in LLMs focuses on three main areas:

Capturing Inputs & Outputs

We log what users are asking and what responses they get. This helps us identify patterns, like whether people are phrasing prompts in ways the model doesn't handle well.

Tracking Upstream & Downstream Decisions

We monitor the entire flow, what information we gather before calling the model and what happens to the answer afterward. If the final output is bad, observability helps us figure out whether the problem started before or after the model was called.

Optimizing Performance

And finally, we analyze latency. Observability lets us find bottlenecks, whether it's the LLM itself, an external API, or even our own database slowing things down.

Tools for Observability & Observability-Driven Development for LLMs

1. Observability Tools in Action

Well, my colleagues explained before what observability is and its relation to LLMs. I will now focus on the tools that enable observability.

When working with Large Language Models, traditional debugging isn't enough—especially because LLMs are non-deterministic, so here's how we can practically apply observability tools to get deep insights:

Structured Logging:

One of the first steps is to capture a detailed, structured log. By logging both the raw user input and the generated prompt, along with metadata like decision points and context, you can create a foundation for debugging. Even if the model returns "mostly correct" answers, these logs help pinpoint if errors stem from the prompt generation or from upstream decisions.

Distributed Tracing with OpenTelemetry:

There are tools like OpenTelemetry, which let you trace the full journey of a request through your system. Imagine each request as a chain of "spans" where every function call, from the initial user interaction to the final API response, is recorded. This approach can reveal if, for example, multiple redundant calls are unintentionally adding latency, as it has already happened in real-world scenarios.

High-Cardinality Data Analysis (Honeycomb):

As you know, LLMs generate unique outputs and receive a wide variety of inputs, which can be challenging for traditional monitoring systems.

Platforms like Honeycomb excel in analysing such high-cardinality data. They help visualize patterns and correlations—whether it's spotting that certain types of prompts consistently lead to errors or detecting that upstream services are causing unexpected delays.

Other Tools:

Well, these tools were not explicitly mentioned in the podcast but there are another observability tools such as Prometheus, Grafana, and even experiment-tracking platforms like Weights & Biases that can complement your setup. These systems provide dashboards and alerts to continuously monitor latency, error rates, and overall system health, ensuring that no critical performance issue goes unnoticed.

2. Observability-Driven Development (ODD)

Observability isn't just for reactive debugging—it's a core part of the development process. You can integrate it into your development cycle following these guidelines:

Iterative Improvement Based on Real Data:

Since LLMs cannot be thoroughly tested with synthetic inputs, observability-driven development means you release a minimally viable product and iterate based on actual user interactions. Your telemetry data can guide you in refining prompts, adjusting parameters, or even rethinking parts of your architecture when you spot recurring issues.

Feedback Loop for Product Evolution:

By continuously logging inputs, outputs, and error patterns, you build a feedback loop. For instance, if you notice that "mostly correct" responses have systematic errors—like missing a key piece of structured data—this knowledge can drive targeted prompt engineering improvements or prompt versioning.

Balancing Reliability and Innovation:

Observability-driven development helps you strike the right balance. On one hand, you need to ensure that performance metrics such as latency and error rates meet user expectations. On the other hand, observability reveals unexpected use cases and innovative interactions that you might not have considered. Using these insights, you can adjust your development priorities to both fix issues and explore new features that enhance user experience.