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 Software

Título del trabajo

Martín López de Ipiña Muñoz

Dirección
Galdos Otermin, Aritz
Pereira Varela, Juanan
Azanza Sesé, Maider

13 de abril de 2025

Resumen

Índice de contenidos

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

CAPÍTULO 2

Antecedentes

Una vez establecido el objetivo de este proyecto en la introducción, en este capítulo se describirán los conceptos generales necesarios para la comprensión de este documento. Para ello, en primer lugar se detalla el proceso de onboarding en proyectos software y las dificultades que presenta, junto con el trabajo previo realizado en este ámbito.

Por otro lado, se explicará a grandes rasgos qué son los agentes basados en grandes modelos de lenguaje (conocidos también por el anglicismo Large Language Models o por sus siglas LLM), su arquitectura, funcionamiento e interacción con herramientas externas. Se introduce además el Model Context Protocol como estándar de comunicación entre estos componentes.

Finalmente, se aborda el estado del arte en arquitecturas de agentes, sus aplicaciones en proyectos software y las técnicas de ajuste de modelos.

2.1. Onboarding en proyectos software

El proceso de incorporación (onboarding) de nuevos desarrolladores de software constituye un desafío persistente para las organizaciones tecnológicas, donde los recién incorporados enfrentan una sobrecarga informativa mientras los desarrolladores senior ven afectada su productividad al destinar tiempo considerable a actividades de formación y mentoría[1].

Si bien prácticas como la designación de mentores han demostrado ser efectivas para facilitar la integración de nuevos miembros, estas incrementan significativamente la carga de trabajo sobre los profesionales experimentados, generando potenciales retrasos en los proyectos[2].

En este contexto, los modelos de lenguaje de gran escala emergen como una alternativa prometedora para transformar el proceso de onboarding, ofreciendo orientación personalizada e instantánea que podría reducir la dependencia de los desarrolladores senior, preservar la

productividad global de los equipos y facilitar una incorporación más eficiente y menos disruptiva[3].

2.1.1. Trabajo previo

La Universidad del País Vasco, en colaboración con LKS NEXT[1], desarrolló el prototipo denominado I Need a Hero (INAH), diseñado para aprovechar el potencial de los LLM en la localización de expertos dentro de la organización[4]. INAH opera en dos fases: primero crea perfiles de "héroes" extrayendo información de currículos de empleados dispuestos a asistir; luego, ante una consulta, utiliza GPT-3.5 para identificar las competencias requeridas y localizar a los profesionales que las poseen.

En una línea de estudio complementaria, un trabajo reciente ha presentado el sistema "Onboarding Buddy", el cual implementa una arquitectura multi-agente que organiza diversos componentes especializados para proporcionar asistencia contextualizada durante la incorporación de nuevos desarrolladores[5].

El sistema fundamenta su funcionamiento en la generación dinámica de planes mediante cadena de pensamiento 2.4.2, evaluados posteriormente para determinar su posible descomposición en sub-tareas procesadas en paralelo por otros agentes.

2.2. Agentes de Grandes Modelos de Lenguaje (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.

2.2.1. Modelos LLM

Los Grandes Modelos de Lenguaje son redes neuronales especializadas en el procesamiento del lenguaje natural que funcionan mediante un mecanismo de entrada-salida de tokens 2.2.1. Estos modelos reciben secuencias de tokens como entrada, denominada comúnmente "prompt", y generan secuencias de tokens como salida, aplicando durante este proceso las representaciones y relaciones semánticas aprendidas durante su fase de entrenamiento con extensos corpus textuales [6].

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

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

[1]LKS NEXT: https://www.lksnext.com/es/

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. 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.

Ajuste de modelos instruct El entrenamiento 2.6.3 de los LLM se estructura en dos fases diferenciadas. La primera corresponde al preentrenamiento, donde el modelo procesa extensos conjunto de datos textuales con operaciones como intentar predecir el siguiente token en la secuencia. Esta fase permite al modelo captar las complejas estructuras sintácticas y relaciones semánticas inherentes al lenguaje natural. La segunda fase consiste en el ajuste fino, donde el modelo previamente entrenado se especializa mediante conjuntos de datos específicos y etiquetados 2.6.3, optimizando su capacidad para ejecutar tareas concretas de clasificación o generación de texto.

Los agentes basados en LLM implementan en su mayoría modelos *instruct*, variantes especialmente ajustadas para responder a consultas e instrucciones de usuarios. Dentro de esta categoría se encuentran GPT (base de ChatGPT[2]) de OpenAI[3], Claude Sonnet de Anthropic[4], y LLama-Instruct de Meta[5].

2.2.2. 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, este tipo de modelos solo genera tokens de texto, por lo que la integración de herramientas se implementa mediante palabras clave o tokens especiales que este 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]ChatGPT:https://chatgpt.com
[3]OpenAI:https://openai.com/
[4]Anthropic:https://www.anthropic.com/
[5]Meta: https://about.meta.com/es/

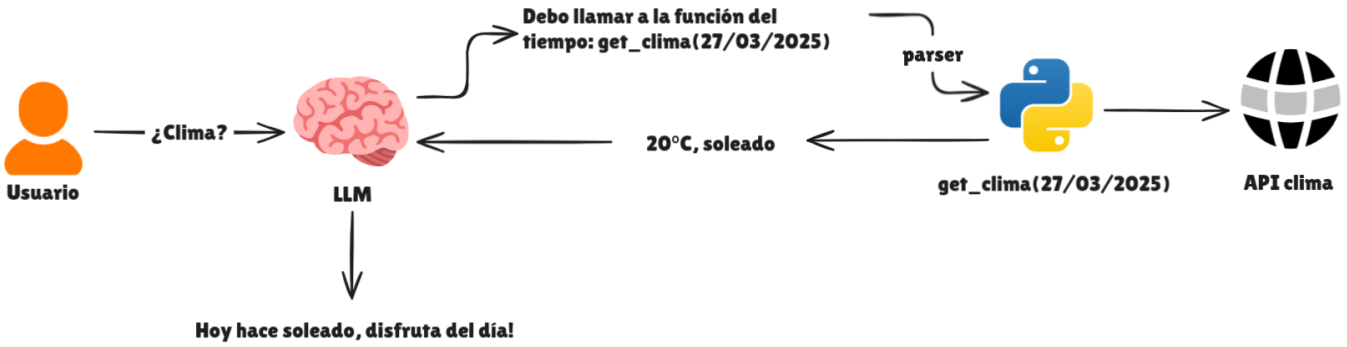

Figura 2.1: Ejemplo de interacción de un modelo LLM con una herramienta externa.

2.2.2.1. Patrón ReAct

El agente Reasoning and Act (ReAct) constituye uno de los patrones más utilizados[7]. Se basa en un ciclo de tres pasos fundamentales: el razonamiento como generación de pensamiento sobre posibles acciones, la acción como ejecución de la herramienta seleccionada y la observación como procesamiento del resultado obtenido. Este ciclo se repite iterativamente hasta que durante la fase de razonamiento se determina que la tarea ha sido completada.

2.2.3. Abstracciones en frameworks

En pleno auge de los agentes LLM, surgen cada vez más bibliotecas y frameworks que estandarizan su implementación. Estos marcos de trabajo ofrecen abstracciones de alto nivel para reutilizar funcionalidades comunes presentes en la mayoría de sistemas de agentes. Las principales funcionalidades proporcionadas son las siguientes:

- **Gestión de modelos:** La ejecución de modelos de lenguaje requiere del dominio de estos, ya que cada uno posee tokenizadores específicos 2.6.3 y esquemas propios de entrada y salida. Los frameworks ofrecen interfaces unificadas, facilitando el uso de diversos modelos sin necesidad de conocimientos técnicos excesivamente detallados.
- **Interacción conversacional:** La comunicación con los agentes 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:** Toda la complejidad de la interacción se abstrae en el framework, por lo que el desarrollador únicamente debe especificar la función que desea incorporar.
- **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.

Para este trabajo utilizaremos fundamentalmente LangChain[6] para la gestión de llamadas a APIs de modelos y prompting, LangGraph[7] para la orquestación de flujos agénticos, y LangSmith[8] para el seguimiento de trazas de llamadas a modelos y herramientas. Para la gestión de los conjuntos de datos y entrenamiento de modelos utilizaremos HuggingFace.[9]

2.3. Model Context Protocol

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[8].

La figura 2.2 ilustra el esquema operativo del protocolo. Los desarrolladores de Jira[10] y GitHub[11] han implementado un servidor MCP que traduce las interacciones con sus APIs y proporciona herramientas al cliente MCP. Esto permite que el desarrollador del agente se enfoque exclusivamente en conectar las herramientas del cliente con el agente, sin necesidad de interactuar con APIs externas. Asimismo, el cliente MCP gestiona la comunicación entre servidores, facilitando al agente el acceso directo a múltiples herramientas.

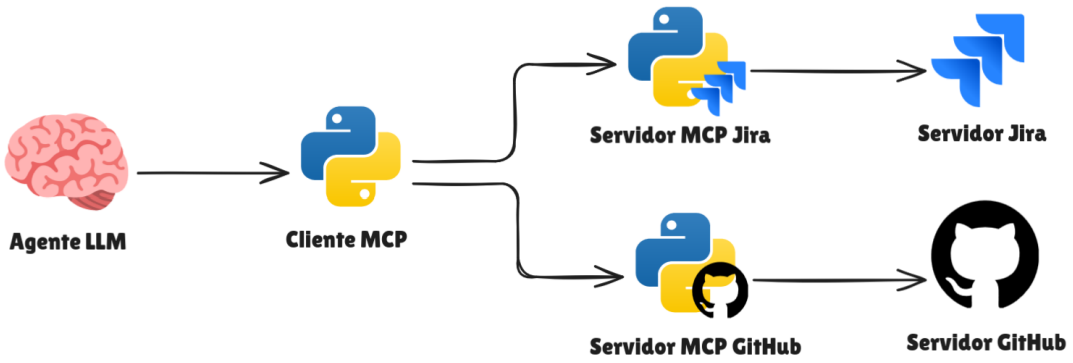

Figura 2.2: Esquema de funcionamiento del Model Context Protocol.

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

[6] LangChain: https://www.langchain.com/
[7] LangGraph: https://www.langchain.com/langgraph
[8] LangSmith: https://www.langchain.com/langsmith
[9] HuggingFace: https://huggingface.co/
[10] Jira: https://www.atlassian.com/es/software/jira
[11] GitHub: https://github.com/

- **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[12] de Anthropic constituye un reflejo del potencial del protocolo. Esta plataforma ofrece la posibilidad de interactuar con servidores mediante una configuración mínima. Implementando el protocolo STDIO, la aplicación ejecuta los servidores distribuidos por terceros a través de gestores de paquetes como uv[13], npx[14] o Docker[15]. Al incorporar un cliente MCP en la aplicación, consigue integrar las herramientas disponibles en la interfaz de chat con los modelos de Anthropic.

2.4. Estado del arte en arquitecturas de agentes LLM

La comunidad científica ha diseñado diversas arquitecturas de agentes para optimizar el rendimiento de los modelos disponibles. La arquitectura RAG se distingue por complementar la entrada del modelo con información recuperada de documentos relevantes. Otras propuestas se centran en mejorar la comunicación y coordinación entre agentes.

2.4.1. 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.

[12] Claude Desktop: https://claude.ai/download
[13] uv: https://astral.sh/blog/uv
[14] npx: https://docs.npmjs.com/cli/v8/commands/npx
[15] Docker: https://www.docker.com/

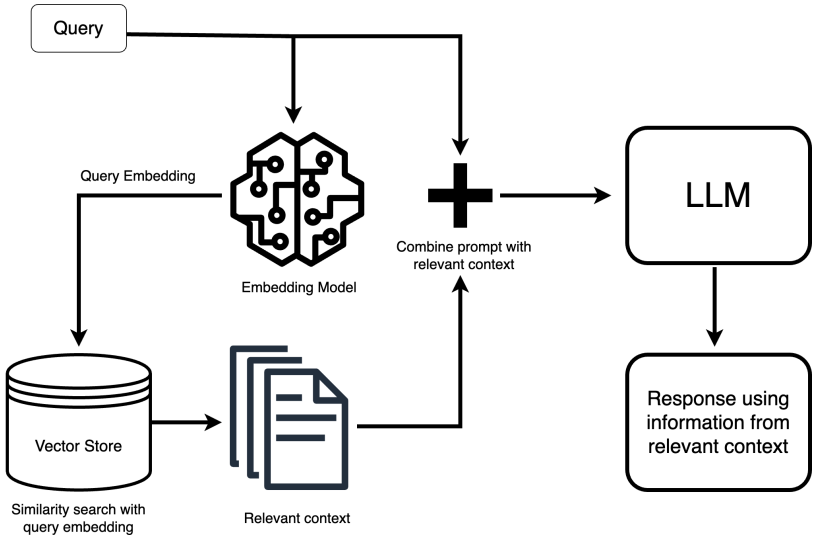

Figura 2.3: Esquema de funcionamiento de la arquitectura RAG en un LLM Fuente.

La recuperación de documentos relevantes se puede implementar mediate recuperadores dispersos: expresiones regulares, búsqueda de n-gramas, 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 2.6.3.

2.4.1.1. Estrategias RAG avanzadas

La optimización del rendimiento en arquitecturas RAG ha sido ampliamente estudiada [9][10], enfocándose en tres áreas principales: el procesado de documentos, los sistemas de recuperación y la mejora del flujo de generación.

- **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 del tamaño óptimo de los segmentos indexados, y la técnica de ventana solapada. Esta última superpone fragmentos para preservar los datos situados en las intersecciones de los segmentos.
- **Sistemas de recuperación:** Los recuperadores densos basados en representaciones, ilustrados en la figura 2.3, precalculan las representaciones vectoriales de los documentos mediante una indexación independiente de las consultas. Por otro lado, los recuperadores basados en interacción, proponen calcular el vector para cada documento junto al vector de consulta durante el tiempo de ejecución, obteniendo así una representación más rica que captura las relaciones contextuales específicas entre ambos elementos[11][12].

Para contrarrestar el sobrecoste computacional de este enfoque, se han desarrollado técnicas de indexación híbridas que combinan ambos métodos. Estas técnicas permiten realizar una búsqueda inicial utilizando representaciones vectoriales y posteriormente refinar los resultados mediante la extracción interactiva[13].

- **Optimización del flujo de generación:** Para tareas que requieren reflexión documentada, se han desarrollado sistemas de salto múltiple que alternan entre recuperación y generación[14][15][16][17][18]. Estos flujos permiten al sistema examinar críticamente la información inicialmente recuperada, identificar lagunas informativas, y formular las consultas correspondientes para subsanar dichas carencias.

2.4.2. Arquitecturas de interacción entre agentes

La interacción entre agentes LLM constituye un campo de investigación activo, distinguiéndose diversos avances en módulos de memoria, planificación e interacción multiagente[19].

- **Módulos de memoria**: En una interacción conversacional, el modelo procesa todos los mensajes previos, pudiendo generar un contexto excesivamente amplio. Para mitigar este problema, se han desarrollados módulos de memoria que almacenan información relevante de interacciones pasadas de forma resumida[20][21][22]. Esta memoria puede consultarse posteriormente mediante RAG, permitiendo recuperar los elementos más relevantes según el contexto[23].

  Algunos módulos se inspiran en la estructura de la memoria humana[24], incorporando mecanismos que almacenan información con distintas temporalidades y niveles de relevancia[19][25].

- **Planificación**: Los mecanismos de planificación potencian el razonamiento de los agentes sobre sus acciones futuras.

  Entre estos mecanismos destaca el prompting de cadena de pensamiento (Chain of Thought)[26], que instruye al modelo para elaborar un razonamiento secuencial previo a su decisión final, permitiendo así descomponer problemas complejos paso a paso. Partiendo de este enfoque, estrategias avanzadas como la autoconsciencia[22] lo amplían mediante la generación de múltiples cadenas de razonamiento independientes y la posterior selección de la respuesta óptima entre ellas[27][28].

  De manera complementaria, las técnicas de reflexión[29][30][31] implementan un proceso iterativo donde el propio modelo evalúa y refina sus respuestas.

  Por otro lado, la estructuración de acciones constituye una metodología ampliamente adoptada en la planificación[32][33][34]. Esta técnica permite definir planes de alto nivel que posteriormente se desglosan en acciones específicas ejecutables por los agentes[35][36][37][38]. Adicionalmente, la definición de interdependencias entre estas acciones permite verificar la validez de los planes generados[39][40][41].

  Los modelos razonadores como o1 de OpenAI o DeepSeek-R1 incorporan estas técnicas de forma nativa[42]. Estos han sido entrenados con datos que incluye ejemplos de

razonamiento y planificación, lo que les permite generar respuestas que incluyen dichas estructuras.

- **Interacción entre agentes:** Los sistemas multi-agente implementan una arquitectura donde diversos agentes especializados son coordinados por un componente central denominado orquestador[43][44]. En este paradigma, cada agente se especializa en una función particular, como la búsqueda de información, la ejecución de herramientas o la generación de texto. El orquestador evalúa las consultas entrantes y las dirige hacia el agente más competente para resolverlas.

  Enfoques complementarios proponen la interacción directa entre agentes especializados como mecanismo de retroalimentación[45][46]. Por ejemplo, ChatDev[47] establece un sistema de colaboración entre agentes programadores, testers y gestores para abordar problemas de ingeniería de software. MetaGPT[48] refina esta propuesta al implementar un protocolo de comunicación basado en el patrón publicador/suscriptor entre los agentes, permitiéndoles difundir información de forma selectiva.

2.5. Agentes LLM en proyectos software

La integración de agentes en proyectos software ha sido objeto de estudio enfocándose principalmente en el ámbito de la generación automática de código. Se pueden destacar tres niveles de autonomía en los productos desarrollados: sugerencias de autocompletado, asistentes de programación y agentes autónomos.

- **Sugerencias de autocompletado:** GitHub Copilot[16] o Cursor[17] ofrecen la integración de un modelo LLM ajustado para generación de código directamente al entorno del desarrollador. De esta forma, el desarrollador recibe sugerencias de autocompletado en tiempo real mientras escribe el código.

  Este tipo de herramientas han demostrado ser eficaces para aumentar la productividad de los desarrolladores en tareas de programación repetitivas[49]. Sin embargo, debido a su enfoque en la rapidez de respuesta, presentan limitaciones en la aplicación de razonamiento profundo.

- **Asistentes de programación:** Principalmente en aplicaciones en forma de chat, herramientas como GitHub Copilot Chat, Cursor o Cody[18] de SourceGraph[19] ofrecen un sistema de chat interactivo que permite al desarrollador realizar preguntas específicas sobre el código[50].

  Cabe destacar el proyecto SourceGraph, el cual al ser inicialmente de código abierto ha publicado una explicación de su funcionamiento. Proporciona un sistema avanzado de navegación de código que permite la integración del agente LLM denominado Cody.

[16] GitHub Copilot: https://github.com/features/copilot
[17] Cursor: https://www.cursor.com/
[18] Cody: https://sourcegraph.com/cody
[19] SourceGraph: https://sourcegraph.com/

La arquitectura de este sistema opera en dos etapas: en primera instancia, los servidores de lenguaje específicos para cada lenguaje de programación analizan la estructura del código fuente, generando un grafo de dependencias que representa las relaciones jerárquicas entre los distintos elementos del código (funciones, clases, métodos, interfaces, entre otros)[51]. Posteriormente, el sistema implementa un mecanismo RAG basado en trigramas sobre el grafo del proyecto, cuyos fragmentos son suministrados al agente Cody, que procesa esta información estructurada y elabora respuestas contextualizadas a las consultas del desarrollador[52].

- **Agentes autónomos:** Con un enfoque más ambicioso, soluciones como Aider[20] o DevinAI[21] persiguen la automatización del ciclo completo de desarrollo de software. Debido a la complejidad inherente a dicha tarea, estas herramientas permanecen en una fase incipiente[53].

  El funcionamiento de Aider presenta similitudes con SourceGraph[54]. En primera instancia, analiza el código fuente mediante la biblioteca Tree-sitter[22], generando un grafo de dependencias que representa las definiciones (funciones, clases, interfaces, etc.) y las referencias entre estas. Posteriormente, implementa un algoritmo de clasificación de grafos para determinar la relevancia de cada nodo respecto al contexto actual. De este modo, incorpora dicho contexto junto con un mapa de los ficheros del proyecto en la entrada del agente LLM.

2.6. Ajuste de modelos para agentes LLM

2.6.1. Limitaciones de los modelos actuales

Los modelos *instruct* del estado del arte poseen la capacidad de resolver tareas que requieren comportamiento agéntico gracias a su amplio conocimiento general. No obstante, su eficiencia es cuestionable, puesto que para tareas específicas pueden carecer de ciertos conocimientos particulares, mientras que disponen de una cantidad considerable de información superflua. Se estima que los modelos más avanzados en la actualidad, como GPT-4o o Claude 3.7 Sonnet, contienen cientos de miles de millones de parámetros[55], lo que los convierte en prohibitivamente costosos para su ejecución en entornos locales.

Debido a estas limitaciones y por tratarse de modelos propietarios, estos están disponibles únicamente a través de API. Este modelo de acceso requiere enviar los datos a servidores externos, planteando así preocupaciones sobre la privacidad y la seguridad de la información. Esta restricción es especialmente relevante en el ámbito del desarrollo de software, donde los datos pueden incluir información sensible o confidencial.

[20] Aider: https://aider.chat/
[21] DeviAI: https://devin.ai/
[22] Tree-sitter: https://tree-sitter.github.io/tree-sitter/

2.6.2. Enfoques de ajuste fino para agentes específicos

Como consecuencia de estas limitación, diversos proyectos de investigación han propuesto el ajuste fino de modelos de menor dimensión para tareas agénticas específicas. Mediante esta aproximación, es posible obtener modelos con mayor precisión en las tareas requeridas, pero con una fracción del coste.

Enfoques como FireAct[56] y AgentTuning[57] proponen el entrenamiento de dichos modelos mediante una estrategia de destilación. Esto consiste en previamente utilizar un modelo instruct de gran tamaño para generar un conjunto de datos sintéticos para las respuestas del agente. Posteriormente, se ajusta un modelo de menor tamaño utilizando este conjunto de datos, de forma que el conocimiento del modelo grande se destila en el modelo pequeño.

Adicionalmente, Agent-FLAN[58] propone la división de dichos datos de entrenamiento en varios conjuntos de aprendizaje. Debido a que los modelos aprenden antes a interpretar el formato de entrada que a razonar sobre la acción a realizar, se propone modificar parte de las trazas de entrenamiento para que estas tengan una estructura conversacional. De esta forma, al ajustar el modelo con las trazas originales, este aprende la estructura de entrada y llamada de herramientas, mientras que en las trazas conversacionales, el modelo aprende a razonar sobre la acción a realizar.

2.6.3. Técnicas de entrenamiento eficiente

El entrenamiento de estos modelos se realiza mediante el ajuste de los pesos de la red neuronal. Debido a que el ajuste fino del modelo completo es costoso, en contextos de bajo coste se propone el uso de la técnica de entrenamiento Low Rank Adaptation (LoRA)[59]. Esta técnica consiste en ajustar únicamente un subconjunto de los pesos de la red neuronal, lo que permite reducir significativamente el coste computacional del ajuste fino.

Anexo A

Definición de términos técnicos

Conjunto de datos etiquetados: Un conjunto de datos etiquetado es una colección estructurada de información donde cada elemento o instancia está asociado a una o más categorías, clases o valores objetivo, denominados etiquetas. Estas etiquetas representan la información que se desea predecir o clasificar mediante un modelo de aprendizaje automático.

En el contexto del aprendizaje supervisado, estos conjuntos constituyen la base para el entrenamiento de modelos, ya que proporcionan ejemplos concretos de la relación entrada-salida que el algoritmo debe aprender a generalizar.

Entrenamiento de redes neuronales: El entrenamiento de una red neuronal consiste en un proceso iterativo de modificación de los pesos de las conexiones entre neuronas artificiales. Estos ajustes permiten que la red aprenda a generalizar a partir de los datos de entrenamiento, extrayendo patrones subyacentes que podrá aplicar posteriormente a datos no observados.

En el aprendizaje supervisado, específicamente durante el ajuste fino, los pesos se modifican comparando las predicciones del modelo con los datos de referencia. Esta comparación se cuantifica mediante una función de pérdida, cuyos gradientes, calculados mediante la regla de la cadena, indican cómo deben ajustarse los pesos para minimizar el error. Este mecanismo de retropropagación permite que la red optimice progresivamente su capacidad predictiva.

Distancia coseno: La distancia coseno es una medida que cuantifica la similitud entre dos vectores basándose en el coseno del ángulo que forman, independientemente de sus magnitudes. Matemáticamente se expresa como:

$$Similitud_coseno(x, y) = \frac{x \cdot y}{|x||y|}$$

El valor 1 indica vectores perfectamente alineados (máxima similitud), 0 representa vectores perpendiculares (sin similitud) y -1 señala vectores en direcciones opuestas (máxima disimilitud.

Tokenizador: Un tokenizador es el componente algorítmico encargado de segmentar el texto en unidades mínimas procesables (tokens), implementando reglas específicas de división basadas en espacios, puntuación, subpalabras o patrones predefinidos según el modelo de lenguaje.

Bibliografía

[1] S.E. Sim and R.C. Holt. The ramp-up problem in software projects: a case study of how software immigrants naturalize. In *Proceedings of the 20th International Conference on Software Engineering*, pages 361–370. ISSN: 0270-5257. Ver página 3.

[2] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F. Redmiles. A systematic literature review on the barriers faced by newcomers to open source software projects. 59:67–85. Ver página 3.

[3] Eva Ritz, Fabio Donisi, Edona Elshan, and Roman Rietsche. Artificial socialization? how artificial intelligence applications can shape a new era of employee onboarding practices. Ver página 4.

[4] Maider Azanza, Juanan Pereira, Arantza Irastorza, and Aritz Galdos. Can LLMs facilitate onboarding software developers? an ongoing industrial case study. In *2024 36th International Conference on Software Engineering Education and Training (CSEE&T)*, pages 1–6. ISSN: 2377-570X. Ver página 4.

[5] Andrei Cristian Ionescu, Sergey Titov, and Maliheh Izadi. A multi-agent onboarding assistant based on large language models, retrieval augmented generation, and chain-of-thought. Ver página 4.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. Ver página 4.

[7] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. Ver página 6.

[8] Model context protocol (MCP). Ver página 7.

[9] 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 9.

[10] 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 9.

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

[12] 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 9.

[13] Omar Khattab, Christopher Potts, and Matei Zaharia. Relevance-guided supervision for OpenQA with ColBERT. Ver página 10.

[14] 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 10.

[15] 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 10.

[16] Peng Qi, Haejun Lee, Oghenetegiri "TG"Sido, and Christopher D. Manning. Answering open-domain questions of varying reasoning steps from text. Ver página 10.

[17] 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 10.

[18] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. Ver página 10.

[19] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. 18(6):186345. Ver página 10.

[20] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. Ver página 10.

[21] Kevin A. Fischer. Reflective linguistic programming (RLP): A stepping stone in socially-aware AGI (SocialAGI). Ver página 10.

[22] Xinnian Liang, Bing Wang, Hui Huang, Shuangzhi Wu, Peihao Wu, Lu Lu, Zejun Ma, and Zhoujun Li. *Unleashing Infinite-Length Input Capacity for Large-scale Language Models with Self-Controlled Memory System.* Ver página 10.

[23] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. 38(17):19632–19642. Number: 17. Ver página 10.

[24] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. 38(17):19724–19731. Number: 17. Ver página 10.

[25] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22. ACM. Ver página 10.

[26] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. Ver página 10.

[27] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. Ver página 10.

[28] Yancheng Wang, Ziyan Jiang, Zheng Chen, Fan Yang, Yingxue Zhou, Eunah Cho, Xing Fan, Xiaojiang Huang, Yanbin Lu, and Yingzhen Yang. RecMind: Large language model powered agent for recommendation. Ver página 10.

[29] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. Ver página 10.

[30] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. SELF-REFINE: Iterative refinement with self-feedback. Ver página 10.

[31] Ning Miao, Yee Whye Teh, and Tom Rainforth. SelfCheck: Using LLMs to zero-shot check their own step-by-step reasoning. Ver página 10.

[32] Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. SWIFTSAGE: A generative agent with fast and slow thinking for complex interactive tasks. Ver página 10.

[33] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. Ver página 10.

[34] Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. Ver página 10.

[35] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. Ver página 10.

[36] Chan Hee Song, Brian M. Sadler, Jiaman Wu, Wei-Lun Chao, Clayton Washington, and Yu Su. LLM-planner: Few-shot grounded planning for embodied agents with large language models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2986–2997. IEEE. Ver página 10.

[37] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. Ver página 10.

[38] Shunyu Liu, Yaoru Li, Kongcheng Zhang, Zhenyu Cui, Wenkai Fang, Yuxuan Zheng, Tongya Zheng, and Mingli Song. Odyssey: Empowering minecraft agents with open-world skills. Ver página 10.

[39] Shreyas Sundara Raman, Vanya Cohen, Eric Rosen, Ifrah Idrees, David Paulius, and Stefanie Tellex. Planning with large language models via corrective re-prompting. Ver página 10.

[40] LLM+p: Empowering large language models with optimal planning proficiency. Ver página 10.

[41] Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic planning with a LLM. Ver página 10.

[42] DeepSeek-r1/DeepSeek_r1.pdf at main · deepseek-ai/DeepSeek-r1. Ver página 10.

[43] Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tenenholtz. MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. Ver página 11.

[44] Yingqiang Ge, Wenyue Hua, Kai Mei, Jianchao Ji, Juntao Tan, Shuyuan Xu, Zelong Li, and Yongfeng Zhang. OpenAGI: When LLM meets domain experts. Ver página 11.

[45] Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R. Ashley, Róbert Csordás, Anand Gopalakrishnan, Abdullah Hamdi, Hasan Abed Al Kader Hammoud, Vincent Herrmann, Kazuki Irie, Louis Kirsch, Bing Li, Guohao Li, Shuming Liu, Jinjie Mai, Piotr Piękos, Aditya Ramesh, Imanol Schlag, Weimin Shi, Aleksandar Stanić, Wenyi Wang, Yuhui Wang, Mengmeng Xu, Deng-Ping Fan, Bernard Ghanem, and Jürgen Schmidhuber. Mindstorms in natural language-based societies of mind. Ver página 11.

[46] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. Ver página 11.

[47] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. Ver página 11.

[48] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. Ver página 11.

[49] Eirini Kalliamvakou. Research: quantifying GitHub copilot's impact on developer productivity and happiness. Ver página 11.

[50] GitHub copilot features. Ver página 11.

[51] sourcegraph/scip. original-date: 2022-05-10T13:18:47Z. Ver página 12.

[52] sourcegraph/sourcegraph-public-snapshot: Code AI platform with code search & cody. Ver página 12.

[53] Kamal Acharya. Devin: A cautionary tale of the autonomous AI engineer. Ver página 12.

[54] Building a better repository map with tree sitter. Ver página 12.

[55] Number of parameters in GPT-4 (latest data). Ver página 12.

[56] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. FireAct: Toward language agent fine-tuning. Ver página 13.

[57] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. Ver página 13.

[58] Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. Ver página 13.

[59] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. Ver página 13.