eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Ingeniería de Computadores

Hau izenburua da/Título del trabajo

Egilearen izen-abizenak / Nombre y apellidos de la o el autor

Dirección

Zuzendaria 1 / Director(a) 1

Zuzendaria 2 / Director(a) 2

9 de abril de 2025

Esker onak / Agradecimientos

Eskerrak eman nahi izanez gero, hemen idatzi testua. En caso de querer añadir agradecimientos, escribir aquí el texto.

Atal hau nahi ez baduzu, *main.tex* fitxategian komentatu lerro hori. En caso de no querer este apartado, comentalo en el fichero *main.tex*.

Laburpena / Resumen

Idatzi hemen laburpena. Escribe aquí el resumen.

Índice de contenidos

Índice de figuras

Índice de tablas

Lista de Algoritmos

CAPÍTULO 1

Introducción

LLMs han avanzado mucho, las apps que usan gen IA están surgiendo blah blah blah.

Los agentes ya hace mucho, pero ahora se están popularizando agentes LLM. Para esto están surgiendo frameworks blah blah blah.

Además se suele trabajar con ajustes de modelos, sistemas RAG para sacarle más provecho a las aplicaciones.

En este contexto LKS Next quiere investigar sobre las diferentes aquitecturas de agentes y sus aplicaciones. Para ello se dispone del escenario de onboarding.

a

CAPÍTULO 2

Antecedentes

Resumen de diferentes partes: - Librerías utilizadas?? ->LangChain, LangGraph, PgVector

- Ajuste de agentes ->LoRa? ->comentar lo de los agentes instruct. - El protocolo MCP - El estado del arte en arquitecturas de agentes LLM y sistemas RAG - - El estado del arte en agentes integrados a proyectos software + contexto de onboarding quizás en la introducción. - Redes neuronales?

2. Agentes LLM (Fundamentos y funcionamiento básico)

¿Qué son los agentes LLM? 2.1. Modelos LLM 2.2. Interacción con herramientas 2.3. Abstracciones en frameworks

2.1. Agentes LLM

Los agentes de Inteligencia Artificial son programas informáticos que implementan modelos computacionales para ejecutar diversas funciones específicas del contexto en el que se aplican. Tras siete décadas y media de investigación, los esfuerzos en el campo se han focalizado en agentes basados en Grandes Modelos de Lenguaje (LLM).

2.1.1. Modelos LLM

Los LLM son redes neuronales especializadas en el procesamiento del lenguaje natural, basados en la arquitectura Transformer. Esta arquitectura se fundamenta en el mecanismo de atención, el cual transforma la representación del texto para incorporar información contextual de manera escalable al hardware.

Para comprender el funcionamiento de estos agentes, resulta imprescindible asimilar previamente conceptos como la tokenización y las representaciones vectoriales del lenguaje.

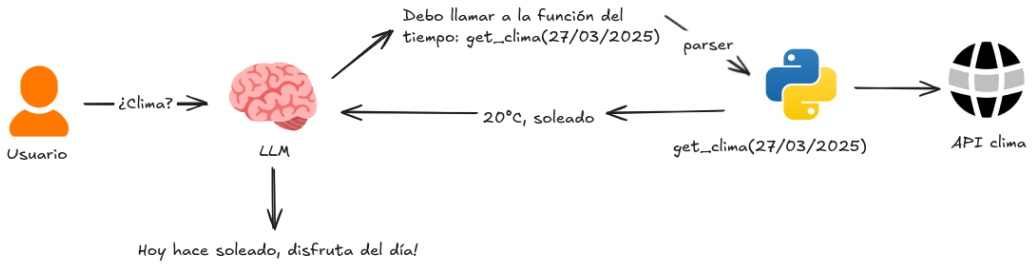

Figura 2.1: Ejemplo de interacción de un modelo LLM con una herramienta externa.

Tokens Los tokens constituyen la unidad mínima de texto que el modelo puede procesar. Dado que dichos modelos operan sobre estructuras matemáticas, requieren transformar el lenguaje natural en representaciones matriciales. Para lograr esta conversión, el texto se segmenta en dichas unidades mínimas, que pueden corresponder a caracteres individuales, fragmentos de texto o palabras completas. El conjunto íntegro de estas unidades reconocibles por el modelo configura su vocabulario.

Representaciones vectoriales Constituyen vectores numéricos de dimensionalidad fija que codifican la semántica inherente a cada token. Estos vectores pueden comprender desde 768 dimensiones en arquitecturas como BERT-base hasta superar las 16.000 dimensiones en los modelos más avanzados del estado del arte. Por ejemplo, una dimensión específica podría especializarse en representar conceptos abstractos. En este contexto, la representación vectorial del token “animal“ contendría un valor más elevado en dicha dimensión que la correspondiente al término “gato“, reflejando su mayor grado de abstracción conceptual.

2.1.2. Interacción con herramientas externas

Los agentes LLM poseen la capacidad de interactuar con diversas herramientas como búsquedas web, bases de datos o interfaces de usuario. Fundamentalmente, un LLM solo genera tokens de texto, por lo que la integración de herramientas se implementa mediante palabras clave que el modelo puede incluir en su salida. Para ello, en el texto de entrada se especifica el esquema de la función a utilizar y, si decide emplearla, el modelo generará el texto correspondiente. Posteriormente, se procesa la respuesta para extraer llamadas a funciones si las hubiese.

La interacción con herramientas es típicamente alternante. Tras realizar la llamada a la herramienta, la salida de esta se utilizará como entrada para el siguiente mensaje del modelo. La figura 2.1 ilustra el esquema de un agente con acceso a una API del clima. Como el modelo carece de información climática en tiempo real, se le indica en el prompt la posibilidad de invocar esta función. Al incluir la llamada en su texto de salida, se ejecuta la función y su respuesta se transmite al modelo para generar el resultado final.

2.1.3. Abstraciones en frameworks

Aunque los agentes LLM son una tecnología de reciente surgimiento, ya se han desarrollado frameworks que estandarizan su implementación. Estas estructuras de trabajo ofrecen abstracciones de alto nivel para reutilizar funcionalidades comunes presentes en la mayoría de sistemas de agentes. Las funcionalidades principales que estos frameworks proporcionan son:

- **Gestión de modelos:** La ejecución de LLM requiere dominio del modelo empleado, ya que cada uno posee tokenizadores específicos y esquemas propios de entrada/salida. Los frameworks ofrecen interfaces unificadas, facilitando el uso de diversos modelos sin conocimientos técnicos precisos.
- **Interacción conversacional:** La comunicación con los modelos se efectúa mediante un esquema conversacional, donde el modelo recibe un texto de entrada y genera una respuesta correspondiente. Las respuestas y entradas se concatenan secuencialmente para preservar el contexto de la conversación, cada consulta subsiguiente incorpora todos los intercambios precedentes.
- **Uso de herramientas externas:** El desarrollador únicamente debe especificar la función que desea incorporar, toda la compljeidad de la interacción se abstrae en el framework.
- **Interacción entre agentes:** Los agentes pueden establecer comunicación entre sí, permitiendo la construcción de sistemas con mayor complejidad. Algunos frameworks establecen protocolos que definen las modalidades de comunicación entre los distintos agentes.

Entre las más populares se encuentran LangChain, LangGraph, LlamaIndex, AutoGen, CrewAI, Smolagents y el reciente OpenAI Agents.

2.2. Model Context Protocol

todo: referencias a las docs.

El Model Context Protocol (MCP), desarrollado por Anthropic, estandariza la comunicación entre agentes LLM y herramientas. Permite que aplicaciones diversas ofrezcan herramientas a agentes externos sin exponer detalles de implementación. Comparable al modelo OSI, el MCP opera en un nivel de abstracción inferior a los frameworks, proporcionando una capa de interoperabilidad.

La figura 2.2 ilustra el esquema operativo del protocolo. En este contexto, los desarrolladores de Jira y GitHub han implementado un servidor MCP que proporciona las herramientas disponibles al cliente MCP. Este servidor realiza la traducción de las interacciones necesarias con las API de Jira y GitHub, permitiendo que el agente LLM únicamente requiera conocer el esquema funcional que debe utilizar. Por su parte, el cliente MCP se encarga de administrar la

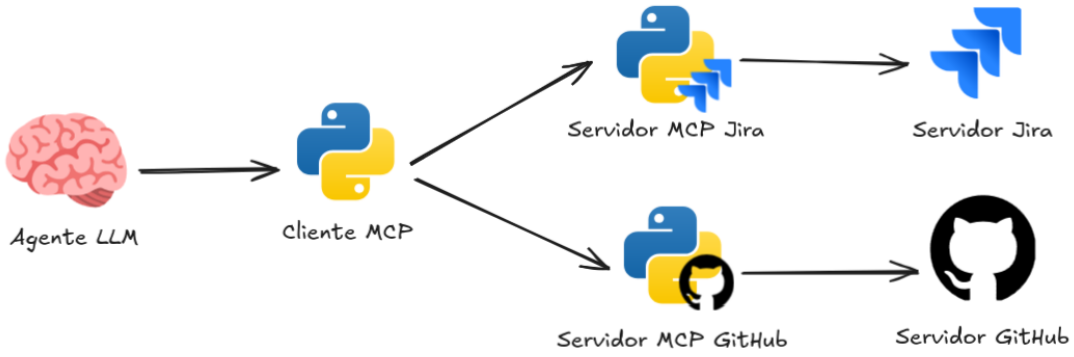

Figura 2.2: Esquema de funcionamiento del Model Context Protocol.

comunicación con los diferentes servidores, facilitando que el agente acceda directamente a las herramientas disponibles.

El protocolo ofrece dos modos de operación para establecer la comunicación entre cliente y servidor:

- **Comunicación SSE:** El protocolo Server-Sent Events (SSE) establece un canal de comunicación unidireccional sobre HTTP desde el servidor hacia el cliente. Proporciona actualizaciones en tiempo real con capacidad de streaming. En el protocolo MCP, el cliente efectúa solicitudes para la ejecución de herramientas en el servidor mediante HTTP, a lo que el servidor puede responder mediante eventos SSE.
- **Comunicación STDIO:** El protocolo de entrada y salida estándar (STDIO) facilita la comunicación bidireccional entre cliente y servidor a nivel de proceso en el sistema operativo. Este mecanismo permite el intercambio de información en formato JSON a través de los canales estándar del sistema. Su diseño, orientado principalmente a entornos locales, restringe la conexión a un único cliente por servidor al limitarse a la comunicación entre dos procesos.

La aplicación de escritorio claude-desktop de Anthropic constituye un reflejo del potencial del protocolo. Esta plataforma ofrece la posibilidad de interactuar con servidores preconfigurados mediante una configuración mínima. Implementando el protocolo STDIO, la aplicación ejecuta los servidores distribuidos por terceros a través del gestor de paquetes UV o Docker. Al incorporar un cliente MCP en la aplicación, consigue integrar las herramientas disponibles en la interfaz de chat con los modelos de Anthropic.

2.3. Estado del arte en arquitecturas de agentes LLM

La comunidad científica ha desarrollado diferentes arquitecturas de agentes para sacar el máximo provecho al rendimiento de los modelos disponibles. Es de destacar la arquitectura

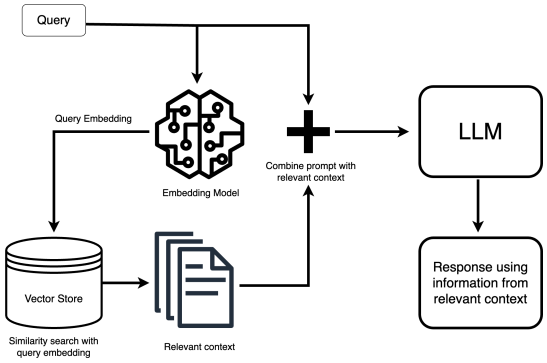

Figura 2.3: Esquema de funcionamiento de la arquitectura RAG en un LLM Fuente.

RAG, la cual complementa la entrada del modelo con la recuperación de documentos relevantes. Por otro lado, se han realizado diversos esfuerzos en investigar diferentes arquitecturas de comunicación, estrategias de planning o sistemas de memoria, entre otros.

2.3.1. Arquitectura RAG

Los modelos LLM poseen un conocimiento restringido a los datos con los que fueron entrenados. Para superar esta limitación, la arquitectura RAG (Retrieval-Augmented Generation) complementa la generación del LLM mediante la recuperación de información relevante desde repositorios de conocimiento externos. La figura 2.3 ilustra un ejemplo de su funcionamiento.

La recuperación de documentos relevantes se puede implementar mediate recuperadores dispersos: expresiones regulares, búsqueda de trigramas [ref], palabras clave, entre otras. No obstante, el enfoque predominante consiste en el uso de recuperadores densos, conocidos como indexación vectorial. En este método, los documentos se transforman en vectores, generalmente mediante LLMs especializados en codificación, denominados Embedders. Al representar los documentos en un espacio vectorial, es posible recuperar aquellos semánticamente más pertinentes mediante la comparación del vector de consulta con los vectores de los documentos indexados, utilizando métricas como la distancia coseno.

2.3.1.1. Estrategias RAG avanzadas

La optimización del rendimiento en arquitecturas RAG ha sido objeto de extensa investigación [1][2], centrándose en cuatro áreas fundamentales: procesado de documentos, sistemas de recuperación, generación y optimización del pipeline general:

- **Procesado de documentos:** La calidad de la indexación documental determina la eficacia del sistema. Entre las estrategias destacadas figuran la eliminación de ruido textual, el ajuste dimensional de chunks y el sistema overlapping window, que superpone fragmentos para preservar la información limítrofe.

  Se han desarrollado también técnicas generativas para mejorar la calidad documental, desde la recuperación de contenidos directamente generados [3], la recitación de evi-

dencias [4], hasta el uso de documentos generados a partir de la query para localizar información relevante [5]. Estas técnicas pueden complementarse con algoritmos como beam search para incrementar su efectividad [6][7][8].

- **Sistemas de recuperación:** Los recuperadores densos se clasifican en tres categorías: basados en representaciones (ilustrados en la figura 2.3); basados en interacción, que emplean LLM embedders para codificar documentos junto con la consulta, evaluando su relevancia mediante clasificadores densos [9][10]; y enfoques híbridos que combinan ambas estrategias, filtrando inicialmente por representaciones y refinando posteriormente mediante técnicas de interacción [11]. Adicionalmente, existen métodos para condensar la información recuperada, minimizando así la ventana contextual de los generadores [12][13].

- **Optimización del pipeline y generación:** Para tareas que requieren reflexión documental, se han propuesto sistemas de salto múltiple que alternan entre recuperación y generación [14][15][16][17]. El framework DSP [18] constituye el referente en sistemas de pregunta-respuesta, utilizando demostraciones para entrenar tanto el extractor como el generador, alternando posteriormente entre búsqueda y predicción. Adaptive RAG [19] propone clasificar previamente las consultas según su complejidad para determinar la ejecución del proceso, mientras que otros enfoques integran modelos más compactos para tareas específicas como el resumen [20].

Adicionalmente, el ajuste fino de modelos para tareas RAG específicas mejora significativamente el rendimiento, ya sea optimizando embedders[21], sistemas de extracción [11][22], generación[23][24] o reescritura de consultas[9]. Diversos trabajos han ajustado extractores para LLMs de caja negra[25][26][27][28], mientras otros enfoques modifican la arquitectura interna del modelo para incorporar contexto de documentos relevantes durante la generación[29].

2.3.2. Arquitecturas de interacción entre agentes

La interacción entre agentes LLM es también un campo de investigación activo.

Eranskina / apéndice

Eranskinak

Bibliografía

[1] Fengbin Zhu, Wenqiang Lei, Chao Wang, Jianming Zheng, Soujanya Poria, and Tat-Seng Chua. Retrieving and reading: A comprehensive survey on open-domain question answering. Ver página 7.

[2] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. Ver página 7.

[3] Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. Generate rather than retrieve: Large language models are strong context generators. Ver página 7.

[4] Zhiqing Sun, Xuezhi Wang, Yi Tay, Yiming Yang, and Denny Zhou. Recitation-augmented language models. Ver página 8.

[5] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777. Association for Computational Linguistics. Ver página 8.

[6] Xin Cheng, Di Luo, Xiuying Chen, Lemao Liu, Dongyan Zhao, and Rui Yan. Lift yourself up: Retrieval-augmented text generation with self-memory. Ver página 8.

[7] Sukmin Cho, Jeongyeon Seo, Soyeong Jeong, and Jong C. Park. Improving zero-shot reader by reducing distractions from irrelevant documents in open-domain question answering. Ver página 8.

[8] Ori Yoran, Tomer Wolfson, Ben Bogin, Uri Katz, Daniel Deutch, and Jonathan Berant. Answering questions by meta-reasoning over multiple chains of thought. Ver página 8.

[9] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models. Ver página 8.

[10] Yoav Levine, Itay Dalmedigos, Ori Ram, Yoel Zeldes, Daniel Jannai, Dor Muhlgay, Yoni Osin, Opher Lieber, Barak Lenz, Shai Shalev-Shwartz, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. Standing on the shoulders of giant frozen language models. Ver página 8.

[11] Omar Khattab, Christopher Potts, and Matei Zaharia. Relevance-guided supervision for OpenQA with ColBERT. Ver página 8.

[12] Omar Khattab, Christopher Potts, and Matei Zaharia. Baleen: Robust multi-hop reasoning at scale via condensed retrieval. Ver página 8.

[13] Fangyuan Xu, Weijia Shi, and Eunsol Choi. RECOMP: Improving retrieval-augmented LMs with compression and selective augmentation. Ver página 8.

[14] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9248–9274. Association for Computational Linguistics. Ver página 8.

[15] Peng Qi, Haejun Lee, Oghenetegiri "TG"Sido, and Christopher D. Manning. Answering open-domain questions of varying reasoning steps from text. Ver página 8.

[16] Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V. Le, and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models. Ver página 8.

[17] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. Ver página 8.

[18] Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive NLP. Ver página 8.

[19] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C. Park. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. Ver página 8.

[20] Yubo Ma, Yixin Cao, YongChing Hong, and Aixin Sun. Large language model is not a good few-shot information extractor, but a good reranker for hard samples! In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10572–10601. Ver página 8.

[21] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. Ver página 8.

[22] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot learning with retrieval augmented language models. Ver página 8.

[23] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. SELF-RAG: LEARNING TO RETRIEVE, GENERATE, AND CRITIQUE THROUGH SELF-REFLECTION. Ver página 8.

[24] Kalpesh Krishna, Yapei Chang, John Wieting, and Mohit Iyyer. RankGen: Improving text generation with large ranking models. Ver página 8.

[25] Zichun Yu, Chenyan Xiong, Shi Yu, and Zhiyuan Liu. Augmentation-adapted retriever improves generalization of language models as generic plug-in. Ver página 8.

[26] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. REPLUG: Retrieval-augmented black-box language models. Ver página 8.

[27] Xi Victoria Lin, Xilun Chen, Mingda Chen, Weijia Shi, Maria Lomeli, Rich James, Pedro Rodriguez, Jacob Kahn, Gergely Szilvasy, Mike Lewis, Luke Zettlemoyer, and Scott Yih. RA-DIT: RETRIEVAL-AUGMENTED DUAL INSTRUC- TION TUNING. Ver página 8.

[28] Haoyan Yang, Zhitao Li, Yong Zhang, Jianzong Wang, Ning Cheng, Ming Li, and Jing Xiao. PRCA: Fitting black-box large language models for retrieval question answering via pluggable reward-driven contextual adapter. Ver página 8.

[29] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-augmented language model pre-training. Ver página 8.