Observabilidad aplicada a modelos largos de lenguaje (LLM)

Introducción: ¿Qué es la observabilidad?

La observabilidad es la capacidad de comprender en qué estado se encuentra un sistema basándose en los resultados externos (*output*) que arroja dados unos datos de entrada. La observabilidad en sí misma es medible, por lo que se dice que un sistema es más o menos observable en función de la percepción que tienen tanto usuarios como desarrolladores sobre el funcionamiento del mismo. Esta medida también puede usarse para comprobar cuán eficiente es un sistema a ojos de un usuario, que probablemente no tiene ni la más remota idea de cómo funciona por dentro.

La observabilidad como tal puede aplicarse sobre diversos ámbitos, no solo sobre la informática. Por ejemplo, en la medicina, los médicos pueden usar las mediciones realizadas para diagnosticar según qué enfermedades sin necesitar comprobar el funcionamiento interno del cuerpo humano.

¿Qué relación tienen los LLMs con la observabilidad?

Dicho lo cual, la pregunta que nos ocupa hoy es: "¿Cómo puedo aplicar la observabilidad a los modelos largos de lenguaje (LLMs)?"

Es sensato preguntarse algo así, ya que recientemente se ha visto el auge de las inteligencias artificiales, los asistentes que nos facilitan la vida tomando decisiones y realizando cálculos a partir de una entrada dada, lo que nosotros como usuarios les consultamos. Hoy en día es difícil encontrar un sitio web que no tenga un asistente integrado.

Desafíos relacionados con la observabilidad

Poniéndonos teóricos, estas inteligencias artificiales producen resultados parecidos a una función impura, es decir, operaciones que con datos de entrada idénticos pueden arrojar resultados variados. Esta naturaleza probabilística puede complicar la tarea de corregir errores, más aún si son IAs generativas, ya que vemos que aún les cuesta generar imágenes de ciertas cosas.

Aparte del propio contenido generado por el LLM, otro factor a tener en cuenta es la latencia, el tiempo de respuesta tras el cual el usuario finalmente recibe la respuesta. Esto está fuertemente relacionado con términos de usabilidad, ya que

de nada sirve que un LLM sea capaz de resolver cálculos complejos y razonamientos imposibles si tarda una eternidad en generar su respuesta.

Un LLM que arroja resultados poco fiables, tarda en responder, o incluso no responde lo que se le pregunta, son causas suficientes para que un usuario pierda su confianza en la aplicación, lo que puede afectar negativamente a su reputación.

Enfoques para solucionar errores

Prácticamente desde que se introdujeron los LLMs y IAs al mercado tecnológico, las empresas y desarrolladores han implantado diversas medidas para encontrar los puntos críticos y solventar los problemas que pueden ocurrir.

- La más evidente a priori es la de **monitorear** las consultas enviadas al LLM. Haciendo un seguimiento junto con el registro de los datos de entrada y salida junto con el tiempo de respuesta puede ayudar a detectar anomalías.
- Otra solución a este dilema es afrontar un ciclo de desarrollo enfocado en la observabilidad: **Observability Driven Development** (ODD). Esta forma de desarrollar sistemas informáticos fomenta que el desarrollo se realice teniendo en cuenta aspectos relacionados con la observabilidad en fases más tempranas del desarrollo, en lugar de posponerlo hacia el final del desarrollo. De este modo, el desarrollo se complementa con métricas y herramientas que facilitan la detección de errores.
- Para finalizar, también es posible emplear técnicas de **logging**, almacenando resultados relacionados con los datos de entrada y salida y etiquetas clave que puedan facilitar análisis posteriores.

Con estas herramientas, se podría facilitar la tarea de encontrar y resolver errores que de otra forma se tardaría un montón de tiempo en resolver. Como todo, un diseño (ODD) eficaz y el uso continuado de herramientas puede favorecer enormemente no solo al rendimiento y observabilidad del sistema, sino la productividad del equipo de desarrollo y el grado de satisfacción del usuario.

Desarrollo enfocado en la observabilidad (ODD)

Como hemos mencionado antes, ODD es un enfoque de desarrollo de software en el que la observabilidad juega un papel clave desde el principio, en vez de limitarse a ser una herramienta de monitoreo y depuración. Este enfoque ayuda a que la observabilidad no se posponga hasta etapas tardías del proceso software, sino que se tome en cuenta desde el principio.

Este tipo de desarrollo tiene una serie de principios que lo caracterizan, además de servir como guía a la hora de implementarlo en un proceso de desarrollo:

1. **Diseño proactivo**. En lugar de esperar a que los fallos se produzcan, estos se anticipan mediante el uso de métricas, logs y trazas desde el principio.
2. Se emplean **herramientas de observabilidad** tales como OpenTelemetry, Prometheus o Honeycomb para llevar a cabo estas métricas, además de relacionarlas con el comportamiento esperado del sistema.
3. **Desarrollo guiado por datos en tiempo real.** En cualquier etapa del desarrollo es posible llevar a cabo experimentos controlados que permitan medir el impacto en la experiencia del usuario; relacionado con usabilidad.
4. **Reducción de la depuración a ciegas**. Cuando el sistema falla, los 3 principios anteriores ya han proporcionado suficientes evidencias y logs para que los desarrolladores sean capaces de identificar la raíz del problema y rectificarlo sin necesidad de reproducirlo manualmente.

En otras palabras, un ODD permite hacer registros (*logging*) de las interacciones del usuario con el sistema (en nuestro caso el LLM), recibiendo además datos adicionales como el tiempo de respuesta, el grado de satisfacción del usuario según su respuesta, el número de intentos para obtener una respuesta satisfactoria... De realizarse un reporte sobre el mal funcionamiento de un LLM, los desarrolladores pueden llevar a cabo los cambios pertinentes según las métricas obtenidas, sin necesidad de reproducir los pasos del usuario anterior.

En definitiva, un ODD favorece a una mayor estabilidad del sistema, satisfacción por parte del usuario, y desarrollo basado en datos fiables en lugar de suposiciones.

Diseño y optimización de LLMs

Perspectivas

A la hora de integrar un modelo de lenguaje en un sistema, existen dos formas principales de entenderlos:

- **Perspectiva de caja negra**: Nos enfocamos únicamente en los datos de entrada y salida del modelo, sin conocer el funcionamiento interno, basándose únicamente en la experimentación empírica. Se toma el LLM como un mecanismo misterioso que funciona, pero no sabemos cómo.

- **Perspectiva de transformadores y mecanismos de atención**: Permite entender cómo el modelo procesa la información, pudiendo optimizar su rendimiento por medio de ajustes y optimizaciones sobre su funcionamiento interno.

Podemos deducir que la segunda de ellas brinda mejores resultados que la primera. Comprender cómo funciona un LLM por dentro, modificarlo a nuestro

gusto y optimizarlo para sacar el máximo provecho puede resultar en un rendimiento mayor, a diferencia de la prueba y error.

Técnicas de optimización

Otro aspecto clave en la optimización de LLMs posterior a su diseño es el empleo de dos técnicas:

- **Fine-tuning**: Entrenar un modelo ya existente con un conjunto de datos diferente, enfocándolo en un ámbito más especializado que permita personalizarlo según las necesidades.
- **Prompt engineering**: Sin modificar el funcionamiento interno de un modelo (e incluso sin siquiera saber cómo funciona) ajustar los datos de entrada con el objetivo de sacar el máximo provecho y obtener resultados mejores. Algunos modelos dan más importancia a unas palabras concretas que otras, arrojan mejores resultados según la estructura de la petición, etc.

Relación entre técnicas de optimización y perspectivas

Dicho esto, se puede establecer una correlación entre las dos mecánicas anteriores y las perspectivas que hemos detallado antes:

- **Fine-tuning y perspectiva de transformadores y mecanismos de atención**: Ajustar un modelo de lenguaje mediante el fine-tuning requiere antes comprender cómo funciona internamente para proporcionarle el conjunto de datos correcto para su entreno, además de optimizarlo con el fin de evitar anomalías y obtener el mejor resultado posible ajustado al dominio requerido.
    - **Ejemplo**: ChatGPT emplea un modelo más generalizado, mientras que Copilot está especializado en asistir a programadores durante el desarrollo. De este modo, Copilot realiza un mejor trabajo relacionado con cuestiones de programación, mientras que con ChatGPT debería emplearse prompt engineering para obtener resultados aproximados.
- **Prompt engineering y perspectiva de caja negra**: Dado un modelo que no se puede modificar y tampoco saber cómo funciona por dentro, una técnica útil es hacer uso del prompt engineering, basándose en la prueba y error para encontrar patrones que puedan arrojar mejores respuestas.
    - Ejemplo: El grueso de usuarios finales de un LLM no conocen cómo funciona por dentro, pero sí pueden optimizar sus datos de entrada para intentar obtener mejores resultados.