Apartado Teórico - Francisco José Morón Reyes

Diferencias entre ‘completion’ y ‘chat’ models

Los modelos de ‘completions’ están entrenados y diseñados para generar texto basándose en un prompt. Este tipo de modelos toman como entrada una instrucción, que suele ser el prompt, y predicen la continuación más probable del texto. Son modelos de un sólo turno, caracterizados por su gran capacidad de personalización a la hora de poder ser *finetuneados*. Son muy buenos generando contenidos, realizando resúmenes de texto, clasificaciones etc…

Por el contrario, los modelos de ‘chat’ están diseñados para trabajar específicamente en escenarios de múltiples turnos, es decir, conversaciones. En lugar de trabajar con un solo prompt, este tipo de modelos aceptan un historial de conversación representado por diferentes roles (user, assistant, system…). El modelo está preparado para procesar el historial de conversación y generar una respuesta que tenga en cuenta el contexto.

Para los modelos de ‘chat’ tenemos un diseño enfocado a ‘multi-turn’ conversation. Aquí no estamos trabajando con un solo prompt, sino que estos modelos aceptan un historial de conversaciones que contienen mensajes procedentes de diferentes roles (user, assistant, system…). Son buenos en contextos de aplicaciones *chatbot*.

Un punto importante a mencionar es que los modelos de ‘chat’ no pueden ser usados, a día de hoy, como modelos base para *finetuning*.

¿Cómo forzar a que el chatbot responda 'si' o 'no'?¿Cómo parsear la salida para que siga un formato determinado?

Condicionar la salida estocástica de los LLMs siempre es complejo, no obstante existen ciertos métodos que pueden condicionar la salida final de estos modelos.

Opción 1: Sistema de reglas basadas en código

Si, por forzar la salida del chatbot, aceptamos posibles manipulaciones tras la respuesta del modelo, una primera opción sería post-procesar la respuesta mediante una serie de reglas y responder siempre sí o no o cualquier otra plantilla.

Esta es una estrategia compleja, ya que, a priori, las opciones de respuesta de un LLM son tantas como posibles combinaciones hay en el lenguaje y eso es inasumible.

Opción 2: Prompting

Otra enfoque que podríamos adoptar sería la inclusión de un prompt que permitiese 'forzar' la generación textual.

Esta estrategia es bastante común en la actualidad y aporta resultados relativamente razonables. En mi experiencia profesional los mejores prompts son aquellos más complejos y exhaustivos, sin embargo los modelos con menos parámetros (gemma-2b-it por ejemplo) entienden peor este tipo de prompts y suelen interpretarlos de forma vaga y poco concisa.

Por otra parte, condicionar la salida mediante un prompt puede volver a tener problemas de abarcabilidad, al hacer difícil adaptar el prompt a todas las situaciones posibles.

Opción 3: Fine-tuning de un modelo de completion

Es posible, por ejemplo, tomar un modelo como llama-8b-it y crear un *dataset* sintético de preguntas y respuestas que siga un formato determinado. De esta forma, enseñaremos al modelo a responder a nuestras preguntas siguiendo un estilo que lo marca desde el entrenamiento. Es una opción que funciona pero no es posible aplicarla a modelos de chat.

Opción 4: Utilizar librerías de generación controlada u otras herramientas por el estilo.

Es posible forzar al modelo a producir una salida textual determinada. Por esto entendemos 'Generación controlada' o 'generación guiada' y hay algunas herramientas que nos permiten hacer esto con buena eficacia.

Ejemplos de esto lo encontramos en librerías como *Outlines* o *Instructor*. Ambas se sirven de *Pydantic*, ampliamente utilizada en Python para validación y gestión de esquemas de datos. Con ayuda de estas librerías podemos definir, por ejemplo, un formato de salida estilo *JSON*, lo que ayudaría enormemente al modelo a saber qué definir. También podemos, por ejemplo, condicionar al modelo para elegir una entre diferentes categorías de clasificación o responder siempre acorde a un esquema de datos específico.

Existen otras posibilidades dentro de las APIs privadas de algunos de los principales distribuidores de IA Generativa, como por ejemplo (y aplicado a chat models), las *function call* de OpenAI, que pueden condicionar la salida de un modelo para que siga un determinado formato.

Con respecto a la última pregunta dentro de este apartado, si tuviera que decidir una forma de parsear la salida de un modelo generativo intentaría siempre utilizar la opción 4, no obstante, creo que esta opción 'abstrae' demasiado el modelo, dificultando el acceso a los parámetros de generación y, por consiguiente, quitando opciones de personalización que

pueden otorgar alguna desventaja. También es posible usar alguna de las otras opciones para ayudar a parsear la salida.

Ventajas e inconvenientes de RAG vs *fine-tunning*

Primero expliquemos de manera breve en qué consiste cada concepto.

Fine-tunning se refiere al proceso de 'ajuste' de un modelo pre-entrenado con un conjunto de datos específicos de una tarea para mejorar su rendimiento en esa tarea en particular. Este proceso especializa al modelo en un dominio específico, lo que suele conllevar una mejora de sus capacidades.

Retrieval Augmented Generation (RAG) se refiere a la combinación de un sistema de Recuperación de la Información con un Modelo de IA Generativa. En este proceso, primero se recupera información relevante de una base de datos y, proporcionando esa información como contexto al modelo, se le pide que realice una respuesta generativa que, generalmente, es más precisa y, sobre todo, rastreable.

No obstante, cada enfoque tiene sus ventajas y sus desventajas:

Fine-tuning

- Ventajas:
    - Adaptabilidad: Puede adaptarse a una gama amplia de tareas y dominios.
    - Menor latencia: ideal para inferencias rápidas.
    - Escalabilidad: Puede escalarse y adaptarse con facilidad si se tienen datos de buena calidad.

- Desventajas:
    - Volúmenes de datos: Este enfoque requiere ingentes cantidades de datos que deben tener buena calidad y ser representativos.
    - Datos etiquetados: Para hacer un buen entrenamiento, es imprescindible que las etiquetas también sean de calidad.
    - Coste computacional: Finetunear los modelos en la actualidad es muy complejo y costoso en lo que a términos de costes computacionales se refiere. Hay que recurrir a clústeres o instalaciones preparadas porque los equipos en su local no son suficientes.

Rag

- Ventajas:
    - Respuestas más ricas: Proporcionar un contexto ayuda a que el modelo generativo sea capaz de responder de forma más completa.
    - Menor propensión a la alucinación: Que el modelo pueda apoyarse en información clara y concisa reduce la capacidad de alucinación.

- Desventajas:
    - Mayor latencia: El proceso complejo de RAG suele traer consigo tiempos de recuperación y generación mas lentos.
    - Escalabilidad de la BBDD: Grandes volúmenes documentales pueden ser un desafío para las bases de datos vectoriales.

¿Cómo evaluar el desempeño de un bot de Q&A? ¿Cómo evaluar el desempeño de un RAG?

Partamos de la base de que siempre es difícil evaluar el desempeño de los modelos generativos. No he trabajado exhaustivamente con bots de QA, aunque sí he hecho trabajos en NLP sobre QA y, generalmente, puedes recurrir a métricas de NLP para medir la calidad de las respuestas, como por ejemplo EM (*Exact Match*) o *F1 Score* sin embargo necesitas tener una respuesta correcta con la que poder comparar los resultados de la generación.

Para no entrar a detalles de las métricas que podemos usar, voy a partir del hecho de que estamos en un paradigma en el que los usuarios usan el bot de QA o el sistema RAG y necesitamos saber si las respuestas generadas son de buena calidad o no.

Existe un paradigma denominado *LLM-as-a-judge* que consiste en utilizar un LLM (típicamente GPT-4) para analizar las respuestas de otro LLM y valorarlas. En este sistema, podemos pedir a un generativo que, dada la pregunta, el contexto o conversación, y la respuesta, determine si la respuesta es verdaderamente útil para la pregunta.

En mi experiencia personal esta técnica ha resultado ser bastante útil, y no requiere de tener datos etiquetados ni recurrir a métricas de superposición de N-gramas para comparar la calidad de la respuesta generada.

Como mención final, decir que en un RAG también tenemos que evaluar la capacidad de *retrieval* del sistema. Existen métricas apropiadas para ello, como NDCG, pero una vez más necesitamos de datos etiquetados.

Generalmente, en un sistema RAG utilizas una base de datos vectorial que ha indexado embeddings procedentes de un determinado modelo. Aquí cobran importancia los conceptos de Bi-encoder y Cross-encoder, ya que son la tipología de modelos utilizados para generar embeddings. Los bi-encoders utilizan dos *encoders* separados, uno para la *query* y otro para los documentos ‘candidatos’, mientras que en el cross-encoder la *query* y el documento se procesan juntos en el mismo *encoder*. Cuento esto porque, generalmente usas un bi-encoder para generar los embeddings de la base de datos (son más rápidos que los cross-encoders) y, sirviéndote de un cross-encoder puedes hacer ranking en el top-k de documentos recuperados. Es difícil evaluar el desempeño de la recuperación de la información, pero el re-ranking puede ayudarte a ver cómo de relevantes eran realmente los documentos recuperados ya que un cross-encoder captura mejor las similitudes entre una query y un conjunto de documentos.

Aplicaciones del apartado técnico a nuestro caso de uso

En esta sección aplicaremos los conceptos y estrategias que hemos discutido anteriormente a nuestro caso de uso: desarrollar un chatbot basado en RAG adaptado a nuestras necesidades.

Modelos de ‘Completion’ y ‘chat’

Para nuestro caso de uso, estamos utilizando un modelo de *completion* como base, que es Meta-Llama-3-8b-Instruct, pero adaptándolo a un flujo de chat, garantizando que el diseño pueda interactuar en contextos multi-turno. Para la implementación hemos usado *langchain*, que provee de varias herramientas para que el modelo siempre pueda acceder al historial de conversación, mantener memoria sobre él y generar respuestas relevantes.

Métodos para Condicionar la Salida del Chatbot

En nuestro código no estamos usando ningún método en específico para controlar la respuesta del chatbot mas allá de orientarlo a responder preguntas basadas en un contexto mediante nuestra Prompt. Podríamos haber indagado más en el tema y adoptar una estructura de respuesta determinada pero esto ha quedado fuera del ámbito por el momento.

RAG vs Finetuning

Ya hemos cubierto los puntos fuertes y débiles de cada enfoque en apartados anteriores. Nostros hemos usado un enfoque RAG sobre la base de datos vectorial ChromaDB. Para indexar los documentos hemos utilizado el modelo de embeddings BGE-M3 ya que en un principio queríamos adaptar un enfoque multilingüe y la buena calidad de este modelo queda reflejada por los *papers*, aunque podríamos haber recurrido a cualquiera que esté bien representado en el *Leaderboard de MTEB*.

Evaluación del Desempeño

De igual forma, no estamos aplicando ningún sistema de evaluación a nuestro chatbot. Lo más sencillo habría sido usar, por ejemplo, el propio Meta-Llama-3-8b-Instruct para, dada una prompt de evaluación, otorgar una puntuación a cada una de nuestras conversaciones y así comprobar la calidad de la respuesta que estamos generando.

Bibliografía

Siddiqi, S. (08-04-2024). OpenAI GPT Completions vs. Chat Completions: A Comprehensive Comparison. Medium. https://medium.com/@sherazsiddiqidotcom/openai-gpt-completions-vs-chat-completions-a-comprehensive-comparison-a3a17cf3466e.

OpenAI. Function calling. https://platform.openai.com/docs/guides/function-calling.

Outlines. Outlines Documentation. https://outlines-dev.github.io/outlines/.

Use Instructor. Getting Started. https://python.useinstructor.com/#getting-started.

Srivatsa, H. (2024). Fine-tuning versus RAG in Generative AI Applications Architecture. Medium. https://harsha-srivatsa.medium.com/fine-tuning-versus-rag-in-generative-ai-applications-architecture-d54ca6d2acb8

Mecklenburg, N., Lin, Y., Li, X., Holstein, D., Nunes, L., Malvar, S., Silva, B., Chandra, R., Aski, V., Yannam, P. K. R., Aktas, T., & Hendry, T. (2024). Injecting new knowledge into large language models via supervised fine-tuning. arXiv. https://doi.org/10.48550/arXiv.2404.00213

Li, X., Yu, P., Zhou, C., Schick, T., Levy, O., Zettlemoyer, L., Weston, J., & Lewis, M. (2024). Self-Alignment with Instruction Backtranslation. arXiv. https://doi.org/10.48550/arXiv.2308.06259