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

12 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, 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 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.

Ajuste de modelos instruct El entrenamiento de los LLM se estructura en dos fases diferenciadas. La primera corresponde al preentrenamiento, donde el modelo procesa extensos corpus 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, 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 Gpt4-o (base de ChatGPT), Claude Sonnet, y la serie LLama-Instruct.

2.2. Onboarding mediante LLMs

2.2.1. Interacción con herramientas externas

Los agentes LLM poseen la capacidad de interactuar con diversas herramientas como búsquedas web[1], 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.

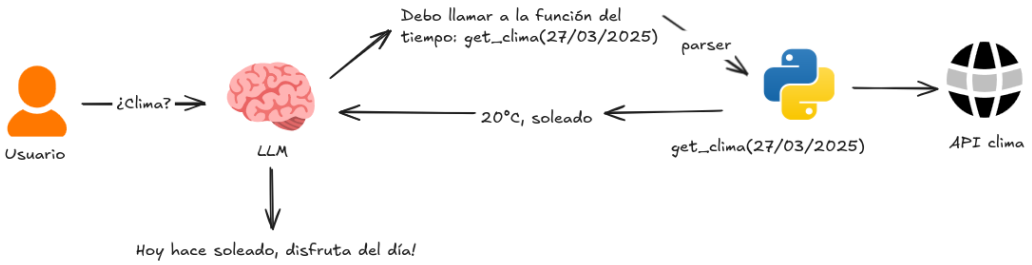

Figura 2.1: Ejemplo de interacción de un modelo LLM con una herramienta externa.

2.2.2. 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.

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

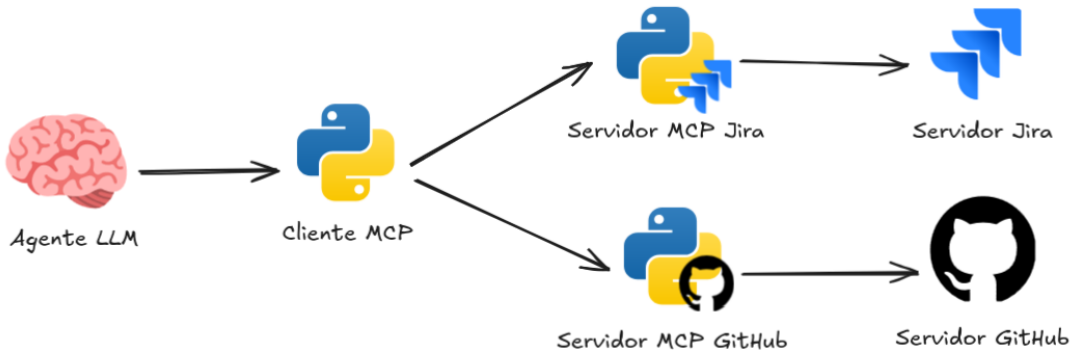

Figura 2.2: Esquema de funcionamiento del Model Context Protocol.

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

- **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 de gestores de paquetes como UV, o alternativamente con Docker. Al incorporar un cliente MCP en la aplicación, consigue integrar las herramientas disponibles en la interfaz de chat con los modelos de Anthropic.

2.4. Estado del arte en arquitecturas de agentes LLM

La comunidad científica ha creado 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.

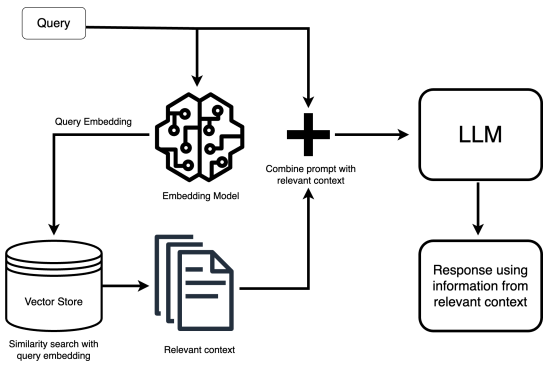

Figura 2.3: Esquema de funcionamiento de la arquitectura RAG en un LLM Fuente.

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.

2.4.1.1. Estrategias RAG avanzadas

La optimización del rendimiento en arquitecturas RAG ha sido ampliamente estudiada [2][3], 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[4][5].

  Para contrarrestar el sobrecoste computacional de este enfoque, se han desarrollado técnicas de indexación híbrida que combinan ambos métodos. Estas técnicas permiten realizar una búsqueda inicial utilizando representaciones vectoriales y posteriormente refinar los resultados mediante la extracción interactiva[6].

- **Optimización del flujo de generación:** Para tareas que requieren reflexión documentada, se han desarrollado sistemas de salto múltiple que alternan entre recuperación y generación[7][8][9][10][11]. 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[12].

- **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[13][14][15]. Esta memoria puede consultarse posteriormente mediante RAG, permitiendo recuperar los elementos más relevantes según el contexto[16].

Algunos módulos se inspiran en la estructura de la memoria humana[17], incorporando mecanismos que almacenan información con distintas temporalidades y niveles de relevancia[12][18].

- **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)[?][?], 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[15] lo amplían mediante la generación de múltiples cadenas de razonamiento independientes y la posterior selección de la respuesta óptima entre ellas[19][20].

  De manera complementaria, las técnicas de reflexión[21][21][22][23] 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[24][25][26]. Esta técnica permite definir planes de alto nivel que posteriormente se desglosan en acciones específicas ejecutables por los agentes[27][28][29][30]. El análisis de las interdependencias entre estas acciones permite verificar la validez de los planes generados[31][32][33].

  Los modelos razonadores como o1 o DeepSeek-R1 incorporan estas técnicas de forma nativa[34]. Son entrenados con datos que incluye ejemplos de razonamiento y planificación, lo que les permite generar respuestas que incluyen razonamiento y planificación.

- **Interacción entre agentes:** Los sistemas multi-agente implementan una arquitectura donde diversos agentes especializados son coordinados por un componente central denominado orquestador[35][36]. 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[37][38]. Por ejemplo, ChatDev[39] establece un sistema de colaboración entre agentes programadores, testers y gestores para abordar problemas de ingeniería de software. MetaGPT[40] refina esta propuesta al implementar un protocolo de comunicación basado en el patrón publicador/suscriptor entre los agentes, permitiéndoles difundir información de forma selectiva.

2.4.3. Agentes LLM en proyectos de software

La integración de agentes LLM 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 o Cursor 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[41]. 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, CursorAI o Cody de SourceGraph ofrecen un sistema de chat interactivo que permite al desarrollador realizar preguntas específicas sobre el código[42].

  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.

  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)[43]. 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[44].

- **Agentes autónomos:** Con un enfoque más ambicioso, soluciones como Aider o DevinAI 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[45].

  El funcionamiento de Aider presenta similitudes con SourceGraph[46]. En primera instancia, analiza el código fuente mediante la biblioteca tree-sitter, 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.

2.4.4. Ajuste de modelos para agentes LLM

2.4.4.1. 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 para dichas tareas. Los modelos más avanzados en la actualidad, como GPT-4.5 o Claude Sonnet 3.7, se estima que contienen varios trillones de parámetros (siguiendo la nomenclatura corta anglosajona), lo que los convierte en prohibitivamente costosos para su ejecución en entornos locales.

2.4.4.2. Enfoques de ajuste fino para tareas específicas

Como consecuencia de esta 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 y AgentTunning 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 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.

2.4.4.3. 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). 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.

Eranskina / apéndice

Eranskinak

Bibliografía

[1] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. WebGPT: Browser-assisted question-answering with human feedback. Ver página 4.

[2] 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 8.

[3] 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 8.

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

[5] 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.

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

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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 páginas 8, 9.

[13] 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 8.

[14] Kevin A. Fischer. Reflective linguistic programming (RLP): A stepping stone in socially-aware AGI (SocialAGI). Ver página 8.

[15] 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 páginas 8, 9.

[16] 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 8.

[17] 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 9.

[18] 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 9.

[19] 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 9.

[20] 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 9.

[21] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. Ver página 9.

[22] 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 9.

[23] Ning Miao, Yee Whye Teh, and Tom Rainforth. SelfCheck: Using LLMs to zero-shot check their own step-by-step reasoning. Ver página 9.

[24] 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 9.

[25] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. Ver página 9.

[26] 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 9.

[27] 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 9.

[28] 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 9.

[29] 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 9.

[30] 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 9.

[31] 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 9.

[32] LLM+p: Empowering large language models with optimal planning proficiency. Ver página 9.

[33] Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic planning with a LLM. Ver página 9.

[34] DeepSeek-r1/DeepSeek_r1.pdf at main · deepseek-ai/DeepSeek-r1. Ver página 9.

[35] 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 9.

[36] 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 9.

[37] 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 9.

[38] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. Ver página 9.

[39] 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 9.

[40] 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 9.

[41] Eirini Kalliamvakou. Research: quantifying GitHub copilot's impact on developer productivity and happiness. Ver página 10.

[42] GitHub copilot features. Ver página 10.

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

[44] sourcegraph/sourcegraph-public-snapshot: Code AI platform with code search & cody. Ver página 10.

[45] Kamal Acharya. Devin: A cautionary tale of the autonomous AI engineer. Ver página 10.

[46] Building a better repository map with tree sitter. Ver página 10.