



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE MÁSTER

Desarrollo de los modelos de LLM adaptados a contextos locales. Una aproximación práctica al ámbito de la defensa.

Máster Universitario en Dirección TIC para la Defensa

ALUMNO: Francisco José Ochando Terreros

DIRECTORES: Luis Álvarez Sabucedo

CURSO ACADÉMICO: 2025-2026

UniversidadeVigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE MÁSTER

Desarrollo de los modelos de LLM adaptados a contextos locales. Una aproximación práctica al ámbito de la defensa.

Máster Universitario en Dirección TIC para la Defensa
Especialidad de Sistemas y Tecnologías de la Información

UniversidadeVigo

RESUMEN

El uso de modelos de inteligencia artificial generativa, y en particular de los modelos de lenguaje de gran escala (LLM), representa un avance relevante en el desarrollo de sistemas de apoyo a la toma de decisiones en el ámbito militar, tal como refleja el estado del arte. En este trabajo se realiza un análisis de las principales tecnologías de agentes basadas en modelos de lenguaje generativos orientadas al apoyo a la decisión en unidades de pequeña entidad, considerando aspectos clave como el tamaño del modelo, la capacidad de integración de herramientas externas y la multimodalidad.

Asimismo, se estudian los principales protocolos propuestos para la comunicación y orquestación de agentes inteligentes, así como distintas arquitecturas fundamentadas en Retrieval-Augmented Generation (RAG). Los resultados experimentales obtenidos muestran que el empleo de modelos de lenguaje pequeños (SLM), combinados con herramientas externas, permite abordar tareas de mayor complejidad que aquellas que resultarían inviables o imprecisas si se resolvieran únicamente mediante modelos de lenguaje sin soporte instrumental. El análisis de los protocolos MCP y ACP evidencia su carácter complementario, más que excluyente, en el diseño de sistemas multiagente.

Finalmente, se propone una arquitectura de agentes basada en SLM para su despliegue en dispositivos personales del combatiente. Dicha arquitectura se complementa con servidores locales privados en entornos de misión, capaces de realizar tareas de recuperación de información, análisis automático de imágenes y generación de informes de inteligencia mediante LLM multimodales, a partir de la ingesta de datos procedentes de diversas aplicaciones, con el objetivo de proporcionar apoyo a la toma de decisiones en unidades de pequeña entidad.

PALABRAS CLAVE

Modelos de Lenguaje, Agentes, Recuperación, Inteligencia Artificial, información privada

AGRADECIMIENTOS

A mi familia, por su paciencia conmigo y mis ausencias.

CONTENIDO

Acrónimos	3
Índice de Figuras	4
Índice de Tablas.....	5
1 Introducción y objetivos	6
1.1 Introducción	6
1.2 Motivación	6
1.3 Objetivos	7
2 Estado del arte	8
2.1 Introducción	8
2.2 Soporte en la toma de decisiones estratégicas	8
2.3 Ciberseguridad, detección y respuesta a incidentes	8
2.4 Análisis de inteligencia	8
2.5 Automatización de comunicaciones y coordinación.....	9
2.6 Simulación y entrenamiento de personal militar.....	9
2.7 Desarrollo de estrategias en guerra electrónica.....	10
2.8 Monitoreo y evaluación de operaciones de paz y humanitarias.....	11
2.9 Desarrollo de contramedidas y defensa en el campo de batalla.....	11
3 Desarrollo de la propuesta	12
3.1 Marco Tecnológico	12
3.1.1 Modelos de lenguaje de gran escala	12
3.1.2 Marcos y motores de inferencia de Modelos	17
3.1.3 Inteligencia Artificial aplicada a agentes.....	18
3.1.4 Sistemas multiagente con Inteligencia Artificial generativa	19
3.1.5 Generación aumentada por recuperación.....	24
3.1.6 Bases de datos vectoriales	26
3.2 Metodología y desarrollo	28
3.2.1 Uso de modelos de lenguaje en internet con herramientas en entorno local	28
3.2.2 Selección de modelos en la nube	29
3.2.3 Selección de modelos de uso local	30
3.2.4 Ventajas y desventajas de los modelos en local	31
3.2.5 Inferencia de modelos en local	32
3.2.6 Uso de herramientas (agentes).....	32

3.2.7 Desarrollo de agentes inteligentes	33
3.2.8 Desarrollo de un servidor de agentes MCP	33
3.2.9 Desarrollo de un servidor de agentes ACP	34
3.2.10 Desarrollo de un cliente de agentes MCP	35
3.3 Desarrollo de una arquitectura de agentes habilitada con SLM y LLM locales	38
3.3.1 Introducción	38
3.3.2 Objetivos de la arquitectura	38
3.3.3 Búsqueda de información en documentación clasificada	39
3.3.4 Búsqueda de información y descripción de imágenes	40
3.3.1 Transcripción y traducción de textos	40
4 Resultados y discusión	41
4.1 Cliente local con agentes locales y modelos en internet	41
4.2 Cliente local con agentes locales y modelos locales	43
4.2.1 Modelos de lenguaje en modo texto	43
4.2.2 Modelos de lenguaje en modo imagen	45
4.2.3 Prueba de un servidor MCP local	49
4.2.4 Prueba de un servidor ACP local	52
5 Conclusiones y líneas futuras	54
5.1 Conclusiones y lecciones aprendidas	54
5.2 Líneas futuras	55
6 Bibliografía.....	56
Anexo I: Despliegue de modelos LLM/SLM con ollama	60
Anexo II: Servidores de agentes y RAG	62

ACRÓNIMOS

AaaS	“Applications as a Service”. Aplicación como un Servicio
ACP	“Agent Communication Protocol”. Protocolo de comunicación de agentes.
ANN	“Artificial Neural Network”. Red neuronal artificial.
API	“Application programming interface”. Interfaz de programación de aplicaciones
BERT	“Bidirectional Encoder Representation for Transformer”.
CLI	“Command line interface”. Interfaz de línea de comando.
GPT	“Generic Pretrained Transformer”
GGUF	“Generic GPT Unified format”.
GUI	“Graphical User interface”. Interfaz gráfica de usuario.
HTTP	“HiperText Transport Protocol”. Protocolo de transporte de hipertexto
IP	“Internet Protocol”. Protocolo de Internet
JSON	“JavaScript Object Notation”. Notacion de objeto Javascript
LLM	“Large Language Model”. Modelo de lenguaje extenso
MCP	“Model Context Protocol”. Protocolo de contexto de modelo
NLP	“Natural Language Proccess”. Procesamiento de Lenguaje Natural
OODA	Observar-Orientar-Decidir-Actuar
OPORD	“Operation Ord”. Orden de Operaciones
PC	“Personal Computer”. Computador Personal
RAG	“Retrieval-Augmented Generation”. Generación con recuperación aumentada
RG	“Report Generation”. Generación de Informes
RPC	“Remote Procedure Call”. Llamada a procedimiento remoto
SaaS	“Software as a Service”. Software como Servicio.
SDK	“Software Development Kit”.
SLM	“Short Language Model”. Modelo de lenguaje pequeño
SO	Sistema Operativo

ÍNDICE DE FIGURAS

Figura 2-1. Arquitectura de agentes para crear una orden de operación	10
Figura 3-1. Neurona natural vs Neurona artificial.....	13
Figura 3-2. Arquitectura del transformador. Fuente: [12].....	14
Figura 3-3. Visualización cronológica del lanzamiento de LLM.....	15
Figura 3-4. Evolución de los modelos de lenguaje hacia modelos más pequeños.	16
Figura 3-5. Orquestación de agentes en sistemas multiagente.	19
Figura 3-6. Arquitectura de un Sistema de Agentes con Modelos de Lenguaje.	21
Figura 3-7. Estructura de A2A. Fuente: [30].....	22
Figura 3-8. Orquestación de agentes en ACP con agentes enrutadores	24
Figura 3-9. Proceso básico de un RAG.	25
Figura 3-10. Proceso detallado de recuperación de información con inteligencia artificial generativa	26
Figura 3-12. Espacio de una Base de Datos vectorial. Fuente: Wikipedia.....	27
Figura 3-13. Frontal de la aplicación Claude Desktop utilizada para probar modelos en la nube ...	30
Figura 3-14. Agente inteligente (LLM + Servidor MCP)	34
Figura 3-15. Arquitectura de un servicio cliente servidor con ACP	35
Figura 3-16. Interacción modelo-cliente-servidor. Fuente: Anthropic.....	36
Figura 3-17. Servicio de agentes con servidor MCP y cliente Ollmcp	37
Figura 3-18. Arquitectura de la aplicación de un agente inteligente con un cliente Streamlit.....	38
Figura 3-19. Sistema de agentes locales con SLM y LLM propuesto	39
Figura 4-1. Instrucción en Claude desktop con herramienta de un servidor local	41
Figura 4-2. Respuesta de Claude con una herramienta de un servidor local.....	42
Figura 4-3. Imagen insertada en el modelo junto a la solicitud de descripción.	45
Figura 4-4. Resultado del prompt.....	46
Figura 4-5. Solicitud al modelo de información detallada	46
Figura 4-6. Imagen de entrada al modelo	47
Figura 4-7. Imagen de dos letreros y respuesta del modelo.	48
Figura 4-8. Imagen de letrero en autovía y respuesta del modelo.....	49
Figura 4-9. Interfaz del cliente Ollmcp.	50
Figura 4-10. Respuesta de un agente LLM+herramienta de cálculo de un círculo.....	52

ÍNDICE DE TABLAS

Tabla 3-1. Tabla de modelos pequeños y grandes para uso local	16
Tabla 3-2. Características de los marcos de inferencia más conocidos.....	18
Tabla 3-3. Primitivas del protocolo de comunicación de Agentes.....	23
Tabla 3-4. Modelos de IA generativa SLM y LLM utilizables en un entorno local.	31
Tabla 3-5. Interfaces disponibles en los paquetes de implementacion MCP	33
Tabla 4-1. Archivo de configuración con los servidores de herramientas MCP disponibles.....	42
Tabla 4-2. Modelos ejecutados con instrucciones de generación de texto.....	44
Tabla 4-3. Resultados de las pruebas de detección de matrículas.....	48
Tabla 4-4. Resultados de las pruebas realizadas con herramientas y varios modelos SLM	51
Tabla 4-5. Métodos de comunicación que utiliza ACP.....	53
Tabla AI-1. Crear entorno virtual con Conda.....	60
Tabla AI-2. Descarga del framework Ollama con Curl.	60
Tabla AI-3. Descarga de modelos con Ollama y listado de modelos locales.....	60
Tabla AI-4. Arranque del framework Ollama con el Modelo Salamadra.	60
Tabla AI-5. Mostrar las características del modelo en Ollama	61
Tabla AII-1. Instalación de modulos mcp y fastmcp	62
Tabla AII-2. Crear un servidor fastmcp con conexión HTTP	62
Tabla AII-2. Configurar un servidor mcp para un sistema de fichero compartido	62
Tabla AII-3. Ejecución síncrona de un agente con Curl	63
Tabla AII-4. Desarrollo de una App con Streamlit.	65
Tabla AII-5. Resultado de la solicitud de listado de agentes mediante la API	66
Tabla AII-6. Crear y activar un entorno virtual para la aplicación	67
Tabla AII-7. Instalar Chroma, LangChain y otras dependencias	67
Tabla AII-9. Funciones de pregunta (query) y búsqueda (retriever) para el LLM.....	68
Tabla AII-10. Código del proceso de incrustaciones en una Base de Datos vectorial.....	70

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

En la actualidad, el uso de modelos de lenguaje de gran escala (*Large Language Model* o LLM, en terminología inglesa) se ha popularizado rápidamente en todo tipo de sectores, desde la educación hasta la industria, lo que demuestra su potencial transformador en la automatización de tareas, el análisis y la búsqueda de información. El uso de LLM en entornos locales surge como respuesta a la necesidad de aprovechar su enorme capacidad de procesamiento del modelo de lenguaje de manera más segura para los datos, ágil y personalizada. Ejecutarlos de forma local permite reducir la dependencia de servicios en la nube, disminuir los tiempos de respuesta y proteger mejor la información sensible, algo especialmente importante en empresas, instituciones científicas y entornos sanitarios, alcanzando una verdadera soberanía digital. Al mismo tiempo, trabajar con LLM en servidores propios facilita su adaptación a necesidades concretas, entrenándolos o ajustándolos con datos especializados sin incurrir en grandes costes de comunicación externa. Así, los LLM locales se presentan como aliados estratégicos para innovar y mejorar procesos en un mundo cada vez más orientado a la inteligencia artificial.

Los LLM han surgido como sistemas de inteligencia artificial de vanguardia capaces de procesar y generar texto con una comunicación coherente y generalizable a múltiples tareas. Además de una mayor generalización y adaptación al dominio, los LLM de nueva generación parecen tener nuevas capacidades, como razonamiento, planificación, toma de decisiones, aprendizaje en contexto o respuesta en entornos sin entrenamiento previo. Estas capacidades parecen emerger del factor de escala, incluso cuando los LLM pre-entrenados, no están entrenados específicamente para poseer estos atributos. Sin embargo, los LLM pre-entrenados pueden ser objeto de alucinaciones y obtienen peores resultados en entornos sin datos previos que en entornos con un contexto previo. Ajustarlos con datos propios y alinearlos con las necesidades particulares mejora la generalización en tareas desconocidas, mejorando significativamente el rendimiento sin entrenamiento previo.

El uso de herramientas mejora la respuesta de un LLM, porque amplía sus capacidades más allá de la generación puramente estadística de texto, permitiéndole interactuar de forma estructurada con fuentes externas de información y servicios computacionales.

Se espera que las capacidades de los agentes para razonar y utilizar herramientas los conviertan en la base de la próxima era de las comunicaciones. La comunicación entre agentes ha dado un salto cualitativo, pasando de la interconexión de dispositivos a la colaboración entre entidades inteligentes. A diferencia de etapas anteriores en la comunicación, los nuevos sujetos de comunicación son agentes impulsados por LLM con capacidades de percepción, memoria, razonamiento y toma de decisiones autónomas [1].

1.2 Motivación

En el ámbito de la Defensa, el empleo de LLM locales se considera particularmente relevante al ofrecer capacidades avanzadas de análisis, soporte a la toma de decisiones y apoyo operativo bajo estrictos criterios de seguridad y confidencialidad. Estos modelos pueden aplicarse en la interpretación y síntesis de información proveniente de múltiples fuentes, en la automatización de informes de inteligencia, o en el desarrollo de asistentes virtuales especializados que apoyen a mandos y analistas en tiempo real. Asimismo, su despliegue local garantiza que datos estratégicos y altamente sensibles no salgan de la infraestructura propia, reforzando la soberanía y la protección frente a ciber amenazas.

El presente trabajo se plantea como respuesta a la necesidad de consolidar los fundamentos teóricos y prácticos de los modelos de lenguaje basados en inteligencia artificial generativa, así como de llevar a cabo una prospectiva tecnológica sobre los sistemas de agentes sustentados en LLM y modelos de lenguaje pequeños (SLM). En este marco, se abordan de manera específica los desafíos asociados a su

despliegue en entornos locales, con el objetivo de facilitar el desarrollo de soluciones adaptadas a contextos y requisitos particulares.

La falta de un conocimiento claro acerca del alcance y las posibilidades de esta tecnología ha derivado en la realización de pruebas cuya aplicación resulta compleja, principalmente debido a limitaciones de distinta naturaleza, entre las que destacan los requerimientos de hardware, la necesidad de diseñar herramientas especializadas para tareas concretas, la adecuada selección y dimensionamiento del modelo, y las restricciones asociadas al despliegue en infraestructuras de tipo nube de combate.

1.3 Objetivos

Se considera pertinente realizar un trabajo de recopilación, análisis y sistematización de técnicas, complementado con la ejecución de pruebas de concepto, que permita abordar el diseño de una propuesta arquitectónica orientada a facilitar la implementación de modelos de referencia para el despliegue eficiente y seguro de agentes inteligentes.

En este contexto, el objetivo del presente Trabajo de Fin de Máster consiste en estudiar y diseñar una metodología dirigida a la implementación de modelos de inteligencia artificial generativa en entornos locales, enmarcados en arquitecturas basadas en agentes y aplicados al ámbito de la Defensa.

Para ello se han identificado los siguientes objetivos parciales:

- I. Objetivo parcial 1.- Revisar la tecnología disponible y evaluar alternativas
- II. Objetivo parcial 2.- Analizar el uso de modelos multimodales en entornos locales
- III. Objetivo parcial 3.- Probar herramientas orientadas a cálculos para el desarrollo de agentes inteligentes autónomos.
- IV. Objetivo parcial 4.- Analizar y desarrollar un servicio de ejecución de herramientas mediante agentes inteligentes con SLM y LLM. Integrar ambos enfoques en dispositivos locales (personales y en red), con el propósito de dotar a las aplicaciones capacidades incrementadas en el apoyo a decisiones, predicción de escenarios, detección de amenazas y automatización de determinados procesos operativos del soldado.
- VI. Objetivo parcial 5.- Evaluar la solución propuesta.

2 ESTADO DEL ARTE

2.1 Introducción

Los modelos de lenguaje de gran escala (LLM), en combinación con técnicas de Recuperación Aumentada de Información (Retrieval-Augmented Generation, RAG) e integrados dentro del paradigma de agentes inteligentes, ofrecen un conjunto de aplicaciones de elevado interés en el ámbito de la Defensa.

Las organizaciones de Defensa se caracterizan por su enorme carga burocrática. En esta burocracia, muchas oficinas se encargan de recopilar información y extraer conclusiones para enviarlas al siguiente nivel de la jerarquía. Otras se encargan de recopilar información, proponer líneas de acción para que los comandantes tomen decisiones y difundir estas decisiones mediante documentación escrita. Cuando estas tareas las realizan exclusivamente seres humanos, resultan increíblemente lentas y engorrosas. Una gran parte de los datos de que disponen las organizaciones en materia de seguridad nacional no están estructurados, y suelen presentarse en forma de texto o imágenes. Las prácticas actuales requieren que una persona ingiera, resuma y extraiga información de estos datos, lo que limita la toma de decisiones a la velocidad de la comprensión humana. Por lo tanto, el resumen automático basado en LLM puede ser especialmente adecuado para aplicaciones de seguridad nacional [2].

A continuación, se muestran una serie de campos para la aplicación de LLM en el ámbito de Defensa:

2.2 Soporte en la toma de decisiones estratégicas

Simulación y modelado de escenarios: Los agentes impulsados por LLM y RAG pueden simular diferentes escenarios de defensa, permitiendo a los planificadores evaluar múltiples estrategias de manera eficiente. En [3] proponen un marco inteligente de simulación de guerra, que integra modelos de lenguaje grandes (LLM) con gemelos digitales de campos de batalla. Mediante la simulación en tiempo real y la fusión de datos de múltiples fuentes, aprovechan las capacidades de interacción en lenguaje natural de los LLM para reducir el umbral de mando y permitir el mando autónomo a través del ciclo Observar-Orientar-Decidir-Actuar (OODA).

2.3 Ciberseguridad, detección y respuesta a incidentes

Permiten analizar grandes volúmenes de registros de seguridad y detectar patrones anómalos que puedan indicar un ciberataque, agilizando la respuesta. También permiten automatizar tareas de defensa cibernética: Agentes impulsados por LLM pueden automatizar tareas de monitorización, detección y respuesta, protegiendo redes y sistemas críticos. El estudio realizado en [4] examina la bibliografía existente y ofrece una caracterización exhaustiva de las aplicaciones defensivas y adversarias de los LLM en el ámbito de la ciberseguridad.

2.4 Análisis de inteligencia

Los LLM+RAG pueden extraer información de grandes volúmenes de datos y analizar rápidamente datos no estructurados, como informes, comunicaciones y datos de inteligencia, para identificar patrones, amenazas o actores relevantes. En el estudio realizado en [5] utilizan un conjunto de datos previamente etiquetados por expertos humanos, con el que tratan de demostrar resultados de la extracción de inteligencia impulsada por la IA para la generación de informes, agilizando las operaciones de ciberseguridad y proporcionar inteligencia sobre amenazas precisa y oportuna. Sin embargo, sugiere la importancia de los enfoques híbridos necesarios para lograr la precisión de la anotación humana.

2.5 Automatización de comunicaciones y coordinación

Generación automática de informes y resúmenes: Los LLM pueden generar informes detallados o resúmenes breves de eventos en tiempo real, facilitando la comunicación y la gestión de información entre unidades o comandos de defensa. El trabajo realizado en [6] se centra en la generación de informes (RG), basado en un RAG que tiene como objeto producir una respuesta extensa y con citas atribuidas a una consulta compleja del usuario. Se distingue por dos características. En primer lugar, la RG pone en primer plano la identidad del solicitante: la misma consulta debería generar informes diferentes para solicitantes con distintos niveles de formación o experiencia en el ámbito. En segundo lugar, el informe ideal representa un resumen de todo el corpus de la información más crítica para el usuario, haciendo hincapié en la cobertura, mientras que los asistentes pregunta-respuesta (QA) tradicionales enfatizan la mera adecuación de la respuesta.

Asistentes virtuales del mando: Los agentes pueden ayudar a los líderes militares a coordinar operaciones, proporcionando recomendaciones basadas en datos actualizados sobre terreno, clima, y amenazas potenciales.

2.6 Simulación y entrenamiento de personal militar

Creación de escenarios de entrenamiento dinámicos: Los LLM y RAG pueden ser usados para crear escenarios realistas para entrenar a soldados y oficiales, ajustando las situaciones según el rendimiento de los participantes en tiempo real [7]. El artículo [8] presenta un marco de razonamiento multimodal y multi-agente que aprovecha los modelos de lenguaje grandes (LLM) para generar escenarios de entrenamiento en operaciones críticas, como las órdenes de operaciones (OPORD). El marco se basa en un agente orquestador, que gestiona el proceso de generación general. El orquestador interactúa con una colección de agentes auxiliares, especializados en un dominio concreto. El orquestador ejecuta tareas a través de un proceso de razonamiento paso a paso, delegando subtareas a los agentes auxiliares pertinentes según sea necesario e integrando sus respuestas para generar resultados coherentes y sensibles al contexto mediante GPT-4o.

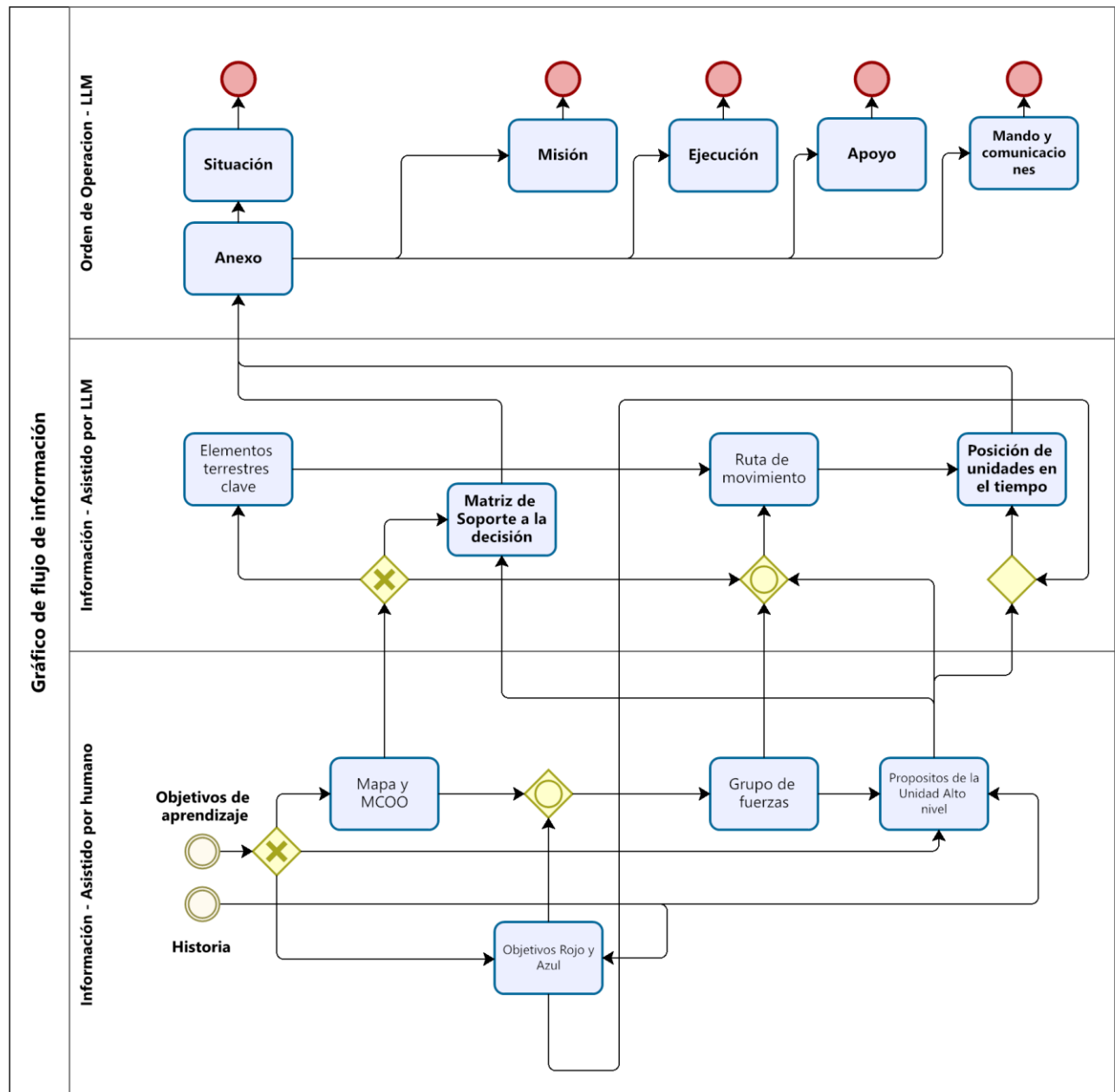


Figura 2-1. Arquitectura de agentes para crear una orden de operación

2.7 Desarrollo de estrategias en guerra electrónica

Análisis de señales de radio y comunicaciones: Los LLM, junto con RAG, pueden analizar grandes volúmenes de datos de comunicaciones electrónicas, identificando señales de interés y proporcionando información relevante para la guerra electrónica. Los estudios teóricos, el presentado en [9], muestran la viabilidad de incorporar LLM en sistemas EW. Las pruebas empíricas realizadas confirman que la utilización de LLM optimiza la adaptabilidad del sistema frente a los cambios en los modos de transmisión del adversario. Adicionalmente, la implementación de RAG reduce los tiempos de latencia inherentes a la introducción de nueva información para su análisis. Los resultados muestran que el sistema mitiga la carga del personal, incrementando la probabilidad de detección de amenazas.

Generación de contramedidas: Los agentes pueden proponer contramedidas o estrategias de interferencia basadas en el análisis de señales y la inteligencia obtenida.

2.8 Monitoreo y evaluación de operaciones de paz y humanitarias

Análisis de situaciones en zonas de conflicto: Los LLM+RAG pueden ayudar a procesar información de diversas fuentes (informes, comunicaciones locales, noticias) para ofrecer una visión clara de la situación en áreas de conflicto, apoyando la toma de decisiones durante misiones de paz o de ayuda humanitaria.

Optimización de recursos: Los agentes pueden ayudar a gestionar y distribuir recursos de manera más eficiente durante misiones de apoyo, garantizando una respuesta más rápida y organizada.

2.9 Desarrollo de contramedidas y defensa en el campo de batalla.

En el campo de batalla se pueden ejecutar diversas tareas relacionadas con la defensa y las contramedidas:

- En el análisis de tácticas de guerra no convencional, los modelos pueden analizar patrones de comportamiento en conflictos asimétricos (como insurgencias o terrorismo) y proporcionar recomendaciones de respuesta adecuadas.
- La detección de patrones de actividad de grupos extremistas: Los LLM, junto con RAG, pueden analizar datos de comunicaciones, redes sociales y otras fuentes abiertas para detectar signos de actividad terrorista o insurgente.
- Automatización de respuestas a amenazas en el campo: Los agentes pueden estar diseñados para reaccionar a eventos en tiempo real, como ataques enemigos, y generar respuestas automatizadas o sugerir estrategias defensivas basadas en la situación del momento.
- Simulaciones de combate basadas en IA: Los LLM, en combinación con RAG y agentes, pueden crear simulaciones de combate realistas, ayudando a predecir el resultado de enfrentamientos y a optimizar las estrategias defensivas.
- Desarrollo de contramedidas y defensa en el campo de Batalla. Defensa contra Desarrollo de contramedidas y defensa en el campo de Batalla.
- Defensa contra desinformación y verificación: Los modelos pueden ayudar a detectar campañas de desinformación o verificar la veracidad de la información en tiempo real, proporcionando un análisis de confianza

3 DESARROLLO DE LA PROPUESTA

3.1 Marco Tecnológico

3.1.1 Modelos de lenguaje de gran escala

3.1.1.1 Descripción

Los LLM son modelos basados en el aprendizaje profundo que utilizan muchos parámetros para aprender a partir de grandes cantidades de texto sin etiquetar. Los LLM son una clase de modelos fundacionales, que están entrenados sobre enormes cantidades de datos para proporcionar capacidades avanzadas en el procesamiento del lenguaje natural (PLN o NLP, por sus siglas en inglés), entre otras cosas. Los modelos fundacionales están diseñados teniendo en cuenta el aprendizaje por transferencia, lo que significa que pueden aplicar eficazmente los conocimientos adquiridos durante el pre entrenamiento a tareas nuevas y relacionadas. Esta transferencia de conocimientos mejora su adaptabilidad, lo que les permite dominar rápidamente nuevas tareas con relativamente poco entrenamiento adicional [10].

Pueden realizar diversas tareas de procesamiento del lenguaje natural, como reconocer, resumir, traducir, predecir y generar texto. El artículo [11] ofrece una visión general sobre una amplia gama de conceptos relacionados con los LLM. En él se analizan los conceptos básicos más relevantes y abarca los temas avanzados en la vanguardia de la investigación sobre los LLM. El artículo pretende ser una referencia rápida y completa para que los investigadores y profesionales puedan avanzar en la investigación sobre los LLM.

3.1.1.2 Fundamentos

Los LLM son un tipo particular de redes neuronales artificiales (RNA o ANN, por sus siglas en inglés): modelos computacionales inspirados en la estructura y el funcionamiento del cerebro humano. El componente básico de una ANN es la neurona artificial, también conocida como nodo o unidad se muestra en la Figura 3-1. Estas neuronas se organizan en capas, y las conexiones entre ellas se ponderan para representar la fuerza de la relación entre ellas. Esas ponderaciones representan los parámetros del modelo que se optimizarán durante el proceso.

Como las ANN son, por definición, modelos matemáticos que funcionan con datos numéricos, cuando se trata de datos textuales no estructurados, como en el contexto de los LLM, hay que realizar dos actividades fundamentales para preparar los datos como entrada del modelo:

- Tokenización: Es el proceso de dividir un fragmento de texto en unidades más pequeñas llamadas tokens. Estos tokens pueden ser palabras, sub-palabras o incluso caracteres, dependiendo del esquema o algoritmo de tokenización elegido. El objetivo es crear una representación estructurada del texto que pueda ser fácilmente procesada.
- Incrustación o embedding, en terminología inglesa: Una vez tokenizado, cada token se convierte en un vector numérico denso llamado incrustación. Estas incrustaciones se aprenden durante el entrenamiento del modelo de lenguaje y capturan las relaciones semánticas entre los tokens. La representación numérica permite al modelo realizar operaciones matemáticas sobre los tokens y comprender el contexto en el que aparecen.

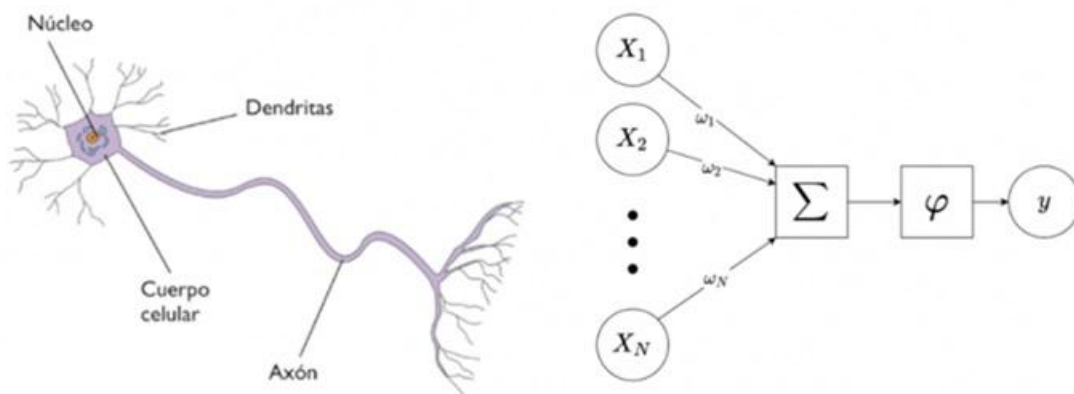


Figura 3-1. Neurona natural vs Neurona artificial

En los modelos convolucionales, el número de operaciones necesarias para relacionar señales de dos posiciones de entrada o salida arbitrarias crece con la distancia entre las posiciones. Esto dificulta el aprendizaje de las dependencias entre posiciones distantes en una conversación. En el trabajo realizado en [12] propusieron el “Transformer” cuya arquitectura se muestra en la Figura 3-2, como una arquitectura de modelo que evita la recurrencia y se basa completamente en un mecanismo de atención para establecer dependencias globales entre la entrada y la salida. La auto atención, a veces llamada intra-atención, es un mecanismo de atención que relaciona diferentes posiciones de una sola secuencia con el fin de calcular una representación de la secuencia.

El transformador está compuesto de dos estructuras, un codificador utilizado en arquitecturas BERT (Bidirectional Encoder Representations from Transformers en inglés), presentado por Google en 2018. BERT [13] está preentrenado con grandes cantidades de datos de texto y se puede ajustar para una amplia gama de tareas de PLN. La arquitectura del decodificador se utiliza para tareas como la generación de lenguaje. El decodificador toma una representación vectorial de tamaño fijo del contexto y la utiliza para generar una secuencia de palabras una a una, condicionando cada palabra a las palabras generadas anteriormente. Uno de los modelos de decodificador transformador más populares fue el GPT-3 (Generative Pre-trained Transformer 3), que fue presentado por OpenAI en 2020.

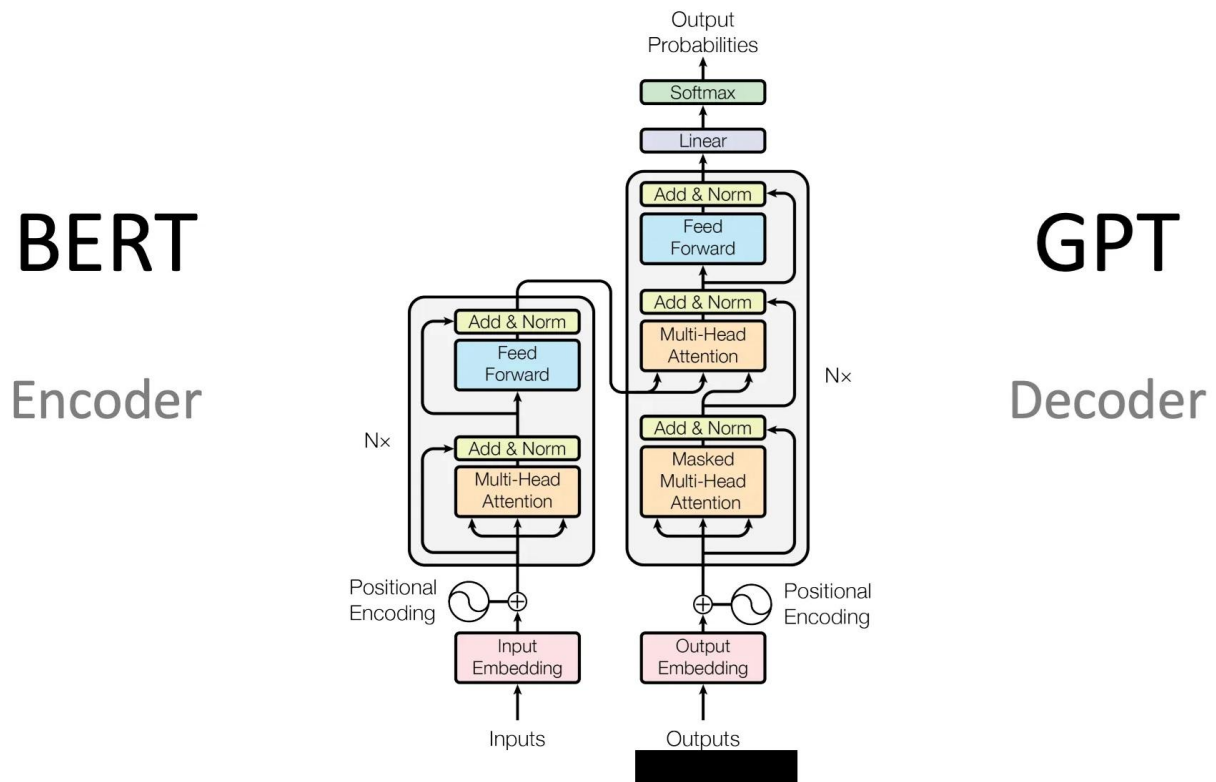


Figura 3-2. Arquitectura del transformador. Fuente: [12]

La calidad de las respuestas de los LLM sin datos contextuales es relativamente baja, como se demuestra en [14] para un sistema aplicable en “Smart Cities”. El estudio demuestra que el sistema sin contexto obtiene puntuaciones de 0,3-0,38. Sin embargo, el sistema con acceso al contexto adecuado, mejora sus resultados oscilando entre 0,68 y 0,74. La ventana de contexto en un LLM, por tanto, puede considerarse como el equivalente a su memoria de trabajo. Determina la duración de una conversación que puede mantener sin olvidar los detalles de lo que ha generado anteriormente. También determina el tamaño máximo de los fragmentos de documento o muestras de código que puede procesar a la vez. Cuando una indicación, conversación, documento o base de código supera la ventana de contexto de un modelo de inteligencia artificial, debe truncarse o resumirse para que el modelo pueda continuar. En términos generales, aumentar el tamaño de la ventana de contexto de un LLM proporciona una mayor precisión, menos alucinaciones, respuestas más coherentes del modelo, conversaciones más largas y una mayor capacidad para analizar secuencias de datos más largas. Sin embargo, aumentar la longitud del contexto a menudo conlleva un aumento de los requisitos de potencia computacional y, por lo tanto, un aumento de los costes.

El ajuste de los hiper parámetros es un factor clave que influye en el rendimiento de los LLM. Por ejemplo, el muestreo Top-K selecciona el siguiente token de entre los K candidatos más probables, mientras que el muestreo Top-P toma muestras del conjunto más pequeño de tokens cuya probabilidad acumulativa supera P. Por otro lado, la temperatura T [15], es uno de los hiper-parámetros más utilizados. Durante la inferencia en los LLM, este parámetro se utiliza para escalar los logits de la capa de salida, controlando eficazmente la aleatoriedad de las predicciones del modelo.

3.1.1.3 Modelos de Lenguaje

Existen muchos factores a tener en cuenta a la hora de elegir un LLM para su aplicación. Esos factores también deben dividirse en dos escenarios: LLM propietarios y LLM de código abierto. A continuación, se indican algunos factores y ventajas e inconvenientes a tener en cuenta a la hora de elegir un LLM:

- Tamaño y precisión en las respuestas
- Coste y estrategia de hosting
- Coste por el consumo del modelo (coste por token)
- Coste por el hosting del modelo (almacenamiento en nube)

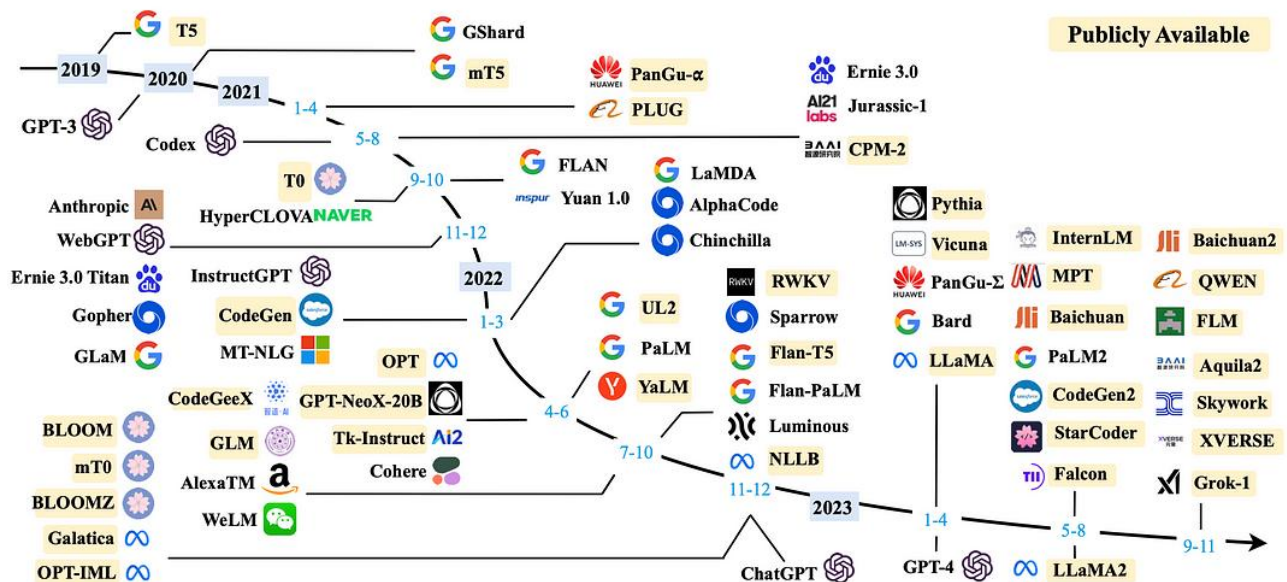


Figura 3-3. Visualización cronológica del lanzamiento de LLM.

En la Figura 3-3 se muestran cronológicamente el lanzamiento de numerosos modelos LLM. Las cajas azules representan los modelos pre-entrenados, mientras que las cajas sepia corresponden a modelos ajustados mediante instrucción.

A medida que evoluciona la inteligencia artificial, las empresas, investigadores y desarrolladores se están dando cuenta que no siempre es mejor el uso de grandes modelos de lenguaje. La era de los modelos de tamaño cada vez mayor está dando paso a modelos más eficientes y compactos, denominados modelos de lenguaje pequeños (SLM). Los SLM ofrecen varias ventajas clave que abordan tanto la creciente complejidad de la IA como los retos prácticos que plantea la implementación de modelos a gran escala. Estos modelos requieren menos recursos computacionales, ofrecen tiempos de inferencia más rápidos y tienen el potencial de ejecutarse directamente en dispositivos embebidos. Esta evolución se muestra en la Figura 3-4 [16].

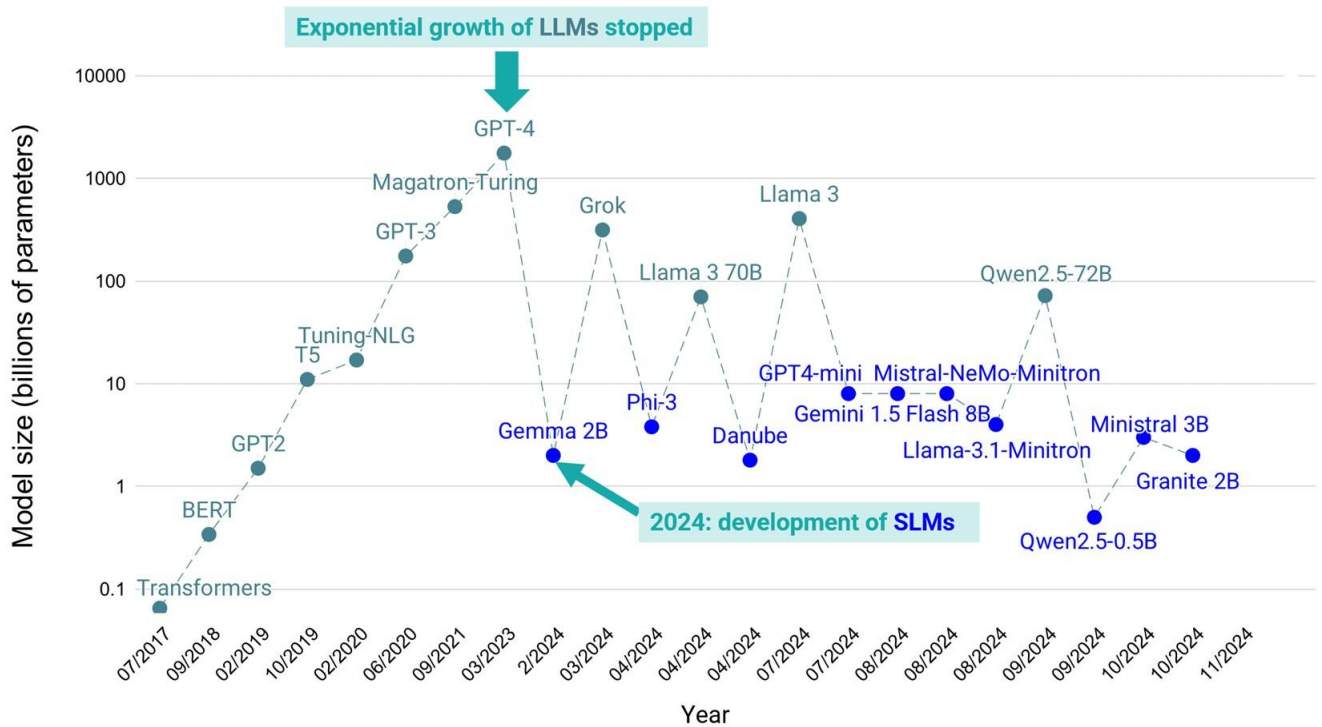


Figura 3-4. Evolución de los modelos de lenguaje hacia modelos más pequeños.

La Tabla 3-1, cuyos datos fueron actualizados a 1 de noviembre de 2025, muestra una lista de modelos descargables para la ejecución en un entorno local. Para su clasificación en modelos grandes y pequeños, se ha considerado el tamaño del modelo expresado en miles de millones de parámetros (billions en terminología anglosajona). Se han definido como LLM aquellos modelos con más de 3B de parámetros, mientras que los modelos con menos de 3B se han categorizado como SLM. Esta distinción resulta relevante, ya que determina la viabilidad de ejecutar un modelo en un entorno local basado únicamente en CPU. También se ha identificado si el modelo es capaz de utilizar herramientas, lo que resultará útil para la definición de agentes basados en SLM/LLM.

Modelo	Parámetros	Tamaño	Necesita GPU	Herramientas
Mistral-small3.2	24B	LLM	Si	Si
Deepseek-r1	7B	LLM	Si	Si
Salamandra-2b-instruct	2B	SLM	No	No
Salamandra-7b-instruct	7B	LLM	Si	No
Gemma3:270m	270M	SLM	No	No
Gemma3:1b	1B	SLM	No	No
Gemma3n:e2b	2B	SLM	No	No
GPT-oss	20B	LLM	Si	Si
Qwen2.5:0.5	0.5B	SLM	No	Si
Qwen2.5:1.5	1.5B	SLM	No	No

Tabla 3-1. Tabla de modelos pequeños y grandes para uso local

3.1.1.4 Riesgos de los modelos de lenguaje

El uso de modelos de lenguaje extenso no está exento de riesgos. En el estudio realizado en [17], se evaluaron los riesgos de uso de los LLM y sus vulnerabilidades. Clasificaron cinco tipos de ataque con ayuda de LLM, según su ubicación dentro de la infraestructura del sistema de inteligencia artificial. Estas categorías abarcan:

- Ataques a nivel de hardware: Los ataques de hardware suelen implicar el acceso físico a los dispositivos. Sin embargo, los LLM no pueden acceder directamente a los dispositivos físicos, aunque pueden acceder a la información asociada al hardware. El ataque de canal lateral [18], [19] es un tipo de ataque que puede ser impulsado por los LLM. Los ataques de canal lateral suelen implicar el análisis de la fuga involuntaria de información de un sistema físico o una implementación.
- Ataques a nivel de SO: Los LLM operan a un alto nivel de abstracción y carecen del acceso de bajo nivel al sistema necesario para ejecutar ataques a nivel del sistema operativo. No obstante, pueden utilizarse para el análisis de la información recopilada de los sistemas operativos, lo que podría facilitar la ejecución de ataques.
- Ataques a nivel de Software: El caso de uso más frecuente a nivel de software es el de desarrolladores maliciosos que utilizan LLM para crear malware.
- Ataques a nivel de red: Accesos no autorizados a los servidores de modelos y ataques de denegación de servicio.
- Ataques a nivel de usuario: El reciente debate se ha centrado principalmente en los ataques a nivel de usuario, ya que los LLM demuestran su capacidad para crear contenidos muy convincentes, pero en última instancia engañosos, así como establecer conexiones entre datos aparentemente inconexos.

Igual de importantes son las vulnerabilidades de los LLM contra ataques y amenazas externas, que se derivan de su propia naturaleza y arquitectura, teniendo en cuenta que los LLM son modelos de IA. Por ejemplo, los atacantes pueden manipular los datos de entrada para generar resultados incorrectos o indeseables del LLM.

3.1.2 Marcos y motores de inferencia de Modelos

Un marco de inferencia y servicio de LLM se refieren al proceso de implementar modelos de lenguaje grandes y hacerlos accesibles para su uso, ya sea a nivel local para proyectos personales o en producción para aplicaciones a gran escala. Dependiendo de sus necesidades, puede optar por una implementación local ligera o una solución robusta de nivel empresarial [20].

Un motor de inferencia constituye un componente de software especializado, concebido para la ejecución de modelos de lenguaje y la subsiguiente generación de respuestas a partir de instrucciones (“prompts”). Su diseño se focaliza exclusivamente en la fase operativa, específicamente en el despliegue de modelos. El objetivo fundamental de dicho motor es ejecutar estos modelos con la máxima eficiencia posible, lo que implica la minimización de la latencia de inferencia y la optimización en la utilización de los recursos del hardware subyacente. El motor de Inferencia es la unidad de computación principal que ejecuta el LLM se muestra en la Figura 4 5 y consta de:

- Procesamiento por lotes: Agrupa varias consultas en cola en un único lote de ejecución en la GPU. Reduce la sobrecarga de las llamadas individuales a la GPU y mejora la eficiencia del procesamiento paralelo.
- Ejecución del modelo: Ejecuta el modelo LLM en sí mismo. Toma la entrada por lotes y genera tokens de salida. Puede utilizar optimizaciones tales como: Almacenamiento en caché clave-valor (KV) para una generación más rápida de tokens. Cuantificación para una mayor eficiencia de la memoria. Decodificación especulativa para mayor velocidad.

- Respuesta a la consulta: Recopila los resultados del modelo, los divide de nuevo en respuestas individuales para cada solicitud original y envía los resultados a la aplicación a través de HTTP/gRPC.

Existen distintos marcos de ejecución de modelos. En la Tabla 3-2, se resumen algunos de los más conocidos:

Marco / Herramienta	Lenguaje	Despliegue	Optimizaciones	Uso potencial
vLLM	Python (PyTorch)	Local & Cloud	PagedAttention, caché KV eficiente	Servicio alta velocidad de propósito general
Triton + TensorRT-LLM	C++, Python (NVIDIA)	On-prem & Cloud	TensorRT, cuantización FP8/INT8	GPU-heavy workloads
Hugging Face TGI	Python	Local, Cloud	Speculative decoding, quantisation	Modelos Hugging Face
CTranslate2	C++, Python	Local	Cuantización Optimizado para CPU	Servicios basados en CPU
Ollama	CLI	Local	Modelos empaquetados	Uso local
WebLLM	JavaScript	Navegador	Ejecución Web GPU	Ejecución en navegador
LM Studio	GUI App	Local	Interfaz chat integrado	Uso local offline
GPT4ALL	Python, GUI	Local	Optimizado para CPU	Uso privado, offline
llama.cpp	C++	Local	Cuantización a 4-bit	Pocos recursos hardware

Tabla 3-2. Características de los marcos de inferencia más conocidos

3.1.3 Inteligencia Artificial aplicada a agentes

3.1.3.1 Introducción a los agentes

Los agentes son sistemas que ejecutan tareas en nombre de otros, como software o seres humanos. Inspirados en la planificación clásica de la IA [21], los agentes modernos integran métodos simbólicos y sub-simbólicos para razonar sobre estados futuros, dependencias causales y objetivos a largo plazo.

Las arquitecturas de los agentes de IA incluyen componentes diseñados para permitir la percepción, el razonamiento, la toma de decisiones y la interacción [22].

Los agentes de IA han empezado a influir en una amplia gama de aplicaciones del mundo real, actuando como componentes críticos en procesos complejos de tomas de decisión. Integran avances en percepción, razonamiento, comunicación y control para proporcionar soluciones adaptativas y sensibles al contexto que antes estaban fuera del alcance de los sistemas de software tradicionales [23].

3.1.3.2 Agentes aplicados con inteligencia Artificial Generativa

El uso de herramientas externas o complementos puede aumentar la calidad de las respuestas de los modelos generacionales. La función de llamada a herramientas permite al usuario proporcionar al LLM una indicación junto con un conjunto de funciones (herramientas), incluidos sus parámetros y descripción de las acciones a realizar. El LLM puede solicitar al usuario que ejecute una de estas funciones y utilizar el resultado en su respuesta.

La interoperabilidad es esencial en los flujos de trabajo multi-agente modulares, reutilizables y resistentes [24]. Los protocolos estandarizados reducen la carga del desarrollo, mejoran la seguridad y permiten la colaboración entre plataformas. En [25] se examina una serie de protocolos, seguido de un análisis comparativo y una hoja de ruta para su integración en los ecosistemas multi-agente.

Los protocolos multi-agente deben ser agnósticos con respecto a los detalles de implementación interna de los agentes. Deben proporcionar una interfaz estandarizada que facilite la comunicación entre agentes, lo que permite una composición perfecta. La estructura de los mensajes y el modelo de ejecución de los agentes son fundamentales para su composición y orquestación. En la Figura 3-5 se muestra un esquema de una arquitectura de agentes orquestados en un servidor mediante un agente orquestador.

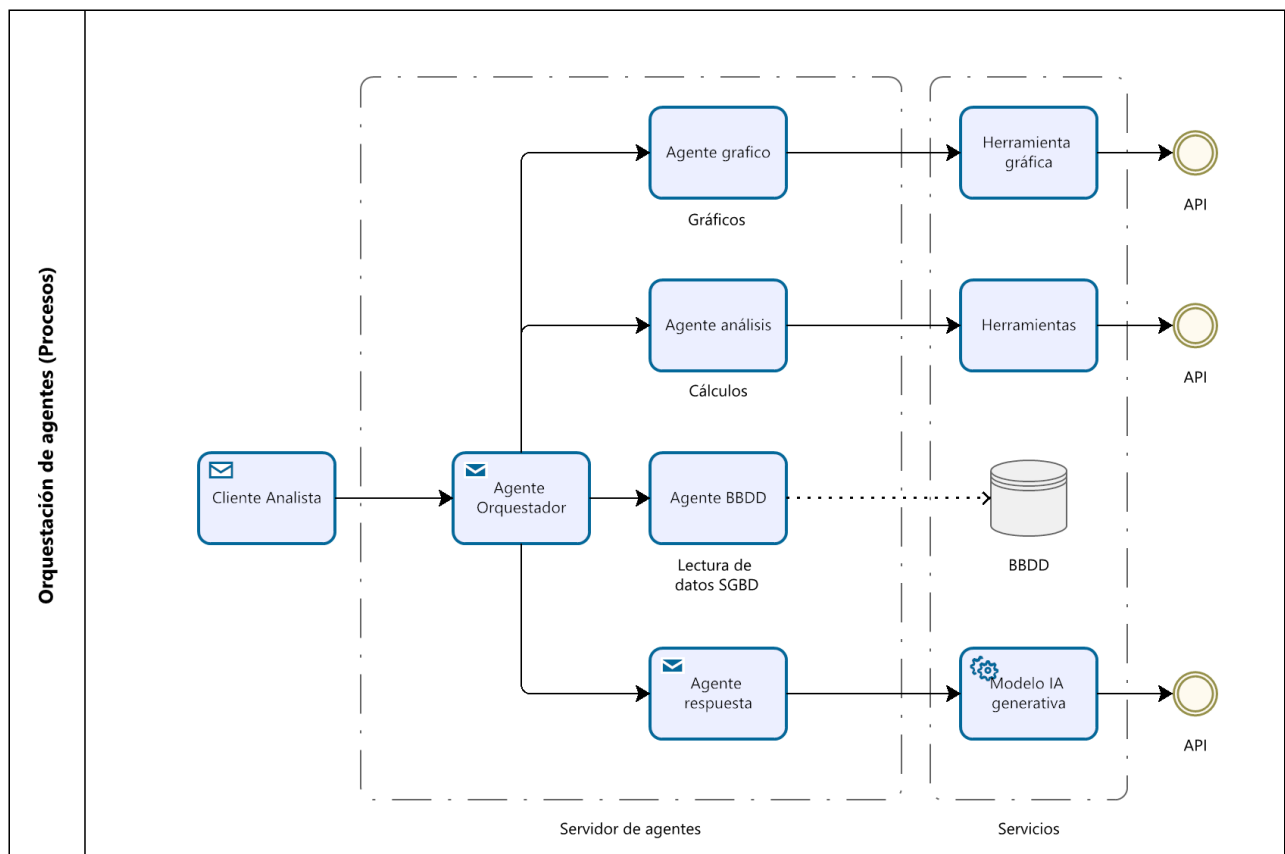


Figura 3-5. Orquestacion de agentes en sistemas multiagente.

3.1.4 Sistemas multiagente con Inteligencia Artificial generativa

Antes del desarrollo del protocolo de contexto de modelo MCP, la interacción entre sistemas de IA y fuentes externas de datos o herramientas de ejecución se caracterizaba por un enfoque fragmentado y no estandarizado. Las integraciones requerían la codificación específica de lógica para cada herramienta, el uso de secuencias de instrucciones frágiles o la dependencia de marcos de plugin propietarios, lo que dificultaba la escalabilidad y la interoperabilidad. Esta situación generaba el denominado problema de

integración $M \times N$, en el que M aplicaciones de IA y N herramientas distintas requerían hasta $M \times N$ desarrollos personalizados para lograr compatibilidad entre ellas.

MCP proporciona una solución estructurada a este desafío mediante la introducción de una interfaz estándar de comunicación. Este enfoque reduce significativamente la complejidad de las integraciones, reemplazando el modelo $M \times N$ por una arquitectura $M+N$: cada aplicación de IA implementa una única vez el cliente MCP, y cada herramienta o fuente de datos implementa una vez el servidor MCP. Esta unificación permite que las conexiones se establezcan de forma dinámica y sin necesidad de codificación adicional, ya que el protocolo proporciona un marco común para la interpretación y gestión de las interacciones entre componentes heterogéneos.

3.1.4.1 Protocolo de contexto de modelos

El Protocolo de Contexto de Modelos (“Model Context Protocol” en inglés, MCP) es un estándar abierto, de facto desarrollado por Anthropic [26] que permite a los desarrolladores crear conexiones bidireccionales seguras entre sus fuentes de datos y las herramientas de IA. Los desarrolladores pueden exponer sus datos a través de servidores MCP o crear aplicaciones de IA (clientes) que se conecten a estos servidores. MCP proporciona un marco flexible que permite a las aplicaciones de inteligencia artificial comunicarse dinámicamente con herramientas externas. A diferencia de los enfoques tradicionales con asignaciones predefinidas, MCP permite que los agentes de IA descubran, seleccionen y coordinen herramientas de forma autónoma según el contexto de la tarea [27].

También admite la participación humana, permitiendo que los usuarios aporten datos o aprueben acciones cuando sea necesario. Al unificar las interfaces, MCP facilita el desarrollo de aplicaciones de IA y mejora su capacidad para gestionar flujos de trabajo complejos. Desde su lanzamiento, MCP ha pasado rápidamente de ser un protocolo especializado a convertirse en una base clave para el desarrollo de aplicaciones nativas de IA.

A pesar de la rápida adopción de MCP, su ecosistema aún se encuentra en una etapa temprana. El trabajo en [28], ofrece el primer análisis del ecosistema MCP: examina su arquitectura y flujo de trabajo, define el ciclo de vida de los servidores MCP e identifica posibles riesgos de seguridad en cada etapa, como suplantación en el instalador o conflictos en los nombres de herramientas.

MCP utiliza JSON-RPC 2.0 como protocolo de llamada remota (“Remote Procedure Call” en inglés, RPC) subyacente [29]. Los clientes y los servidores se envían solicitudes entre sí y responden en consecuencia. Se pueden utilizar notificaciones cuando no se requiere respuesta.

MCP define tres primitivas básicas que los servidores pueden exponer:

- Herramientas: funciones ejecutables que las aplicaciones de IA pueden invocar para realizar acciones (por ejemplo, operaciones con archivos, llamadas a API, consultas a bases de datos).
- Recursos: fuentes de datos que proporcionan información contextual a las aplicaciones de IA (por ejemplo, contenidos de archivos, registros de bases de datos, respuestas de API).
- Instrucciones: Son plantillas reutilizables que ayudan a estructurar las interacciones con los modelos de lenguaje.

MCP también define primitivas que los clientes pueden exponer. Estas primitivas permiten a los programadores de servidores MCP crear interacciones más ricas:

- Muestreo (“sampling” en inglés): permite a los servidores solicitar completaciones del modelo de lenguaje a la aplicación de IA del cliente. Esto resulta útil cuando los autores de servidores desean acceder a un modelo de lenguaje, pero quieren mantener la independencia del modelo y no incluir un SDK de modelo de lenguaje en su servidor MCP. Pueden utilizar el método de muestreo/completación para solicitar una completación del modelo de lenguaje a la aplicación de IA del cliente.

- Obtención (“elicitation” en inglés): Permite a los servidores solicitar información adicional a los clientes. Esto es útil cuando los servidores desean obtener información del cliente o solicitar la confirmación de una acción. Pueden utilizar el método elicitation/request para solicitar información adicional al cliente.
- Registro (“logging” en inglés): Permite a los servidores enviar mensajes de registro a los clientes con fines de depuración y supervisión.

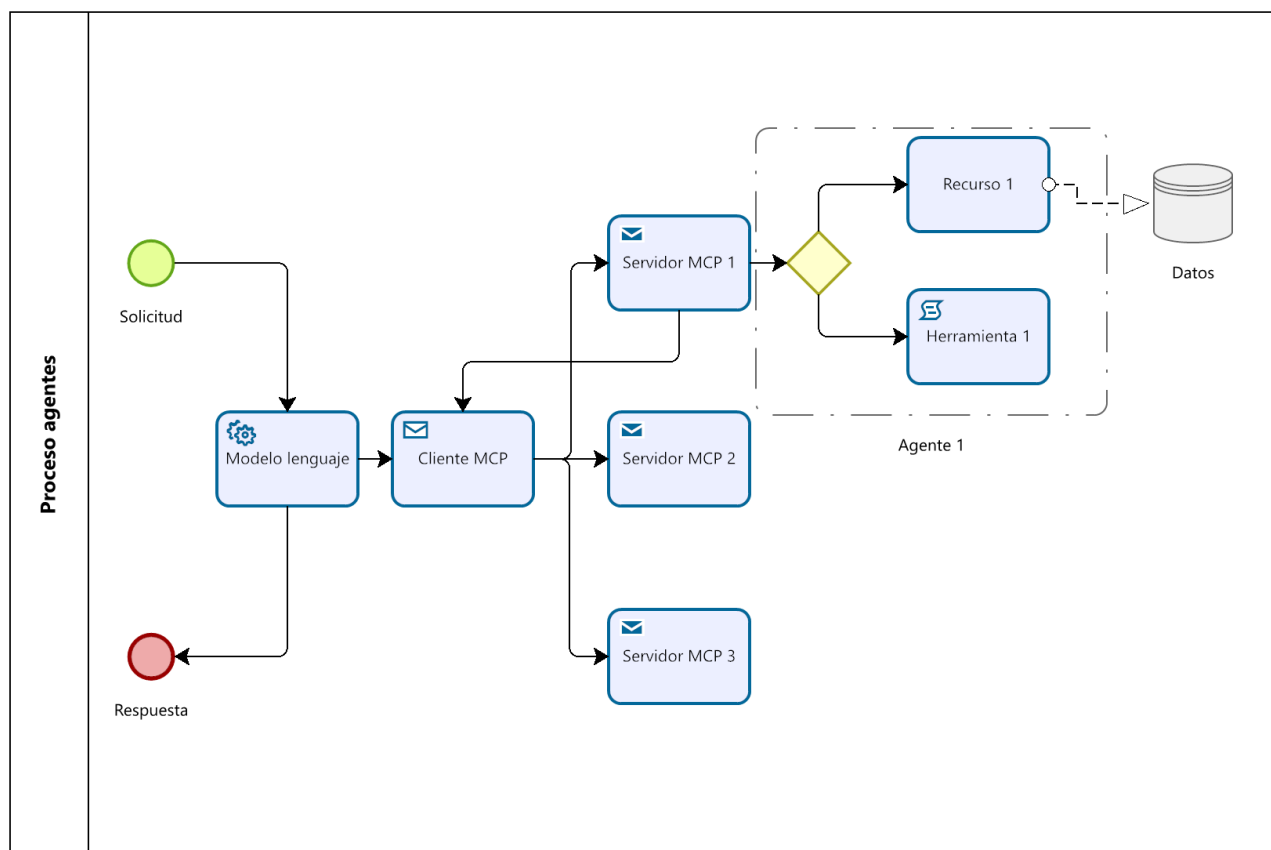


Figura 3-6. Arquitectura de un Sistema de Agentes con Modelos de Lenguaje.

3.1.4.2 Protocolo de comunicación agente a agente

El protocolo de comunicación agente a agente (A2A) [30] es un protocolo abierto que proporciona una forma estándar para que los agentes colaboren entre sí, independientemente del marco subyacente o del proveedor. A2A facilita la comunicación entre un agente “cliente” y un agente servidor “remoto”. El agente cliente se encarga de formular y comunicar tareas, mientras que el agente remoto se encarga de ejecutarlas con el fin de proporcionar la información correcta o tomar las medidas adecuadas.

A2A y el Protocolo de contexto de modelo (MCP) son estándares complementarios para crear aplicaciones agenticas robustas. ACP de IBM, mostrado a continuación, es un protocolo incorporado al protocolo A2A.

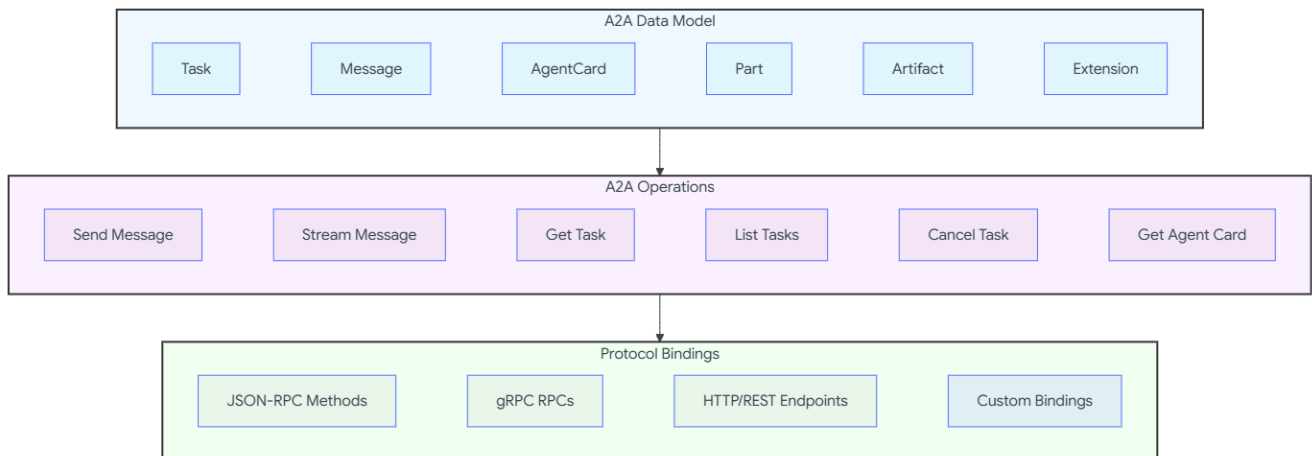


Figura 3-7. Estructura de A2A. Fuente: [30]

La arquitectura del protocolo A2A se presenta en la Figura 3-7 y se organiza en tres capas diferenciadas: una capa de modelo de datos, una capa de operaciones abstractas y una capa de protocolos. Los agentes locales interactúan con agentes alojados en servidores remotos a través de una pila de protocolos que incluye mecanismos de llamada a procedimiento remoto (RPC), HTTP/REST y protocolos propietarios. La flexibilidad de la capa de protocolos favorece la interoperabilidad entre distintos tipos de agentes y entornos. Asimismo, se evidencia la compatibilidad del protocolo A2A con el Model Context Protocol (MCP), lo que posibilita el acceso a herramientas remotas expuestas por servicios MCP.

A diferencia de otros protocolos, el protocolo A2A dispone de diferentes SDK para desarrollo multi-lenguaje como Python, Javascript, Java, C#/.NET y Golang.

El protocolo A2A se percibe como un super-protocolo que circunscribe los otros dos protocolos analizados en este documento (MCP y ACP).

3.1.4.1 Protocolo de comunicación de agentes

El Protocolo de Comunicación de Agentes (“Agent Communication Protocol” en inglés, ACP) es un protocolo abierto interoperable de agentes que resuelve el actual reto de conectar agentes de IA, aplicaciones y seres humanos. Los agentes de IA se construyen a menudo de forma aislada, en diferentes marcos, equipos e infraestructuras. Este aislamiento dificulta que los agentes trabajen juntos de forma eficaz. ACP resuelve este inconveniente permitiendo que los agentes se comuniquen a través de una API RESTful estandarizada [31].

ACP tiene las siguientes características importantes:

- Permiten comunicación síncrona y asíncrona
- Interacciones en streaming
- Patrones de funcionamiento con y sin estado
- Descubrimiento de agentes en línea y fuera de línea

ACP tiene las siguientes ventajas:

- Sustitución flexible de agentes a medida que la tecnología de IA evoluciona.
- Colaboración multi-agente: Permite crear sistemas sofisticados en los que agentes especializados trabajan como equipos coordinados.
- Integración entre plataformas, conectando los agentes a través de su pila tecnológica.
- Colaboración segura de agentes entre organizaciones o departamentos. Los agentes mantienen el estado y el historial de conversaciones a lo largo de las interacciones mediante sesiones.

- Al crear una base universal para la comunicación entre agentes, impulsa una mayor adopción de la IA y permite sistemas inteligentes más potentes e interconectados.

La Tabla 3-3 muestra las distintas primitivas que maneja el protocolo.

Primitiva	Control	Descripción
Prompts	Usuario	Plantillas interactivas invocadas por el usuario
Recursos	Aplicación	Datos contextuales gestionados por la aplicación cliente
Herramientas	Modelo	Funciones expuestas al LLM

Tabla 3-3. Primitivas del protocolo de comunicación de Agentes

3.1.4.2 Servidor de agentes ACP

Un servidor ACP puede alojar uno o más agentes ACP que ejecutan solicitudes y devuelven resultados al cliente utilizando el protocolo ACP. El servidor ACP expone a los agentes a través de un interfaz REST. Se puede realizar un descubrimiento básico de los agentes existentes en un servidor ACP mediante una solicitud Python o una solicitud REST al API. El descubrimiento abierto se realiza mediante un archivo de manifiesto (fichero `_manifest.yml`). Estos dos métodos son métodos en línea, pero existen otros fuera de línea, como el descubrimiento basado en registro o el descubrimiento incrustado. Los métodos fuera de línea no son de interés ya que incluyen el tratamiento a través de terceros como BeeAI [32].

El servidor ACP envuelve a los agentes y expone un punto final HTTP (“endpoint” en inglés), el agente se invoca y devuelve respuestas en el formato ACP estandarizado.

3.1.4.3 Cliente ACP

Un cliente ACP puede ser utilizado por un agente ACP, una aplicación u otro servicio que realice solicitudes a un servidor ACP utilizando el protocolo ACP .

En su forma más sencilla una implementación ACP conecta un cliente directamente a un único agente a través de una interfaz REST sobre HTTP. Este patrón es ideal para la comunicación directa con un único agente especializado, configuraciones ligeras con requisitos mínimos de infraestructura, entornos de desarrollo y depuración, e implementaciones de prueba de concepto.

3.1.4.4 Orquestador de agentes

La orquestación avanzada permite una arquitectura multi-agente sofisticada para flujos de trabajo complejos. El orquestador ejecuta las tareas mediante un proceso de razonamiento paso a paso, delegando las subtareas a los agentes auxiliares pertinentes según sea necesario e integrando sus respuestas para generar resultados coherentes y sensibles al contexto. El agente orquestador/enrutador es un agente central común que:

- Descompone solicitudes complejas en subtareas especializadas.
- Deriva las tareas a los agentes especializados adecuados.
- Agrega las respuestas en resultados coherentes.
- Utiliza además de sus propias herramientas, las expuestas por agentes a través de una extensión MCP.

En la Figura 3-8 se muestra una arquitectura de agentes ACP con un elemento agente de orquestación. Es muy relevante el aspecto de la compatibilidad entre ACP y MCP desde el punto de vista de la interoperabilidad entre agentes.

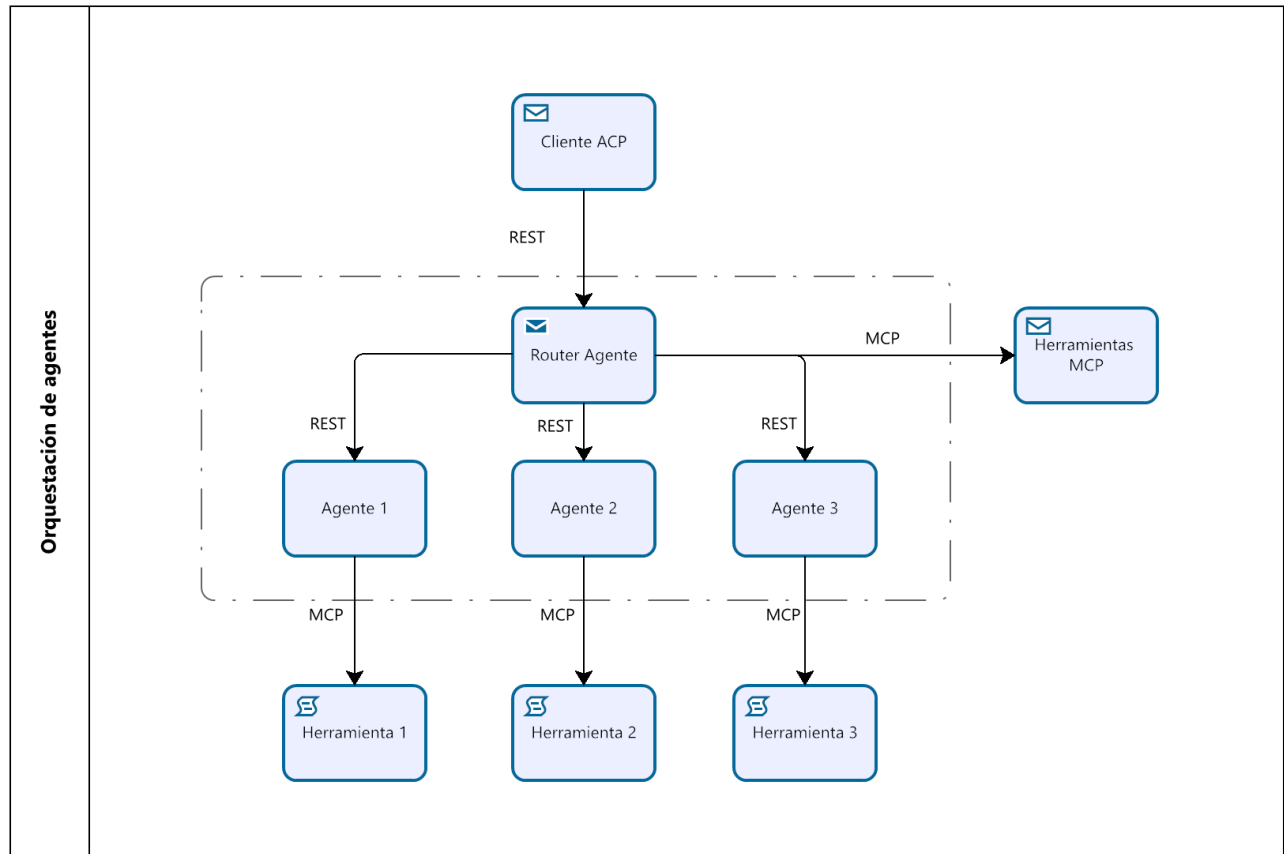


Figura 3-8. Orquestación de agentes en ACP con agentes enrutadores

3.1.5 Generación aumentada por recuperación

La búsqueda de información permite que la IA encuentre la información más útil y precisa para responder a preguntas, tomar decisiones o aprender de los datos existentes. El acceso a información de texto permite que la IA comprenda el contenido y el contexto de los textos, más allá de simples coincidencias de palabras clave, para ofrecer resultados más precisos y significativos.

La generación aumentada por recuperación (RAG) es una arquitectura híbrida emergente diseñada para abordar las limitaciones de los modelos generativos puros. La RAG integra dos componentes clave: un mecanismo de recuperación, que recupera documentos o información relevante de una fuente de conocimiento externa, y un módulo de generación, que procesa esta información para generar texto similar al humano [33].

La RAG combina las características de la recuperación de información y la generación de texto para crear respuestas más precisas y contextualmente relevantes. El RAG incluye de forma dinámica contexto complementario al responder a una consulta del usuario, adaptada al contenido específico del mensaje o pregunta del usuario [34]. Es un modelo híbrido que mejora las capacidades de los modelos de lenguaje incorporando una base de conocimiento externa o un almacén de documentos.

El proceso básico involucra dos acciones principales identificadas en la Figura 3-9:

- **Recuperación:** en esta fase, el modelo recupera piezas de información relevantes de una fuente externa, como una base de datos o un almacén de vectores, en función de la consulta de entrada.
- **Generación:** la información recuperada se utiliza a continuación mediante un modelo de lenguaje generativo para producir una respuesta coherente y contextualmente apropiada.

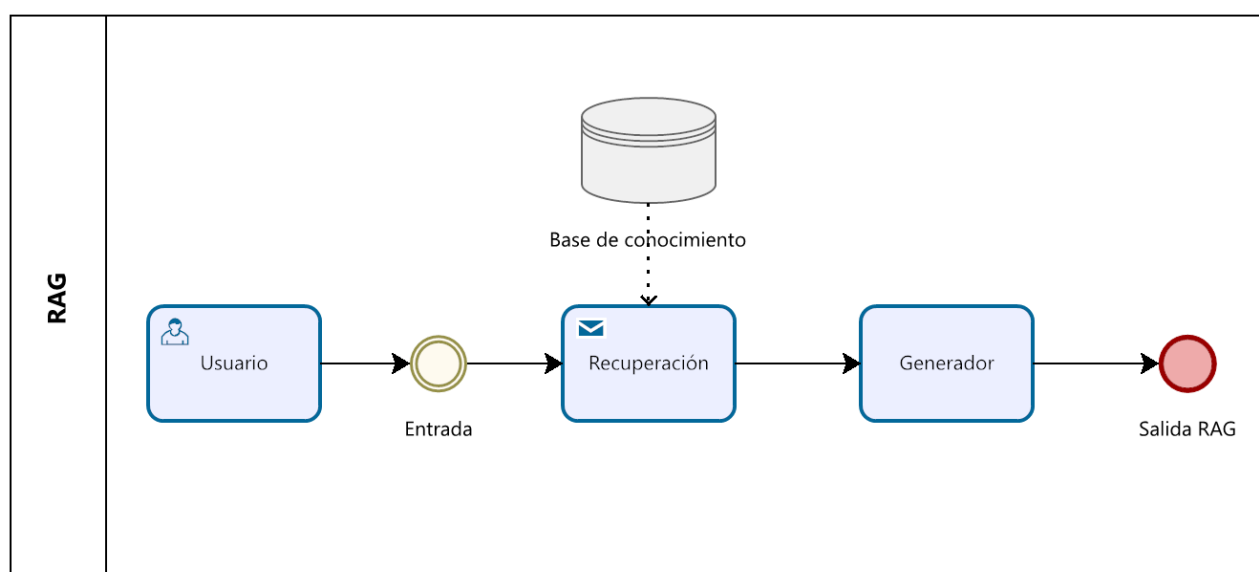


Figura 3-9. Proceso básico de un RAG.

El paradigma de investigación RAG está en continua evolución, y se clasifica en tres etapas [35]: RAG ingenuo, RAG avanzado y RAG modular. El RAG “ingenuo” sigue un proceso tradicional que incluye indexación, recuperación y generación. Sin embargo, este tipo de RAG presenta algunas deficiencias. En la fase de recuperación a menudo tiene dificultades con la precisión y la recuperación, lo que lleva a la selección de fragmentos desalineados o irrelevantes, y a la pérdida de información crucial. Al generar respuestas, el modelo puede enfrentarse al problema de las alucinaciones, en el que produce contenido no respaldado por el contexto recuperado.

El RAG avanzado se centra en mejorar la calidad de la recuperación, empleando estrategias previas y posteriores a la recuperación. Perfecciona sus técnicas de indexación mediante el uso de un enfoque de ventana deslizante, segmentación detallada y la incorporación de metadatos. La arquitectura modular RAG va más allá de los dos paradigmas RAG anteriores, ofreciendo una mayor adaptabilidad y versatilidad.

La **¡Error! No se encuentra el origen de la referencia.** muestra el proceso detallado de la función de búsqueda y recuperación de información mediante un RAG. El proceso detalla cada uno de los dos elementos fundamentales, la recuperación y la generación de contenido en función de los datos recuperados.

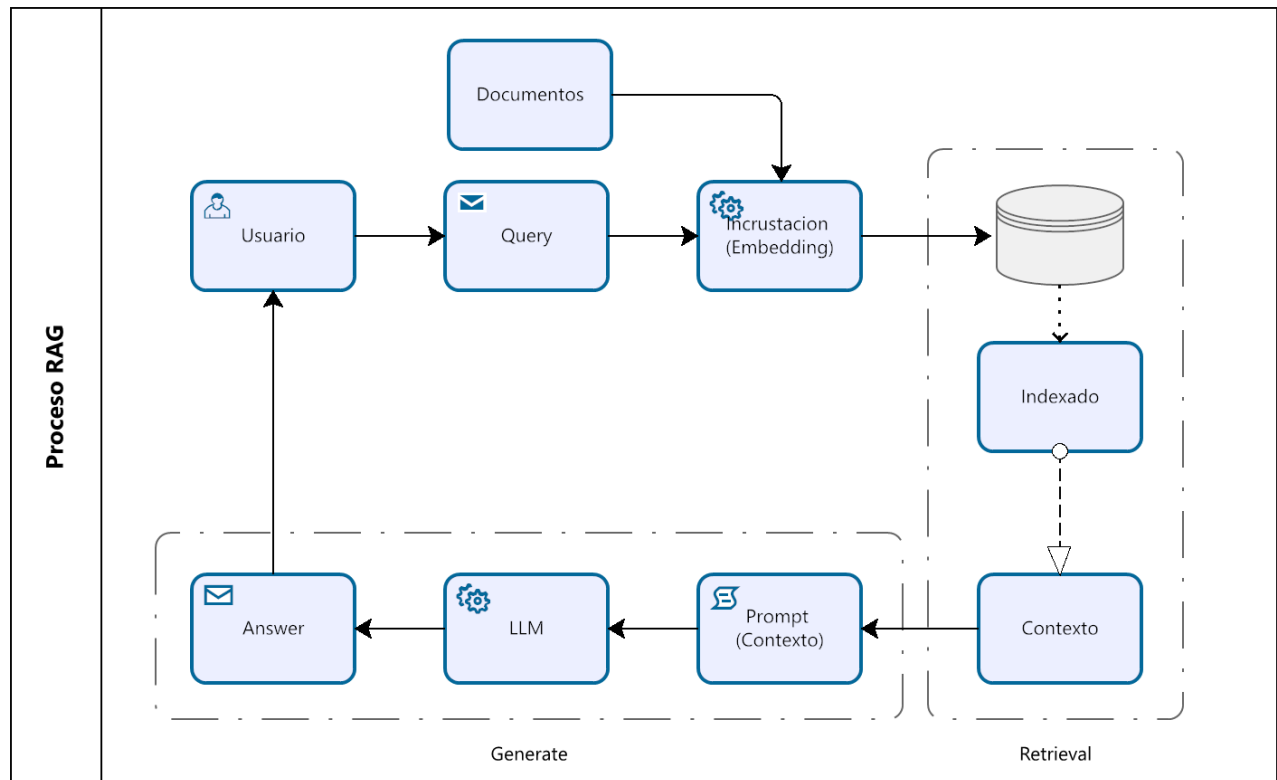


Figura 3-10. Proceso detallado de recuperacion de información con inteligencia artificial generativa

En el estudio de aplicación de RAG a la evaluación de riesgos para la salud realizado en [36], probaron un RAG ingenuo y un RAG avanzado introduciendo técnicas más complejas, como la reescritura de consultas, la reordenación de documentos y la síntesis rápida, con el objetivo de mejorar la relevancia de la recuperación y la calidad del texto generado. Los resultados del estudio indicaron que el sistema avanzado de preguntas y respuestas con recuperación mejorada obtuvo precisiones que oscilaron entre 0,606 y 0,723, superando a modelos como GPT-4. Sólo en el ámbito de la evaluación de riesgos para la salud, la precisión del sistema avanzado de preguntas y respuestas (0,583) fue ligeramente inferior a la del RAG ingenuo (0,599).

3.1.6 Bases de datos vectoriales

3.1.6.1 Definición

Un sistema de gestión de bases de datos vectoriales (VDBMS) es un tipo especializado de sistema de gestión de bases de datos centrado en la gestión eficiente de datos vectoriales de alta dimensión. Una base de datos vectorial (VDB) almacena y recupera información basándose en incrustaciones vectorizadas o representaciones numéricas que capturan el significado y el contexto del texto. En las VDB, se pueden realizar búsquedas y recuperaciones semánticas basadas en la similitud de significados en lugar de en palabras clave. Suelen admitir la búsqueda por similitud mediante métodos de indexación que permiten una búsqueda rápida y precisa de vectores similares, es decir, la búsqueda de vectores que se asemejan mucho a un vector de consulta dado basándose en métricas de distancia específicas, como la distancia euclídea (Ecuación 1) o la similitud coseno (Ecuación 2) [37].

$$d(P, Q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

Ecuación 1

$$\text{soft_coseno}_{(a,b)} = \frac{\sum_{i,j}^N (s_{ij} a_i b_j)}{\sqrt{\sum_{i,j}^N (s_{ij} a_i a_j)} \sqrt{\sum_{i,j}^N (s_{ij} b_i b_j)}}$$

Ecuación 2

3.1.6.2 Funcionamiento

En el procesamiento del lenguaje natural, las palabras y las frases se convierten en vectores de tal manera que palabras similares tienen representaciones vectoriales similares. La similitud puede ser distinta dependiendo del contexto de la frase. Al igual que con la inserción de datos en la base de datos vectorial, las consultas en lenguaje natural o los valores legibles por humanos en consultas en deben vectorizarse antes de que la VDB pueda evaluar la similitud vectorial.

Una vez que los objetos de datos se han vectorizado y almacenado en la base de datos vectorial, los datos se indexan para permitir consultas más rápidas, como ocurre con prácticamente todos los modelos de datos [38].

Los vectores con tres dimensiones se pueden representar en el espacio vectorial de forma que cada elemento se encuentra identificado por un punto en el espacio euclideo como se puede observar en la Figura 3-11.

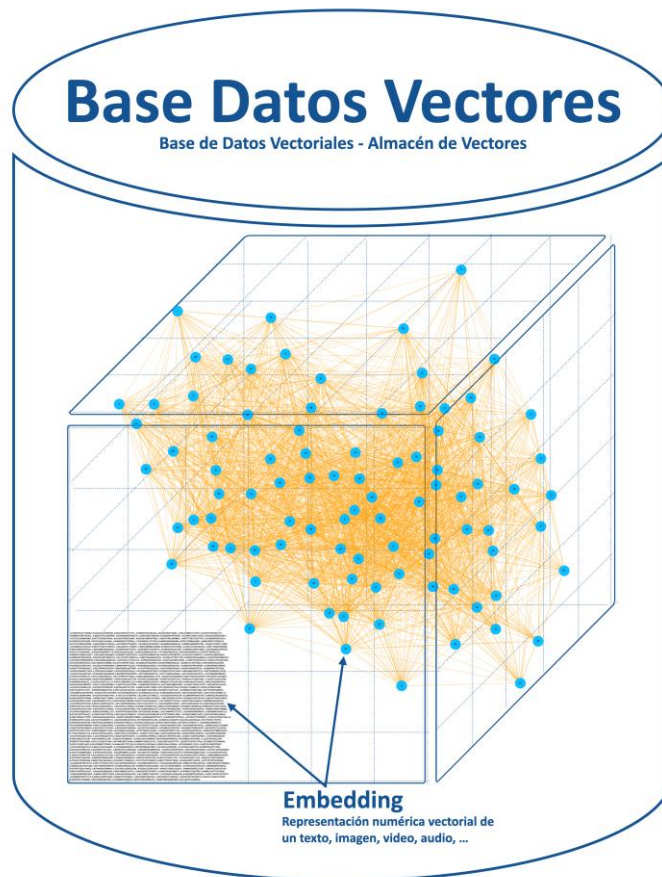


Figura 3-11. Espacio de una Base de Datos vectorial. Fuente: Wikipedia

3.2 Metodología y desarrollo

En este apartado se expone el trabajo desarrollado a lo largo del estudio. La metodología adoptada contempla el análisis de una arquitectura orientada a la aplicación de modelos de lenguaje en entornos locales, concebidos como agentes inteligentes. La integración de dichos agentes en el ámbito de la Defensa, y más específicamente en el Ejército de Tierra, presenta su aplicación como un servicio centralizado a nivel de unidades de entidad Batallón, Regimiento o Brigada. Asimismo, se contempla la ejecución de agentes inteligentes basados en modelos de lenguaje de menor tamaño (SLM) para su uso individual en unidades de menor escala, como escuadras o pelotones, mediante dispositivos personales.

Para ejecutar la propuesta se han ejecutado los siguientes hitos.

1. Uso de modelos de lenguaje en internet con herramientas en entorno local
2. Uso de modelos de lenguaje con motor de inferencia en entorno local
3. Desarrollo y aplicación de un servidor MCP local
4. Desarrollo y aplicación de un servidor ACP local
5. Desarrollo y prueba de un cliente local
6. Desarrollo de una arquitectura de agentes habilitada con SLM y LLM en entornos locales que permite ejecutar agentes en dispositivos Edge y llamar agentes remotos para búsqueda de información o ejecutar tareas más complejas en un servidor local.
7. Evaluación de la arquitectura y lecciones aprendidas

3.2.1 Uso de modelos de lenguaje en internet con herramientas en entorno local

En 2025, destacan modelos conversacionales y asistentes de contenido que combinan precisión, multimodalidad y facilidad de integración. Las siguientes son las soluciones más relevantes hoy en día:

- ChatGPT-5: Es el modelo más avanzado de OpenAI [39] disponible en 2025. Representa la evolución de la familia GPT y es accesible a través de la versión Plus de ChatGPT. Esta versión ha mejorado de forma notable la interacción multimodal, combinando texto, imagen y voz en un solo sistema optimizado para tareas complejas.
- Claude Sonnet 4: Claude Sonnet 4 es la versión más reciente del modelo de Anthropic, lanzada en mayo de 2025. Está diseñado con un enfoque en la seguridad, el razonamiento y la transparencia, y se posiciona como un competidor directo de ChatGPT-5.
- Gemini 2.5 Pro: Es el modelo más reciente de Google DeepMind, integrado en el ecosistema de Google Workspace. Ofrece una experiencia optimizada para usuarios que trabajan en entornos colaborativos y de productividad digital.
- Perplexity AI: Combina motor de búsqueda y generación de respuestas con IA, ofreciendo resultados precisos y verificados. Es una de las plataformas más destacadas para consultas documentadas.
- LLaMA 4: LLaMA 4 es la nueva generación de modelos de lenguaje de Meta, lanzada en 2025. Introduce una arquitectura modular basada en Mixture of Experts (MoE), mejorando la eficiencia y la escalabilidad. Además, aporta capacidades multimodales y mantiene un enfoque open source.

El despliegue de modelos de lenguaje a gran escala (LLM) en entornos cloud presenta diversas ventajas y limitaciones técnicas. Entre las ventajas destacan:

1. Al ser ofrecidos bajo el esquema de Software como Servicio (SaaS), la abstracción de la infraestructura subyacente, incluyendo la plataforma, sistema operativo y middleware, simplifica la integración y el despliegue para el usuario final.

2. Generalmente, estos servicios proporcionan interfaces de programación de aplicaciones (APIs) estandarizadas, como las expuestas por OpenAI, facilitando la orquestación y el desarrollo de agentes y aplicaciones clientes locales.

Por otro lado, se identifican las siguientes limitaciones:

1. La transmisión de datos hacia el servicio en la nube no garantiza la confidencialidad criptográfica total, lo que implica restricciones para el procesamiento de información sensible o clasificada debido a potenciales vulnerabilidades en la cadena de transmisión o almacenamiento.
2. El modelo de costes basado en el consumo por token generado implica que, a largo plazo, la escalabilidad económica de estas soluciones puede resultar limitada, incrementando significativamente el gasto operativo en escenarios de uso intensivo.
3. La confidencialidad de los datos introducidos en los Chat comerciales no está garantizada. Como muestra de ello la política de privacidad de ChatGPT [40] incluye lo siguiente: Los modelos fundacionales de OpenAI, incluidos los modelos en los que se basa ChatGPT, se desarrollan utilizando tres fuentes de información principales: (1) información disponible públicamente en Internet, (2) información a la cual tenemos acceso por medio de terceros con quienes nos asociamos y (3) información que proporcionan o generan nuestros usuarios o entrenadores e investigadores humanos.

3.2.2 Selección de modelos en la nube

A continuación, se presentan algunos factores relevantes a considerar en el proceso de selección de un modelo de lenguaje de gran tamaño (LLM) disponible en Internet:

- Tamaño del modelo y rendimiento.
- Coste y estrategia de hosting.
- Coste asociado al consumo del modelo.
- Coste derivado del alojamiento del modelo.

Dado que el presente trabajo se centra en la prueba de agentes de inteligencia artificial generativa con integración de herramientas externas, se ha optado por emplear Claude Desktop [41] como solución de circunstancias para validar el funcionamiento de los servidores de herramientas desarrollados para efectuar pruebas del concepto agente.

Existen otras aplicaciones que utilizan modelos en la nube con resultados equivalentes, sin embargo, Claude es una aplicación propietaria que utiliza modelos de lenguaje desarrollados por Anthropic, los cuales se alojan en la nube. Claude Desktop es el interfaz de la aplicación instalado en entorno local, que permite la conexión a diversos servidores de herramientas y recursos, incluidos aquellos alojados localmente. La aplicación ofrece un uso gratuito sujeto a determinadas limitaciones, lo que la convierte en una opción adecuada para prueba de agentes desarrollados de forma experimental.

El cliente constituye el componente fundamental de la interacción con el usuario y puede materializarse mediante una aplicación de escritorio, como en el caso de Claude Desktop o a través de una interfaz web. Claude Desktop proporciona una aplicación de escritorio para la interacción con el usuario que posibilita la conexión con servidores de herramientas MCP. La Figura 3-12 ilustra el interfaz frontal de la aplicación instalada en el escritorio de un computador personal (PC).

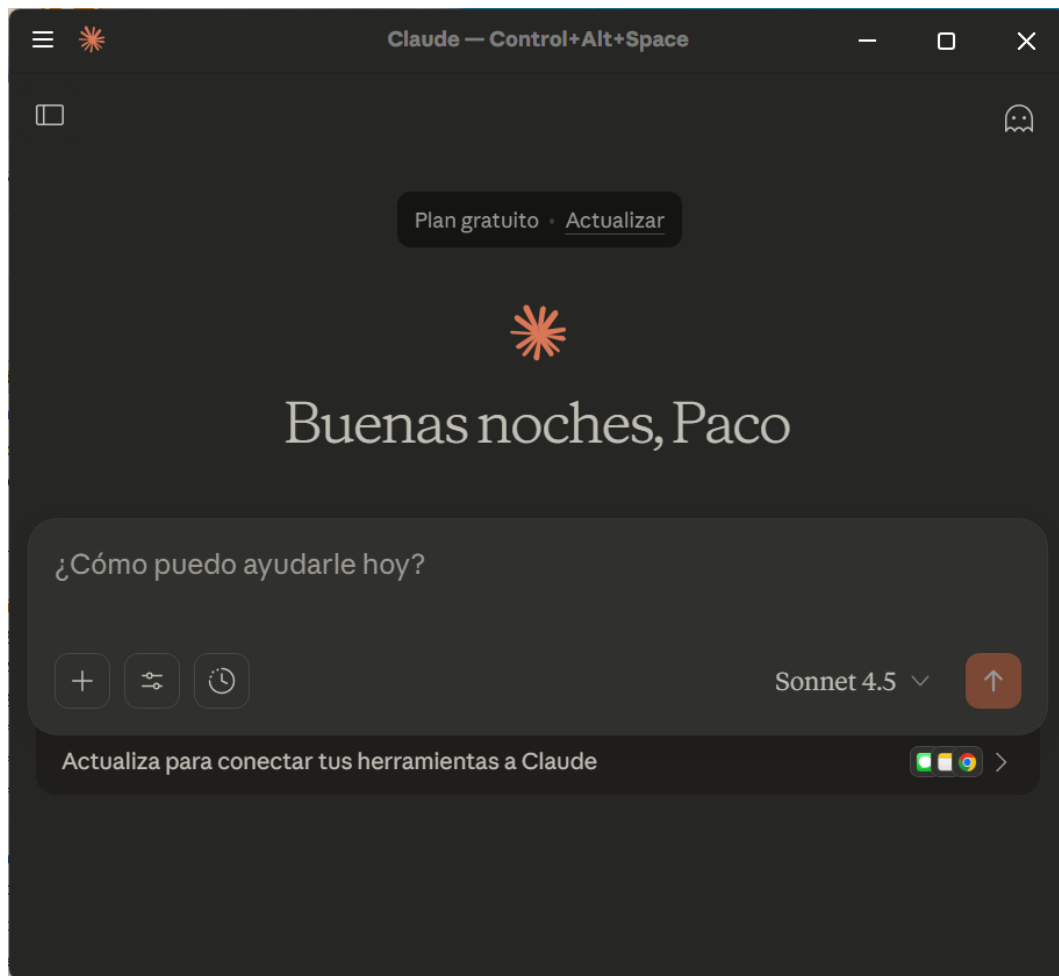


Figura 3-12. Frontal de la aplicación Claude Desktop utilizada para probar modelos en la nube

3.2.3 Selección de modelos de uso local

Vimos en el capítulo del marco tecnológico, que los costes de desarrollo de un modelo de lenguaje extenso y la complejidad del entrenamiento con bases de conocimiento propio son elevados. Por tanto, para poder desarrollar un modelo conversacional propio es necesario utilizar modelos pre-entrenados existentes. Obviamente, esto conlleva algunos riesgos a la hora de garantizar que el modelo y los datos que se le han proporcionado son de calidad suficiente para nuestra aplicación.

Existen muchos factores a tener en cuenta a la hora de elegir un LLM para su aplicación. Esos factores también deben dividirse en dos escenarios: LLM propietarios y LLM de código abierto. Respecto a los modelos abiertos debemos tener en cuenta los siguientes parámetros:

1. Tamaño del modelo
2. Recursos computacionales necesarios
3. Licencias
4. Soporte para fine-tuning
5. Compatibilidad e integración
6. Seguridad
7. Comunidad y soporte

En la Tabla 3-4, cuyos datos fueron actualizados al 1 de noviembre de 2025, se presenta una comparación de diversos modelos para su evaluación en un entorno local. Para su clasificación en

modelos grandes y pequeños, se ha considerado el tamaño del modelo expresado en miles de millones de parámetros (billones en terminología anglosajona). Se definen como LLM aquellos modelos con más de 7B de parámetros, mientras que los modelos con menos de 7B se categorizan como SLM. Esta distinción resulta relevante, ya que determina la viabilidad de ejecutar un modelo en un entorno basado únicamente en CPU.

Modelo	Parámetros	Memoria	Tamaño	Licencia	Herramientas	Contexto tokens
Salamandra-2b-instruct:fl6	2B	4.5G	SLM		No	8K
Gemma3:270m	270M	192MB	SLM	Apache2.0	No	32K
Gemma3:1b	1B	815MB	SLM	Apache2.0	No	32K
Gemma3n:e2b	2B	5.6G	SLM	Apache2.0	No	32K
Gemma3:4b	4B	3.3G	LLM	Apache2.0	No	128K
Qwen2.5:0.5	0.5B	398MB	SLM	Apache2.0	Si	32K
Qwen2.5:1.5	1.5B	986MB	SLM	Apache2.0	Si	32K
Qwen3:1.7	1.7B	1.4G	SLM		Si	40K
Llama3.2:1b	1.0B	1.3G	SLM	Meta	Si	128K

Tabla 3-4. Modelos de IA generativa SLM y LLM utilizables en un entorno local.

El uso de la memoria varía en función de la cuantización del modelo que puede ser desde 4b hasta 16 o 32 bits de punto flotante (FP16/32). Los modelos Gemma están declarados como modelos de código libre con pesos abiertos.

3.2.4 Ventajas y desventajas de los modelos en local

El uso de modelos de lenguaje en el entorno local tiene una serie de ventajas, para su aplicación en Defensa:

1. Seguridad de los datos: Mantener los datos “On-premise” asegura la confidencialidad de la información clasificada manteniendo la soberanía digital, que los modelos en internet no son capaces de asegurar.
2. Ahorro de costes de servicio: Los servicios de internet tienen costes asociados al número de tokens generados y el uso de API.
3. Funcionalidad fuera de línea: Se pueden instalar en modo local lo cual evita el uso a través de redes de datos.

También conlleva algunas desventajas:

1. El coste del hardware necesario puede ser elevado debido a las necesidades de computación avanzadas.
2. Es necesario disponer de una plantilla de ingenieros y técnicos con formación específica en desarrollo de aplicaciones basadas en IA generativa.

En el ámbito de los modelos lingüísticos, el almacenamiento y la implementación eficiente de los modelos en entornos locales con menos capacidades de computación, se han convertido en aspectos

cruciales. Aquí es donde entran en juego formatos de almacenamiento de modelos como GGUF (Generic GPT Unified Format) [42]. GGUF es un formato de almacenamiento de modelos creado específicamente para almacenar y ejecutar de manera eficiente modelos de lenguaje cuantificados de gran tamaño. GGUF proporciona una forma de comprimir modelos basados en GPT y almacenarlos en un formato que se puede implementar en CPU o dispositivos de bajo consumo sin sacrificar demasiada precisión.

3.2.5 Inferencia de modelos en local

Para ejecutar modelos en local necesitamos un motor de ejecución o motor de inferencia como los descritos en la sección 3.1.1.

Se ha seleccionado la herramienta Ollama [43] por ser una herramienta de código abierto diseñada para implementar y gestionar LLM en la máquina local. Permite simplificar el proceso al empaquetar los pesos, las configuraciones y las dependencias del modelo en un único “archivo de modelo”, muy similar a una aplicación en contenedor. Ollama está basado en el eficiente motor de inferencia en C++, llama.cpp. Es compatible con numerosos modelos populares como Llama 3.2, Mistral y DeepSeek R1. Puede interactuar con ella a través de la línea de comandos (CLI), API REST o interfaces gráficas de usuario fáciles de usar, como Open WebUI, y funciona en macOS, Linux y Windows.

Existen otras alternativas basadas en interfaz gráfica de usuario (GUI, por sus siglas en inglés), como por ejemplo LMStudio[44]. Sin embargo, al tratarse de aplicaciones de escritorio se optó por utilizar Ollama ya que podía ejecutarse desde el CLI.

Antes de cargar y utilizar modelos GGUF, hay que asegurarse de que el entorno esté correctamente configurado para manejar este formato. La mayoría de los desarrolladores que trabajan con modelos GGUF utilizan entornos de programación como Python y herramientas como bibliotecas de transformadores o motores de inferencia personalizados que admitan el formato. Las herramientas más comunes que admiten los modelos GGUF son extensiones de marcos de aprendizaje profundo existentes (por ejemplo, PyTorch, Hugging Face Transformers). Nos aseguraremos de tener lo siguiente:

- Python 3.7+ (utilizamos 3.10)
- Biblioteca Hugging Face Transformers (si procede)
- Bibliotecas específicas de GGUF o cargadores personalizados (a menudo disponibles en GitHub). GGUF está diseñado principalmente para su uso en CPU.
- CUDA o ROCm si se utilizan GPU para la inferencia.

Ollama destaca por su empaquetado de modelos integrado y la función "Modelfile", que permite a los usuarios personalizar modelos e integrarlos sin problemas en scripts y diversas aplicaciones.

3.2.6 Uso de herramientas (agentes)

Como se expuso en el capítulo del Estado del Arte, los modelos de lenguaje (LLM) son modelos estadísticos y lingüísticos que presentan capacidades intrínsecamente limitadas en la ejecución de tareas que requieren habilidades de razonamiento complejo, inducción lógica formal, cálculos matemáticos precisos, manipulación de datos estructurados, acceso a información en tiempo real y ejecución de acciones en entornos externos. Por esta razón, es necesario integrar herramientas especializadas que permitan extender la funcionalidad de los LLM más allá del procesamiento lingüístico puro, habilitando la ejecución de actividades como inferencia lógica formal, resolución de problemas matemáticos, consulta y actualización de bases de datos, interacción con APIs externas, y automatización de flujos de trabajo complejos.

Ollama admite la invocación de herramientas o invocación de funciones, lo que permite a un modelo utilizar herramientas e incorporar contexto a la pregunta del usuario para mejorar los resultados en las respuestas.

El marco admite varios tipos de invocación de herramientas:

- Llamada simple: una sola herramienta.
- Llamada en paralelo: más de una herramienta a la vez
- Bucle de herramientas (agente): Un bucle permite al modelo decidir cuándo invoca herramientas e incorpora sus resultados en su respuesta.
- Llamada en streaming: Al realizar la llamada, recopila todos los fragmentos de pensamiento, contenido y llamadas a herramientas como información de contexto y luego devuelve la información junto con los resultados de las herramientas en la solicitud.

La página web de Ollama dispone de un listado de modelos compatibles con indicación de su compatibilidad con el uso de Herramientas. En la columna herramientas de la Tabla 3-4 se muestra la compatibilidad de cada uno de los modelos analizados.

3.2.7 Desarrollo de agentes inteligentes

Tal como se expuso en el estado del arte y el capítulo de marco tecnológico, los modelos LLM requieren el uso de herramientas auxiliares para ejecutar tareas que trascienden el procesamiento lingüístico, tales como cálculos numéricos u operaciones complejas. Con este propósito, se han desarrollado los denominados agentes, en el marco de la denominada como inteligencia artificial agentica. En este contexto, se presentaron los protocolos de comunicación de agentes.

Hemos visto en el capítulo marco tecnológico, que en la actualidad existen, al menos dos protocolos de comunicación de agentes los cuales se presentaron en el capítulo de marco tecnológico, el MCP y el A2A (ACP).

3.2.8 Desarrollo de un servidor de agentes MCP

Construimos un sencillo servidor MCP para ejecutar agentes de predicción climática y lo conectamos a un host de escritorio, basado en Claude. Empezamos realizando una configuración básica y luego pasamos a casos de uso más complejos[45].

Para desarrollar el servidor MCP se pueden utilizar las librerías Python mcp [46] o Python fastmcp que es un conjunto de herramientas avanzado de MCP[47]. FastMCP 2.0 es la versión mantenida actualmente en activo. Ambas librerías permiten desplegar un servidor MCP de forma sencilla. Sin embargo, “fastmcp” cuenta con un conjunto completo de funciones que van más allá de la especificación MCP básica. Entre ellas se incluyen la implementación, la autenticación, los clientes, el proxy y la composición de servidores, la generación de servidores a partir de API REST, la reescritura dinámica de herramientas, herramientas de prueba integradas, integraciones y mucho más.

Según la versión utilizada tienen funcionalidades distintas que hemos probado, cuyas diferencias se muestran en la Tabla 3-5.

Librería	Versión	Interfaz de comunicación
mcp	1.0.0	STDIO
fastmcp	2.0.0	STDIO
fastmcp	2.10.6	STDIO, SSE y HTTP

Tabla 3-5. Interfaces disponibles en los paquetes de implementación MCP

Los servidores MCP proporcionan tres tipos principales de capacidades:

- Recursos: Datos tipo archivo que pueden ser leídos por los clientes (como respuestas de API o contenidos de archivos)
- Herramientas: Funciones que pueden ser llamadas por el LLM (con la aprobación del usuario)

- Prompts: Plantillas escritas que ayudan a los usuarios a realizar tareas específicas.

La Figura 3-13 presenta la arquitectura software de un agente inteligente basado en un modelo de lenguaje de gran escala. En dicha arquitectura, el cliente web realiza llamadas al modelo, el cual se ejecuta en el entorno Ollama y, de forma complementaria, hace uso de herramientas MCP proporcionadas por un servidor local. Este servidor es capaz de indexar diversos recursos, tales como bases de datos, herramientas que emplean técnicas de aprendizaje automático (o machine learning, ML) e incluso prompts específicos definidos por el usuario, los cuales no se representan explícitamente en la figura.

El acceso al servidor MCP puede efectuarse mediante la interfaz estándar o, alternativamente, como se ilustra en la figura, a través de un socket IP utilizando el protocolo HTTP en el puerto 8000. Esta última modalidad permite desacoplar el cliente del servidor, facilitando su despliegue en un host independiente y favoreciendo una mayor flexibilidad en la arquitectura del sistema.

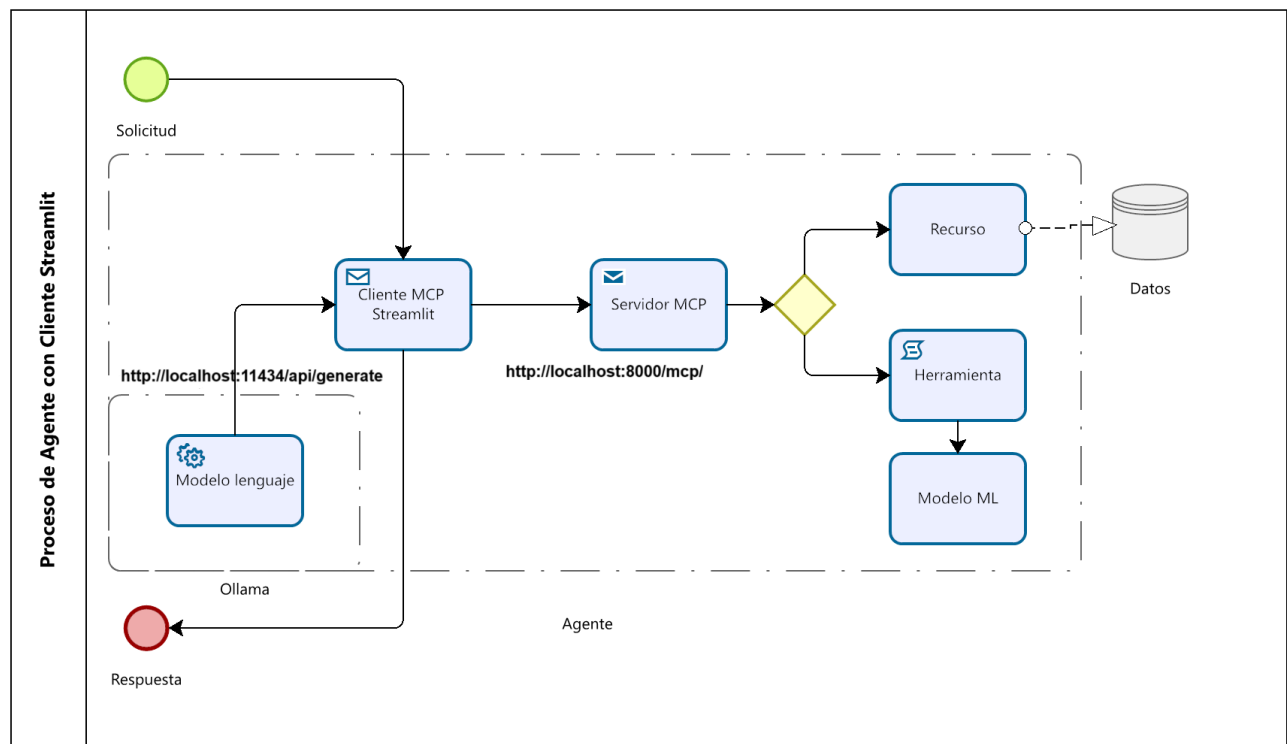


Figura 3-13. Agente inteligente (LLM + Servidor MCP)

3.2.9 Desarrollo de un servidor de agentes ACP

Para ejecutar agentes, es indispensable identificar previamente cuáles se encuentran disponibles. A diferencia de MCP, los servidores ACP exponen los metadatos de los agentes mediante un punto final (“endpoint” en inglés) de descubrimiento, el cual enumera los agentes registrados junto con sus respectivas capacidades. Esto representa una ventaja, ya que elimina la necesidad de implementar una tarea específica para el listado de agentes, requisito que sí resulta necesario en MCP.

La arquitectura de un servicio cliente-servidor ACP se muestra en la **¡Error! No se encuentra el origen de la referencia..** La arquitectura incluye una aplicación cliente ACP conectada a un agente enrutador (“router”) en el Servidor ACP. Este agente router selecciona un agente específico en función de la tarea a desarrollar. Los agentes utilizan herramientas declaradas en el servidor que pueden comunicarse mediante el protocolo de ACP o mediante MCP.

El elemento que integra la lógica del servidor incluye la declaración e instanciación de un servidor y la declaración de los agentes funcionales. Para crear un servidor de agentes ACP en lenguaje Python

se propone el código que aparece en Anexo II. Al arrancar, por defecto crea un servidor en <http://127.0.0.1:8000>.

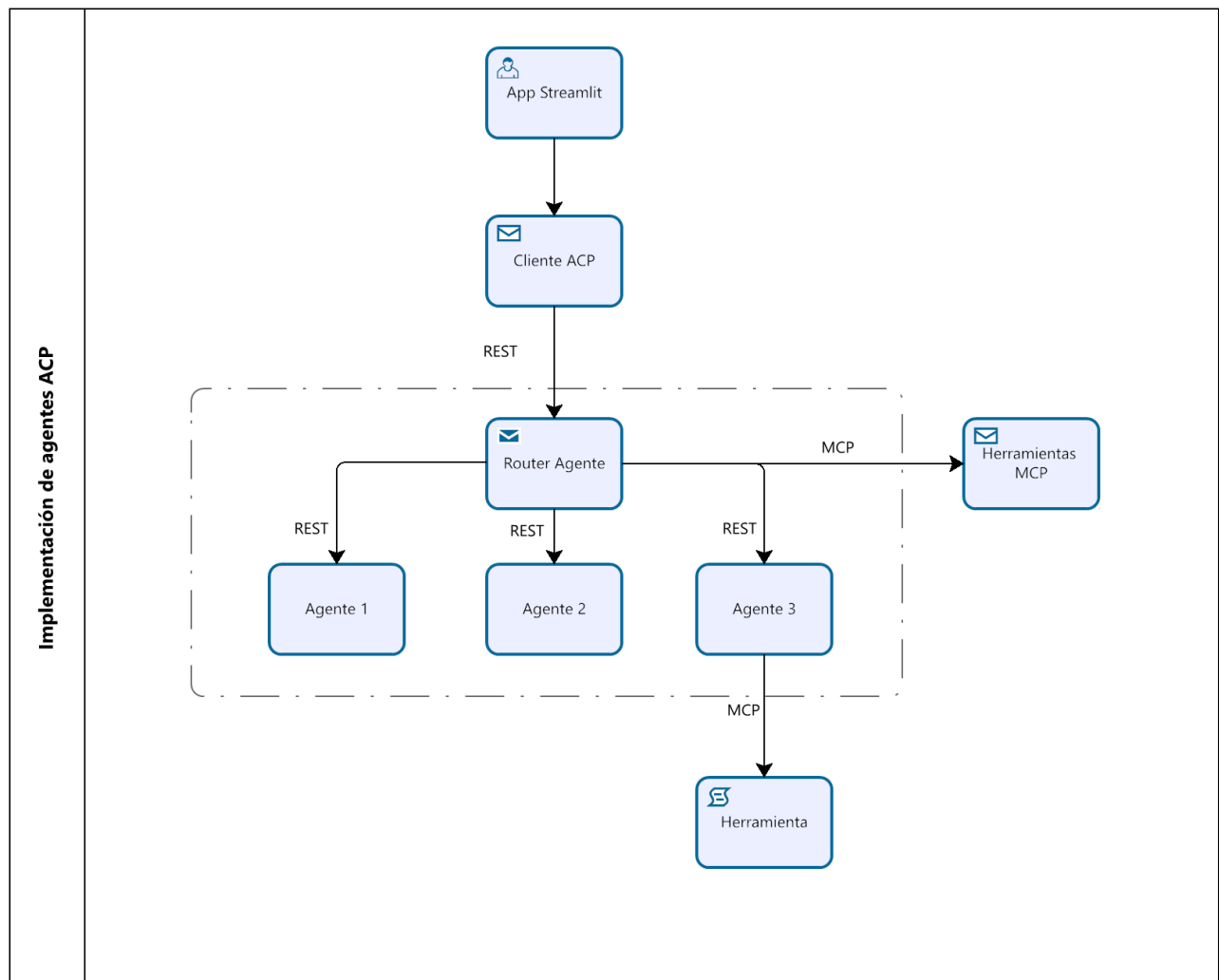


Figura 3-14. Arquitectura de un servicio cliente servidor con ACP

3.2.10 Desarrollo de un cliente de agentes MCP

Según la especificación del protocolo Model Context Protocol (MCP), un cliente puede establecer comunicación con un servidor mediante dos métodos principales: a través de la interfaz estándar de entrada/salida (STDIO) o mediante el protocolo de transferencia de hipertexto (HTTP). Si bien la interfaz STDIO está orientada a interacciones locales entre procesos ejecutados en la misma máquina física, el uso del protocolo HTTP permite desacoplar físicamente cliente y servidor, favoreciendo su despliegue distribuido en diferentes nodos o entornos de red. Este enfoque basado en HTTP garantiza un mayor aislamiento entre componentes, facilita la escalabilidad del sistema y permite una interoperabilidad más robusta en arquitecturas distribuidas, manteniendo la conformidad con las especificaciones del protocolo MCP.

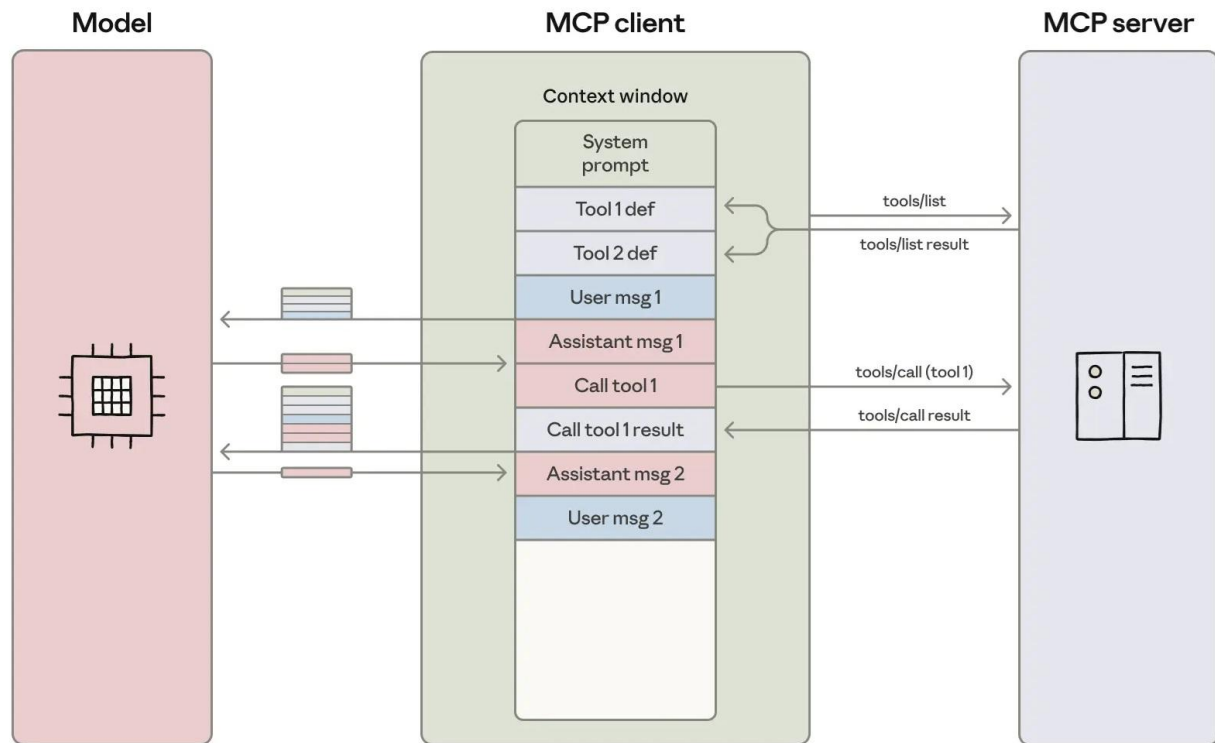


Figura 3-15. Interacción modelo-cliente-servidor. Fuente: Anthropic

El cliente MCP que se muestra en la Figura 3-15 realiza tres tareas fundamentales:

- Conecta con servidor MCP y descubre herramientas solicitando un listado.
- Llama a herramientas y recursos definidos en el cliente.
- Utiliza directa un modelo de IA alojado en el Host proporcionando el contexto a partir de los resultados de las herramientas y recursos.

Los clientes que operan a través de la interfaz estándar en una aplicación distribuida basada en JSON-RPC para MCP presentan un fuerte acoplamiento con la máquina (host) en la que se ejecutan. Habitualmente, estos clientes están diseñados para establecer comunicación con el servidor durante su fase de inicialización, ya que se crea un interfaz local de herramientas (“Stub”), el cual actúa como un mapa de las capacidades expuestas por el servidor. Dicho mapa se construye a partir del mecanismo de descubrimiento de herramientas proporcionado por el “Skeleton” del servidor, generalmente mediante una función de enumeración que permite identificar y registrar las herramientas disponibles.

3.2.10.1 Cliente local con LLM en el host local

Existen algunos clientes ya desarrollados, como Ollmcp [48], que es un interfaz en línea de comandos, que permite conectar servidores MCP utilizando Ollama como motor de modelos de inteligencia generativa. El cliente permite realizar llamadas al servidor CLI como llamadas al servidor SSE o Streaming-HTTP. Esta última opción permite desacoplar el host del cliente del servidor MCP.

En la Figura 3-16 se muestra la arquitectura del servicio de agentes con LLM a través de Ollama, un cliente Ollmcp con interfaz CLI y un servidor MCP. El servidor puede ejecutarse en modo CLI, en modo SSE o Streaming-HTTP.

Para que los clientes MCP locales como “Claude Desktop” o “Ollmcp” en nuestro caso, puedan acceder a los servidores MCP existe un archivo de configuración JSON que incluye un listado de los servidores MCP disponibles.

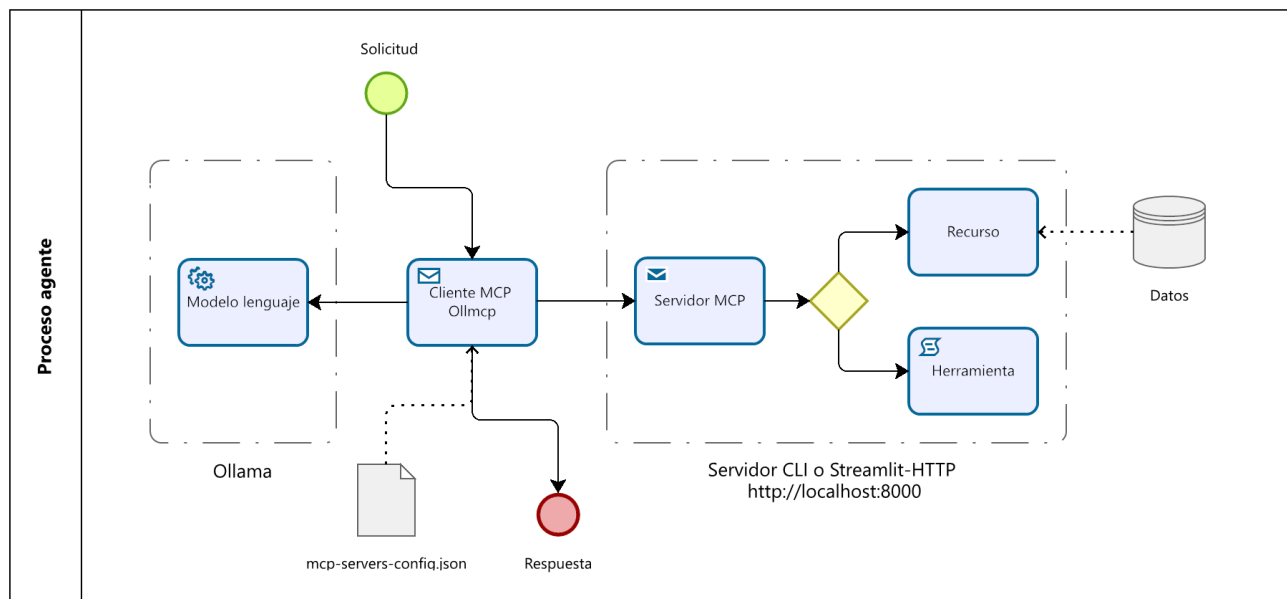


Figura 3-16. Servicio de agentes con servidor MCP y cliente Ollmcp

Sin embargo, se implementó un cliente Web mediante el marco de programación Streamlit [49]. Streamlit es un marco de aplicaciones de código abierto con el que es muy fácil diseñar y desplegar aplicaciones web.

Con Streamlit se puede crear una aplicación en unas pocas líneas de código gracias a una API sencilla. Se actualiza automáticamente a medida que se guarda el archivo fuente en Python, de forma iterativa. Permite añadir widgets como se declara una variable. No es necesario escribir un backend, definir rutas, gestionar solicitudes HTTP, conectar un FrontEnd, escribir HTML, CSS o JavaScript. La solución permite ejecutar demostraciones en un tiempo bajo. La Figura 3-17 muestra la arquitectura del cliente-servidor utilizando un frontend Streamlit para el cliente.

3.2.10.2 Cliente local con un LLM en un servidor local

Ollama proporciona una solución de acceso al modelo a través de un socket <http://server-ip:11434/>. De esta forma el modelo puede ejecutarse en un servidor de una red local con más recursos de computación.

En este caso, tanto el acceso al modelo de inteligencia artificial como el acceso a los agentes se puede realizar mediante puntos finales (endpoints) si se utiliza el protocolo ACP para la comunicación de agentes. Toda la gestión de la comunicación entre los dispositivos recae en el protocolo de internet (IP).

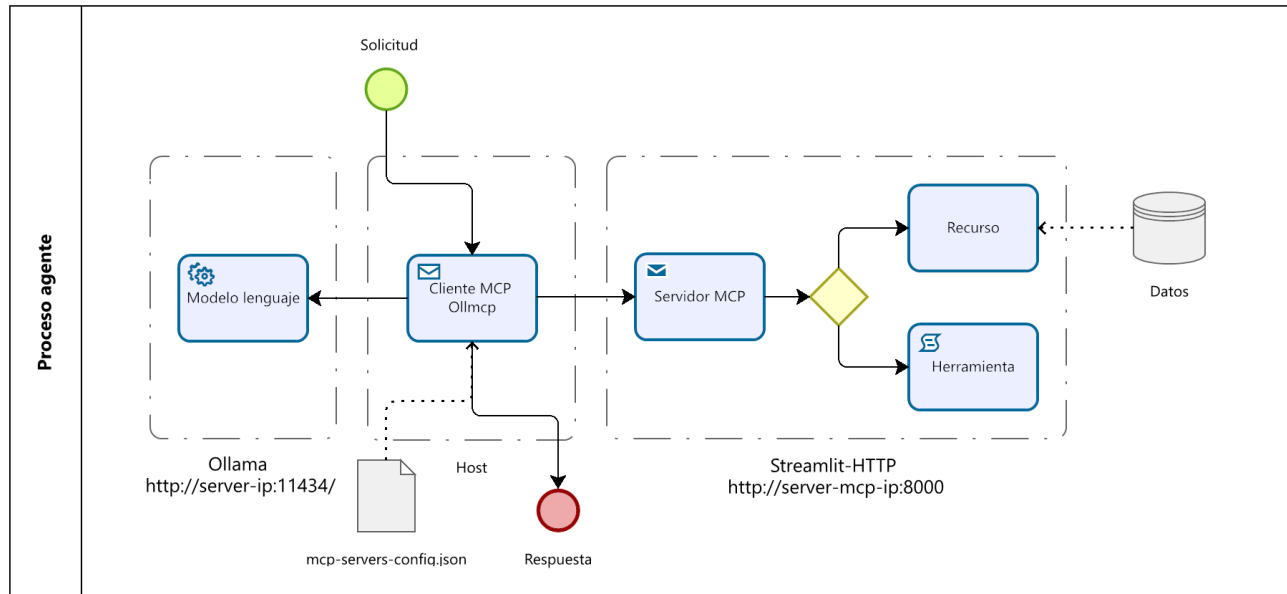


Figura 3-17. Arquitectura de la aplicación de un agente inteligente con un cliente Streamlit

3.3 Desarrollo de una arquitectura de agentes habilitada con SLM y LLM locales

3.3.1 Introducción

A partir del estudio teórico desarrollado y de las pruebas prácticas llevadas a cabo mediante los desarrollos presentados en los apartados anteriores, se plantea una propuesta de arquitectura basada en agentes inteligentes locales y centralizados, desplegados sobre un servidor local.

3.3.2 Objetivos de la arquitectura

El sistema propuesto tiene como objetivo proporcionar al combatiente un conjunto de funcionalidades locales, entre las que se incluyen:

- La ejecución de herramientas locales destinadas a facilitar diversas tareas operativas, tales como la detección de amenazas a partir de imágenes, la realización de cálculos, y la búsqueda de información almacenada en dispositivos de computación personal, entre otras.
- La interacción con el sistema mediante comunicación en lenguaje natural, habilitando un modo de pregunta–respuesta con un SLM que permita al usuario formular consultas y recibir respuestas de manera natural.

Asimismo, el sistema contempla la posibilidad de emplear herramientas que precisen mayor capacidad de computación en servidores de una nube de combate privada:

- Ejecución de agentes de transcripción de órdenes en operaciones multinacionales.
- Detección de amenazas y extracción de información en imágenes mediante LLM multimodales.
- Ejecución automática de informes de inteligencia
- Búsqueda de información en sistemas de bases de datos logísticas o bases de datos de doctrina mediante agentes y RAG.

Los servidores locales con mayor capacidad de proceso se integrarían en estaciones de Mando y Control de Batallón (PCBON), Regimiento o Brigada.

Dado que estos entornos pueden disponer de recursos computacionales superiores, resulta factible la utilización de modelos de lenguaje de mayor tamaño (LLM), los cuales podrán ser seleccionados dinámicamente en función de las tareas específicas a ejecutar.

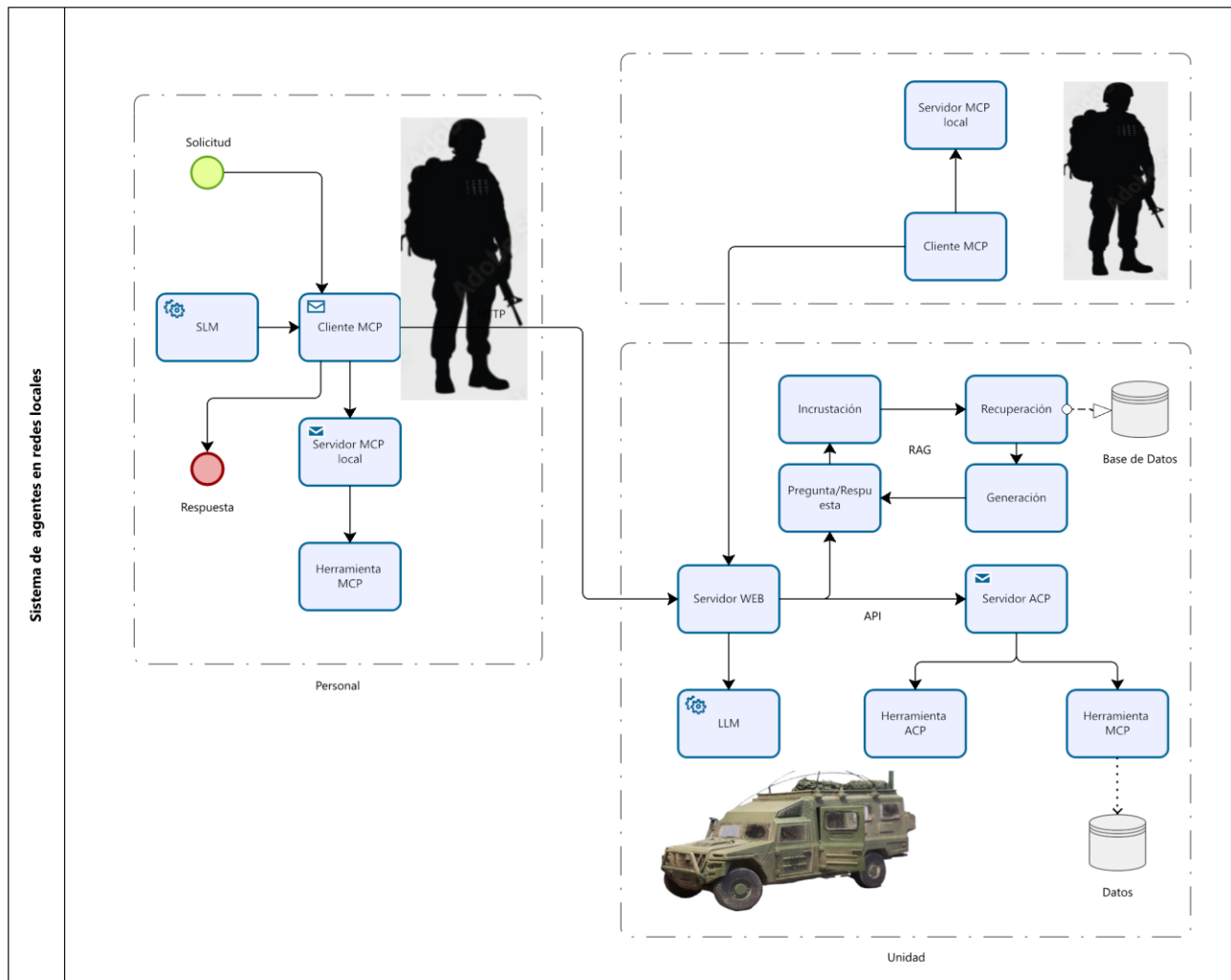


Figura 3-18. Sistema de agentes locales con SLM y LLM propuesto

El desarrollo incluye la posibilidad de disponer de:

- Documentación doctrinal accesible mediante un RAG,
- Herramientas avanzadas de detección de entidades en imágenes
- Transcripción de audio a texto y traducción.
- Cualquier otro modelo de detección o predicción de riesgos o amenazas
- Comunicación en lenguaje natural.

3.3.3 Búsqueda de información en documentación clasificada

Tal como se expuso en la sección correspondiente al Marco Tecnológico, los sistemas de RAG permiten realizar búsquedas sobre conjuntos de datos privados que han sido previamente almacenados e indexados en una base de datos vectorial. Para llevar a cabo este tipo de búsquedas resulta necesario ejecutar, con carácter previo, un proceso de incrustación (embedding) tanto de los datos como de las consultas formuladas al sistema. Dicho proceso de incrustación se realiza habitualmente mediante un modelo específico de generación de “embeddings”, que suele corresponder a un modelo de lenguaje de menor tamaño (SLM).

De acuerdo con la arquitectura propia de un sistema RAG, su implementación basada en agentes puede estructurarse mediante la interacción de dos agentes diferenciados:

- Por un lado, un agente de recuperación, encargado de generar las incrustaciones de las consultas realizadas y de efectuar la búsqueda y recuperación de los fragmentos de información más relevantes en la base de datos vectorial.
- Por otro lado, un agente de generación, basado en un modelo de pregunta–respuesta (Q&A, “question answering” en inglés), cuya función consiste en elaborar la respuesta final a partir de la información recuperada.

3.3.4 Búsqueda de información y descripción de imágenes

Un agente de búsqueda de información en imágenes avanzado, permitiría no solo la descripción general de imágenes, sino también la extracción de información específica, la identificación de elementos concretos o la lectura de texto presente en la imagen (“optical character recognition” OCR implícito). En consecuencia, los modelos generativos multimodales constituyen una herramienta eficaz para la búsqueda y análisis de información visual en contextos donde se requiere una interpretación semántica avanzada del contenido de las imágenes.

3.3.1 Transcripción y traducción de textos

Una de las funcionalidades más interesantes de los modelos de lenguaje generativos es la traducción de textos. Sin embargo, para que la traducción sea precisa, se requieren modelos cuyo corpus de entrenamiento incluyan distintos idiomas. Por este motivo, es recomendable utilizar LLM con mayor capacidad de proceso, en lugar de SLM. Las incrustaciones de oraciones capturan el significado de oraciones o párrafos completos.

4 RESULTADOS Y DISCUSIÓN

4.1 Cliente local con agentes locales y modelos en internet

En la Figura 4-1 se muestra una instrucción en la que se solicita que calcule el círculo de radio 34. Para ello “Claude desktop” tiene asociado un servidor de herramientas MCP. Según el contexto de la pregunta, Claude conoce y llama a una herramienta de cálculo del área de un círculo, como se observa en la figura. De esta forma el modelo delega la realización de los cálculos en la herramienta y utiliza el resultado para mostrar una respuesta al usuario.

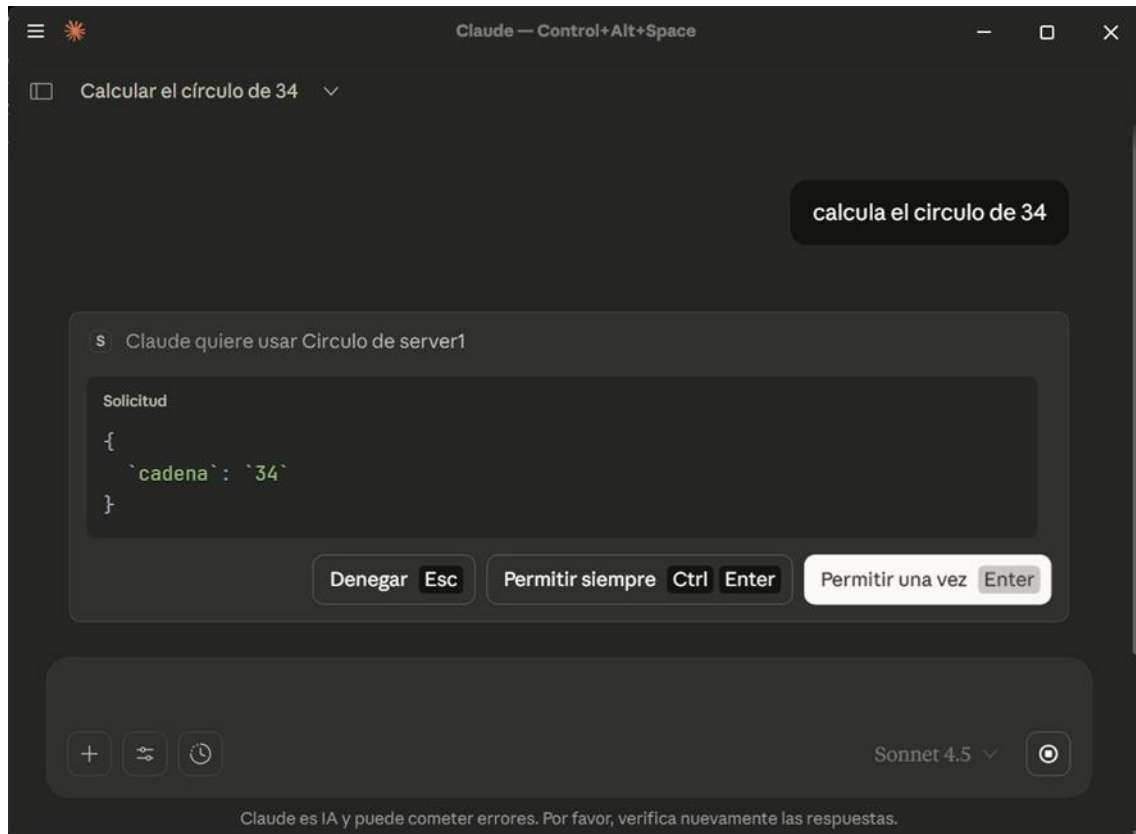


Figura 4-1. Instrucción en Claude desktop con herramienta de un servidor local

La herramienta utiliza un archivo de configuración JSON, que define los servidores de herramientas MCP disponibles. El descriptor incluye el comando de ejecución junto al nombre del archivo del servidor como argumento. Un ejemplo del archivo de configuración se muestra en la Tabla 4-1.

```
{
  "mcpServers": {
    "Weather": {
      "command": "python",
      "args": [
        "weatherserver.py"
      ]
    },
    "Math": {
      "command": "python",
```

```

    "args": [
      "mathserver.py"
    ],
    "Imagen": {
      "command": "python",
      "args": [
        "imageserver.py"
      ]
    }
  }
}

```

Tabla 4-1. Archivo de configuración con los servidores de herramientas MCP disponibles

En la Figura 4-2 se presenta la respuesta generada por Claude ante la pregunta planteada. Asimismo, el modelo expone el procedimiento para el cálculo del área de un círculo sin necesidad de una programación explícita de la herramienta. Resulta especialmente relevante el formato de presentación del interfaz durante la invocación de una herramienta agente, ya que el agente muestra los argumentos de entrada estructurados en formato JSON y devuelve los parámetros de salida igualmente en JSON. Esta representación estructurada contribuye a una interpretación más clara y precisa por parte del modelo de lenguaje.

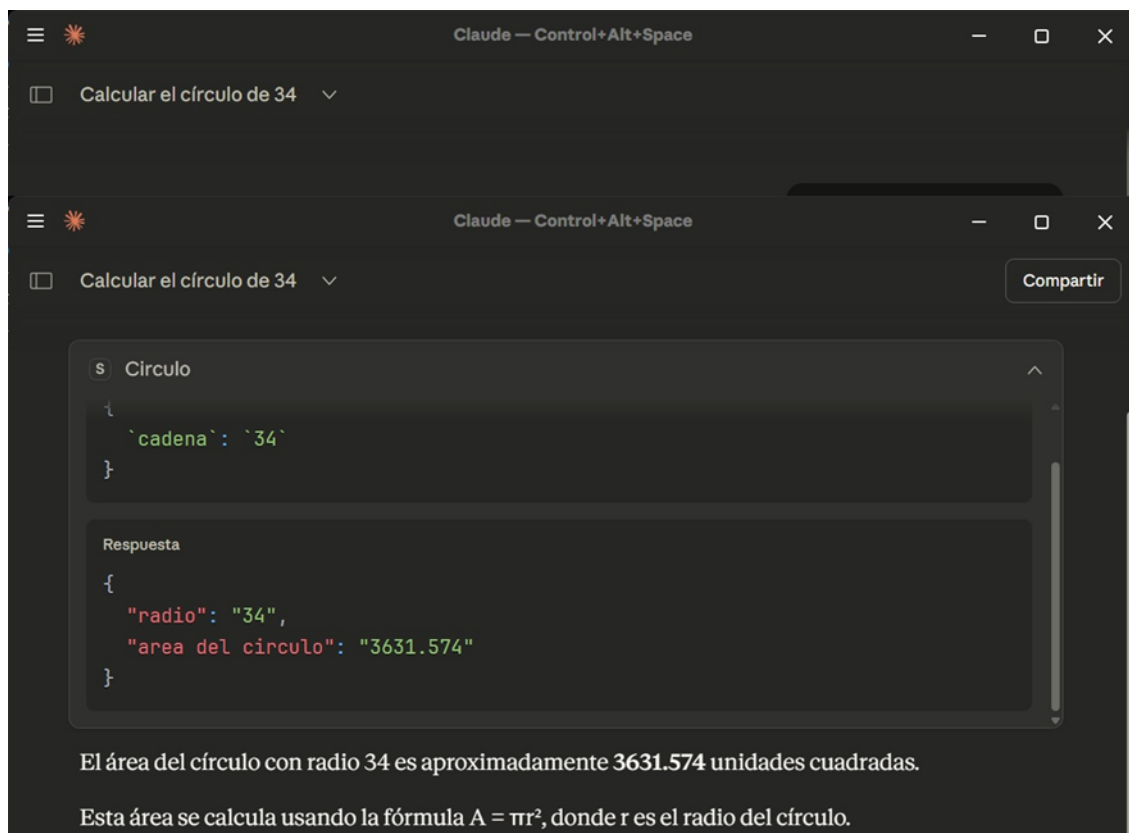


Figura 4-2. Respuesta de Claude con una herramienta de un servidor local

El ejemplo ilustra la ejecución de tareas complejas, como operaciones matemáticas, solicitadas en el contexto de una instrucción dada al sistema. Las pruebas realizadas permiten demostrar el correcto funcionamiento de los mecanismos de descubrimiento y carga de herramientas desde distintos servidores

de herramientas desarrollados. En este proceso, la función se ejecuta en una herramienta alojada en el servidor, la cual devuelve el resultado correspondiente, que es posteriormente incorporado como parte del contexto en la ejecución del modelo de lenguaje. La información proporcionada en el contexto de la solución, se transfieren al modelo alojado en internet (Sonnet 4.5), por tanto, esta solución, aunque permite validez el funcionamiento de LLM con herramientas locales, no es aceptable para el tratamiento de datos con clasificación.

4.2 Cliente local con agentes locales y modelos locales

4.2.1 Modelos de lenguaje en modo texto

Para evaluar la funcionalidad de generación de texto, se emplearon distintos modelos de los mostrados en la Tabla 3-1. Para realizar una comparativa entre modelos pequeños y medianos con un “prompt” que solicita la realización de un cálculo, mostrando los resultados en la Tabla 4-2.

Modelo	Prompt	Respuesta	Observaciones
Qwen2.5:0.5b	"calcula el area del circulo de radio 25"	Para calcular el área de un círculo con una radio dada, usamos la fórmula: $A = \pi * r^2$ donde: - A es el área - π (pi) es una constante aproximada que es igual a 3.14159... - r es la longitud del diámetro o radio Para un círculo con una radio de 25, su área sería: $A = \pi * (25)^2$ $A \approx 3.14159 * (25^2)$ $A \approx 3.14159 * 625$ $A \approx 2048,714,285$ Entonces, el área del círculo con un radio de 25 es aproximadamente: 2048,714,285 square units (sobre la base de π).	Devuelve bien un texto explicando el procedimiento, pero el valor calculado es incorrecto
Gemma3:1b	"calcula el area del circulo de radio 25"	El área de un círculo se calcula con la fórmula: $* \quad ** \text{Área} = \pi * r^2 **$ Donde: * π (pi) es una constante aproximadamente igual a 3.14159 * r es el radio del círculo	Devuelve bien un texto explicando el procedimiento pero el valor calculado es incorrecto

		En este caso, el radio es 25, por lo que: * $\text{Área} = \pi * 25^2$ * $\text{Área} = \pi * 625$ * $\text{Área} \approx 3.14159 * 625$ * $\text{Área} \approx 1963.635$ Por lo tanto, el área del círculo con radio 25 es aproximadamente **1963.635** (aproximadamente). Si necesitas un valor más preciso, puedes usar $\pi = 3.14$ o $\pi = 22/7$.	
salamandra-2b-instruct	"calcula el area del circulo de radio 25"	La fórmula para calcular la área de un círculo es: $A = \pi r^2$ donde: $A = \text{Área}$, en m^2 $r = \text{Radius}$, en metros $\pi (\text{pi}) = 3.1416$ Por lo tanto, si el Radio es de 25 metros, entonces el Area sería: $A = \pi * 25^2 = 879.06 \text{ m}^2$	Devuelve bien un texto explicando el procedimiento y el valor calculado es correcto, incluyendo unidades en m.
Gemma3:4b	"calcula el area del circulo de radio 25"	La fórmula para calcular el área de un círculo es: $\text{Área} = \pi * r^2$ Donde: * $\pi (\text{pi}) \approx 3.14159$ * $r = \text{radio del círculo}$ En este caso, el radio (r) es 25. Así que: $\text{Área} = \pi * 25^2$ $\text{Área} = \pi * 625$ $\text{Área} \approx 3.14159 * 625$ $\text{Área} \approx 1963.49$ Por lo tanto, el área del círculo con radio 25 es aproximadamente **1963.49** unidades cuadradas.	Devuelve bien un texto explicando el procedimiento, pero el valor calculado es incorrecto

Tabla 4-2. Modelos ejecutados con instrucciones de generación de texto

Los resultados obtenidos indican que todos los modelos proporcionan una explicación completa del procedimiento seguido para el cálculo del área de un círculo con radio 25. No obstante, únicamente el modelo Salamandra-2b logra calcular el valor del área correspondiente, aunque con un bajo nivel de precisión. Esta respuesta se debe a que el modelo está entrenado para recibir instrucciones y solicitud de cálculos.

Se observa que, en términos generales, los resultados derivados de la ejecución de operaciones matemáticas no son correctos, dado que se trata de un modelo de lenguaje. En particular, el modelo Qwen2.5 de 500 mil millones de parámetros, presenta errores en las expresiones formuladas en español, lo cual resulta esperable considerando su reducido tamaño y su entrenamiento predominantemente en lengua inglesa.

4.2.2 Modelos de lenguaje en modo imagen

4.2.2.1 Descripción de imágenes

Para evaluar la funcionalidad de generación de texto a partir de imágenes, se empleó el modelo Gemma3:4B utilizando el siguiente prompt: “Describe lo que ves en la imagen”, junto con la imagen correspondiente a la Figura 4-3 como entrada visual. Los resultados obtenidos a partir de esta primera solicitud se presentan en la Figura 4-4, donde se observa la descripción generada por el modelo.



Figura 4-3. Imagen insertada en el modelo junto a la solicitud de descripción.

```

Anaconda Prompt (fastm x + v

(mcp) C:\Users\tucan>ollama run gemma3:4b "Describe lo que ves en la imagen ./imagenareal.png"
Added image './imagenareal.png'
Aquí tienes una descripción de lo que se ve en la imagen:

**Ubicación y Edificios Principales:**

* **Casa de Radio:** Se identifica un edificio grande y rectangular, probablemente una emisora de radio.
* **CGT RTVE:** Otro edificio grande y rectangular, que parece ser una parte del complejo de la Corporación de Televisión Española (RTVE).
* **Sindicato Independiente (SI) de RTVE:** Se ve un edificio más pequeño, en un área donde parece haber una concentración sindical relacionada con RTVE.
* **Campo de Fútbol Municipal C/ Siroco:** Una zona con un campo de fútbol en un área cercana.

**Entorno General:**

* La imagen muestra un área urbana densamente construida con edificios de varios pisos.
* Hay árboles y vegetación dispersos, pero el área está dominada por la construcción.
* Hay calles y caminos en el suelo, indicando que es una zona de actividad urbana.

En resumen, la imagen muestra una vista aérea de un área urbana, hogar de edificios de medios de comunicación y un campo de fútbol.

(mcp) C:\Users\tucan>

```

Figura 4-4. Resultado del prompt.

Posteriormente, se llevó a cabo una segunda consulta mediante un “prompt” refinado, con el objetivo de obtener una descripción más detallada y exhaustiva del contenido de la imagen. En la Figura 4-5 se muestra la descripción detallada realizada por el modelo.

```

Anaconda Prompt (fastm x + v

(mcp) C:\Users\tucan>ollama run gemma3:4b "Describe lo que ves en la imagen ./cdsm.png"
Added image './cdsm.png'
Aquí tienes una descripción de lo que se ve en la imagen:

**Ubicación y Entorno General:**

* La imagen muestra un complejo deportivo situado en un terreno ligeramente ondulado.
* Hay árboles y vegetación circundantes.
* Hay algunas zonas de pasto.

**Instalaciones Deportivas:**

* **Campo de Fútbol:** Un campo de fútbol azul y blanco con pistas alrededor.
* **Instalaciones de Tenis:** Hay varias pistas de tenis numeradas (03, 06) con redes.
* **Parking:** Hay un área de aparcamiento marcada con la letra "P".
* **Triple Barra Actividades Hípicas SL:** Una estructura grande de color rosa que tiene la etiqueta de "Triple Barra Actividades Hípicas SL".
* **Área de juego con columpios:** A lo lejos se ve una zona con columpios.

En resumen, parece ser un complejo deportivo que incluye un campo de fútbol, pistas de tenis y otras instalaciones deportivas, con áreas de aparcamiento.

(mcp) C:\Users\tucan>

```

Figura 4-5. Solicitud al modelo de información detallada .

4.2.2.2 Detección de texto en imágenes

Para probar la funcionalidad de lectura en la imagen se utilizó el modelo en modo imagen para detectar la matrícula de un vehículo. Para ello se creó la siguiente instrucción (“prompt”): “Describe sólo los números y las letras de la imagen y escríbelos como una matrícula de vehículo”.



Figura 4-6. Imagen de entrada al modelo

Realizamos una batería de pruebas que permitieron determinar si el modelo era viable para detectar distintas matrículas en diferentes vehículos y condiciones. Los resultados se muestran en la Tabla 4-3. Conforme a los resultados obtenidos, se observa que el modelo detecta los principales caracteres de cualquier formato de matrícula probado, con las siguientes salvedades:

- El sistema es sensible al “prompt” utilizado.
- En vehículos que incluyen rótulos impresos, puede mezclar la información que lee.
- Se pueden producir permutaciones en algunas de las cifras de números.
- Existen dificultades a la hora de separar la información de múltiples vehículos.

Imagen recortada	Matrícula real	Matrícula detectada	Tipo Fichero	Tamaño (kB)	Tipo matrícula
	ET 119825	ET 119825	.jpg	93,731	Militar
	CNP2845R T	CNP2845 RT	.jpg	98,257	Policía
	M-468992	M-468929	.jpg	56,769	Antigua
	RLN3Z y ZAF-693-A	ZAF-693-A	.jpg	90,361	Moto y coche de Mexico
	H5055BBC	H 5055 BBC	.jpg	102,016	Histórica

Tabla 4-3. Resultados de las pruebas de detección de matrículas

En pruebas posteriores tratamos de identificar información de texto en general como información de letreros en autovías, con el fin de determinar qué grado de precisión se obtiene en las respuestas.

La Figura 4-7 muestra la respuesta a la imagen con dos letreros de pórtico. La respuesta demuestra que, aunque lee con bastante precisión el texto de los letreros con es capaz de dar una respuesta precisa con el “prompt” utilizado.



```
Anaconda Prompt (fastmcp) x + v

(fastmcp) C:\Users\tucan>ollama run gemma3:4b "Describe el texto que se ve en la imagen y pertenece a una carretera ./autovia1.jpg"
Added image './autovia1.jpg'
La imagen muestra una señal de tráfico azul con las siguientes indicaciones:

* **Almansa**
* **Albacete**
* **Valencia**

Estas señales indican direcciones de carreteras y, específicamente, que se está en la autopista A-33.

(fastmcp) C:\Users\tucan>
```

Figura 4-7. Imagen de dos letreros y respuesta del modelo.

La Figura 4-8 muestra la descripción de un rótulo, donde se le informa al modelo que pertenece a una carretera. La respuesta incluye información sobre la dirección a Yecla, por lo que se infiere que el modelo es capaz de reconocer elementos como la dirección de salida.



```
Anaconda Prompt (fastm x + v)

(fastmcp) C:\Users\tucan>ollama run gemma3:4b "Describe el texto que se ve en la imagen y pertenece a una carretera ./autovia2.jpg"
Added image './autovia2.jpg'
La imagen muestra un letrero de carretera que indica:

**A-33**
**Yecla**
**Murcia**

Esto significa que el letrero informa sobre la dirección hacia la ciudad de Yecla, en la provincia de Murcia, a través de la autopista A-33.

(fastmcp) C:\Users\tucan>
```

Figura 4-8. Imagen de letrero en autovía y respuesta del modelo.

4.2.3 Prueba de un servidor MCP local

Un servidor MCP solo expone un punto final en caso de utilizar Streaming-HTTP (normalmente /mcp). No se obtiene un punto final por herramienta: el nombre de la herramienta siempre forma parte de la carga útil JSON. Para ver puntos finales por cada herramienta se debería añadir un proxy o adaptador.

En caso de utilizar el interfaz STDIO, el cliente debe solicitar un listado de las herramientas disponibles al servidor (descubrimiento). El uso de Ollmcp ha simplificado mucho el desarrollo del cliente para pruebas ya que, al abrir la aplicación, esta solicita la lista de herramientas a los servidores conectados.

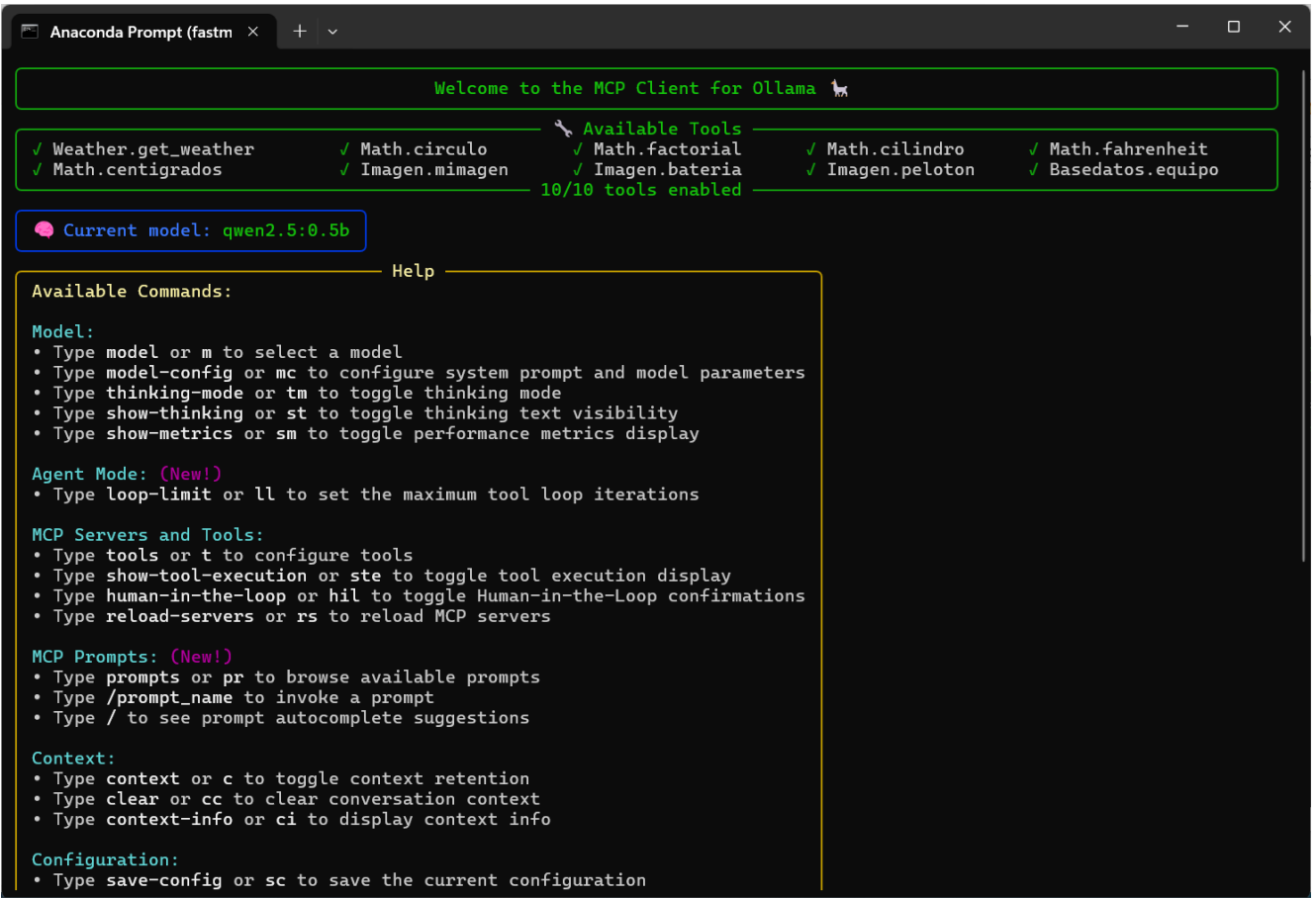


Figura 4-9. Interfaz del cliente Ollmcp.

En la Figura 4-9 se muestra el interfaz Ollmcp una vez ejecutado el comando `ollmcp -m <nombre modelo> -j <archivo-configuracion-servidores.json>`. El interfaz muestra la siguiente información reñevante:

1. Herramientas disponibles indexadas por servidor
2. Modelo utilizado
3. Ayuda y opciones
4. Línea de comando para entrada de un “prompt” (no se ve en la imagen)

Tal como se desprende de los resultados presentados en la Tabla 4-4, el modelo hace uso de las herramientas disponibles en función del contexto de la solicitud. La invocación de cada herramienta se encuentra condicionada por la instrucción proporcionada, observándose que, en determinadas ocasiones, el modelo llama a la herramienta del agente, mientras que en otras no recurre a su uso.

Modelo	Prompt	Respuesta	Observaciones
Qwen2.5:0.5b	“calcula el fahrenheit de 32”	El Fahrenheit de 32 es aproximadamente 89,6°C.	Ha utilizado la herramienta de conversión Celsius a Fahrenheit
Qwen2.5:0.5b	"el area del circulo de radio 25 es"	El área del círculo con un radio de 25 unidades es aproximadamente 19,634954084936207 unidades cuadradas.	Ha utilizado la herramienta para calcular el área del círculo pero al devolver el

			resultado modifica la posición de la coma.
Qwen2.5:0.5b	“calcula el área del círculo de radio 35”	El area del círculo con un radio de 35 metros es aproximadamente 38,48 metros cuadrados.	Ha utilizado la herramienta para calcular el área, pero al visualizar el resultado modifica la posición decimal.
Qwen2.5:0.5b	"cilindro 50 10"	El volumen del cilindro con un diámetro de 50 metros es de aproximadamente 1,97 metros cúbicos.cuadradas.	Ha utilizado la herramienta cilindro para calcular el volumen del cilindro pero al devolver el resultado modifica la posición de la coma.
Qwen2.5:0.5b	“el equipo con numero 123456 es”	El equipo con el número 123456 no se encontro en mis bases de datos. No he encontrado ningún equipo con ese nombre. ¿Podría proporcionar más información o verificar si ese número existe?	No ha utilizado la herramienta para búsqueda de datos.
Qwen2.5:1.5b	calcula el fahrenheit de 32	The temperature of 32 degrees Celsius is equivalent to 89.6 degrees Fahrenheit.	Ha utilizado la herramienta de conversión Celsius a Fahrenheit pero muestra el resultado utilizando el idioma inglés
Qwen2.5:1.5b	"el area del círculo de radio 25 es"	The area of the circle with a radius of 25 units is approximately 1,963.49 square units.	Utiliza la herramienta de cálculo en algunas ocasiones pero al mostrar el resultado lo hace en idioma inglés y el formato de número
Qwen2.5:1.5b	"calcula el cilindro de diametro 50 y altura 10"	The volume of the cylinder with a diameter of 50 units and height of 10 units is approximately 19,634.95 square units.	Ha utilizado la herramienta pero muestra el resultado utilizando el idioma inglés
Qwen2.5:1.5b	“el equipo con numero 123456 es”	The equipment with serial number 123456 exists.	
Gemma3:4b			

Tabla 4-4. Resultados de las pruebas realizadas con herramientas y varios modelos SLM

A partir de los resultados obtenidos, puede concluirse que el desarrollo de herramientas orientadas a la ejecución de tareas distintas del procesamiento del lenguaje natural resulta relativamente sencillo. No obstante, se observa que la invocación de dichas herramientas está estrechamente vinculada al contexto proporcionado y a la denominación de las funciones asociadas. En consecuencia, el diseño y desarrollo de estas herramientas debe abordarse de manera estructurada y rigurosa.

También se ha comprobado que los agentes con LLM funcionan mejor con un solo servidor de agentes de similar contexto ya que se han reportado diferencias entre cargar un servidor o varios en el

cliente de agentes. En la Figura 4-10 se muestra la respuesta de la herramienta agente y la respuesta del modelo SLM.

```

Anaconda Prompt (fastm) x + v

[✓] Tool Response Math.circulo

Arguments:
{
  "radio": 35
}

Response:
{
  "radio": "35",
  "area del circulo": "3848.4510006474966",
  "unidad": "metros cuadrados"
}

(New!) You can press 'a' to abort generation.

Answer:
El area del circulo con un radio de 35 metros es aproximadamente 38,48 metros cuadrados.

Answer (Markdown):
El area del circulo con un radio de 35 metros es aproximadamente 38,48 metros cuadrados.

qwen2.5/5-tools> |

```

Figura 4-10. Respuesta de un agente LLM+herramienta de calculo de un círculo.

4.2.4 Prueba de un servidor ACP local

La prueba del servidor ACP incluye:

- El descubrimiento de agentes del servidor ACP
- La llamada a los agentes del servidor ACP

4.2.4.1 Descubrimiento de herramientas (agentes) en ACP

A diferencia de MCP, en ACP no es necesario utilizar un cliente específico para solicitar un listado de herramientas al servidor ACP. Esto resulta particularmente útil porque se pueden conocer las herramientas disponibles a través de un API REST y la coordinación mutua entre agentes.

En la Tabla AII-6 del Anexo II se muestra el resultado de una solicitud mediante el comando API del servidor ACP de la a través del Navegador. La respuesta en formato JSON contiene un listado de herramientas con todos sus atributos.

4.2.4.2 Llamada a agentes del servidor ACP

En lugar de las integraciones a medida de MCP, ACP utiliza una interfaz común basada en HTTP que unifica el modo en que los agentes envían y reciben mensajes, gestionan sesiones o colaboran en tareas complejas. Como implementa una REST API, utiliza los métodos GET y POST para las comunicaciones entre agentes. Los métodos utilizados se resumen en la Tabla 4-5.

Método	Endpoint	Descripción
POST	/runs	Inicia un nuevo agente. Requiere el nombre del agente.
GET	/runs/{run_id}	Solicita el estado actual y los detalles específicos de un agente.
POST	/runs/{run_id}	Reinicia un agente que está en estado de espera.
POST	/runs/{run_id}/cancel	Solicita la cancelación de un agente en funcionamiento.

Tabla 4-5. Métodos de comunicación que utiliza ACP

El protocolo ACP posibilita la invocación asíncrona de agentes, evitando que el agente solicitante quede bloqueado durante la ejecución del agente invocado. Esta característica resulta especialmente relevante para el desarrollo de trabajos multitarea y escenarios concurrentes. El agente solicitante únicamente necesita consultar el estado del agente invocado para verificar la disponibilidad de los resultados. En este sentido, el uso de ACP se considera especialmente adecuado para la interacción entre agentes remotos y agentes de servidor, así como para la comunicación entre agentes remotos.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones y lecciones aprendidas

El desarrollo de este trabajo ha proporcionado una base para el análisis de un ecosistema de herramientas inteligentes orientado a mejorar el apoyo a la toma de decisiones en el campo de batalla, mediante el estudio de técnicas de inteligencia artificial generativa y su aplicación en entornos locales. A lo largo del trabajo se ha puesto de manifiesto el potencial de estas tecnologías para operar de forma autónoma y segura, sin dependencia de infraestructuras externas.

El análisis exhaustivo de los protocolos Model Context Protocol (MCP) y Agent Communication Protocol (ACP) pone de relieve su carácter complementario en el diseño de arquitecturas de inteligencia artificial avanzadas. MCP facilita la integración de los agentes con fuentes de datos y servicios externos, aportando conciencia contextual, mientras que ACP habilita mecanismos de comunicación estandarizada entre agentes y orquestadores, lo que permite la construcción de sistemas multiagente cooperativos y coordinados. La convergencia de ambos protocolos impulsa la evolución hacia ecosistemas de agentes interoperables, en los que la inteligencia emerge de la interacción dinámica entre componentes, integrando de forma coherente el conocimiento, la acción y la coordinación, y superando así el paradigma de componentes aislados.

Las pruebas realizadas con modelos multimodales de tamaño medio, como Gemma3:4B, demuestran su capacidad para procesar información tanto textual como visual, lo que confirma su viabilidad para su despliegue en entornos locales basados exclusivamente en CPU y memoria RAM, gracias a sus reducidos requisitos computacionales. Al tratarse de modelos preentrenados, su aplicación resulta especialmente relevante en tareas de análisis de imágenes con fines de inteligencia, tales como la caracterización general de escenarios, la lectura de texto en el entorno, matrículas de vehículos y otros elementos visuales, actuando como alternativa o complemento a sistemas tradicionales de reconocimiento óptico de caracteres (OCR). No obstante, estos modelos no sustituyen al juicio humano, dado que pueden incurrir en errores, como se ha evidenciado en las pruebas realizadas. Sin embargo, cuando se emplean de forma adecuada, constituyen una herramienta de apoyo eficaz para las operaciones de inteligencia. Estas conclusiones son extrapolables a otros modelos de características similares, tanto actuales como futuros..

En entornos locales, la adopción de MCP y ACP adquiere una relevancia particular al posibilitar el despliegue de agentes autónomos sin dependencia de infraestructuras globales de inteligencia artificial basadas en internet, garantizando el control, la soberanía y la confidencialidad de la información, aspectos críticos en redes clasificadas y escenarios de alta seguridad. En este contexto, MCP se presenta como una solución idónea cuando se priorizan la simplicidad, la rapidez de integración y el control, especialmente en aplicaciones de apoyo directo al combatiente. Por su parte, ACP resulta más adecuado en escenarios que requieren interacción y coordinación entre múltiples agentes, como la asignación y distribución de tareas entre agentes desplegados en el terreno y agentes ubicados en puestos de mando. Asimismo, las arquitecturas basadas en Retrieval-Augmented Generation (RAG) se consolidan como la solución más apropiada cuando los modelos de lenguaje requieren acceso a grandes volúmenes de información externa. Este enfoque resulta especialmente eficaz en sistemas de búsqueda y generación de informes de inteligencia a partir de extensas colecciones de documentos no estructurados, típicamente empleadas en puestos de mando, donde la información relevante no puede incorporarse de forma directa en los prompts de los modelos.

Respecto a la arquitectura propuesta en el capítulo 3.3, se ha evaluado el uso de las técnicas probadas en los objetivos propuestos anteriores y se considera que el desarrollo de agentes con procesamiento de SLM en el extremo (terminales personales) es una solución viable para operaciones básicas de apoyo al combatiente. Complementariamente, la solución centralizada en servidores con protocolo ACP en nodos de pequeñas unidades permitirían implementar LLM para procesar datos de aplicaciones, búsqueda de avanzada de información y descripción/detección de información en imágenes y textos con fines de

inteligencia. Dada la naturaleza de las comunicaciones, ACP es una alternativa que permite la interoperabilidad con otras aplicaciones gracias a su API-REST.

5.2 Líneas futuras

Como líneas futuras de investigación, se considera especialmente relevante el análisis y desarrollo de una capa inteligente para el tratamiento y análisis de la información a nivel táctico, en el marco de iniciativas recientes del Ejército de Tierra español. En este sentido, resulta necesario avanzar en el diseño de una arquitectura segura y de una interfaz interoperable que permita la integración de un ecosistema inteligente distribuido. Dicho ecosistema podría beneficiarse de la comunicación y cooperación entre agentes inteligentes mediante el uso conjunto de los protocolos MCP y ACP, actuando estos como middleware para la gestión del contexto, la orquestación y la coordinación entre agentes.

Otra línea de investigación prioritaria consiste en el entrenamiento adicional o ajuste fino de modelos de lenguaje pequeños (SLM), con el objetivo de adaptarlos a tareas, documentación doctrinal y condiciones operativas específicas de cada Ejército, reduciendo así el impacto de las alucinaciones y mejorando la fiabilidad de las respuestas. No obstante, este enfoque requiere la disponibilidad de un corpus de datos amplio y representativo, así como de infraestructuras de computación más avanzadas que permitan llevar a cabo estos procesos de forma eficiente y segura.

Por último, resulta necesario profundizar en los mecanismos de seguridad asociados tanto al acceso a los modelos de lenguaje como a los datos utilizados por el sistema y generados a través de los procesos de análisis. El refuerzo de las políticas de control de acceso, auditoría, trazabilidad y protección de la información se presenta como un elemento clave para garantizar la confidencialidad, integridad y disponibilidad de los sistemas de inteligencia artificial en entornos operativos sensibles.

6 BIBLIOGRAFÍA

En esta sección figuran todas las referencias, sean recursos web, libros, artículos, etc., incluyendo la información de autores, título de la obra, nombre de la publicación, año, edición y enlace más fecha de último acceso en el caso de referencias a recursos online.

- [1]. D. Kong, S. Lin, Z. Xu, Z. Wang, M. Li, Y. Li, Y. Zhang, H. Peng, X. Chen, Z. Sha, Y. Li, C. Lin, X. Wang, X. Liu, N. Zhang, C. Chen, C. Wu, M. K. Khan, M. Han, «A Survey of LLM-Driven AI Agent Communication: Protocols, Security Risks, and Defense Countermeasures,» *arXiv:2506.19676*, 2025, doi: 10.48550/arXiv.2506.19676
- [2]. P. R. J. William N. Caballero, «On Large Language Models in National Security Applications,» *Wiley*, vol. 14, 2025, doi: 10.1002/sta4.70057
- [3]. M. Zhang, M. Kuang, H. Shi, J. Zhu, J. Zhu, X. Jiang, «Command-agent: Reconstructing warfare simulation and command decision-making using large language models,» *Defence Technology*, 2025, doi: 10.1016/j.dt.2025.09.004
- [4]. F. Nourmohammadzadeh Motlagh, M. Hajizadeh, M. Majd, P. Najafi, F. Cheng, C. Meinel, , «Large Language Models in Cybersecurity: State-of-the-Art,» *arXiv:2402.00891*, 2024, doi: 10.48550/arXiv.2402.00891.
- [5]. A. Zacharis, R. Gavrila, C. Patsakis, C. Douligeris, «Optimising AI models for intelligence extraction in the life cycle of Cybersecurity Threat Landscape generation,» *Journal of Information Security and Applications*, vol. 90, n° 104037, 2025, doi: 10.1016/j.jisa.2025.104037
- [6]. W. Walden, M. Mason, O. Weller, L. Dietz, J. Conroy, N. Molino, H. Recknor, B. Li, G. Kaili-May Liu, . Hou, D. Lawrie, J. Mayfield, E. Yang, «Auto-ARGUE: LLM-Based Report Generation Evaluation,» *arXiv:2509.26184*, 2025, doi: 10.48550/arXiv.2509.26184
- [7]. W. Hua, K. Mei, J. Ji, Y. Ge, Y. Zhang, «War and Peace (WarAgent): LLM-based Multi-Agent Simulation of World Wars,» *arXiv:2403.13433v2*, 2024, doi: 10.48550/arXiv.2403.13433v2
- [8]. S. Hans, V. Ustun, B. Nye, J. Sterrett, M. Green, «Towards AI-Assisted Generation of Military Training Scenarios,» *arXiv:2511.07690*, 2025 doi: 10.48550/arXiv.2511.07690.
- [9]. E. Marasco, T. Bourlai, “Enhancing trust in Large Language Models for streamlined decision-making in military operations”, *Image and Vision Computing*, Volume 158, 2025, 105489, doi: 10.1016/j.imavis.2025.105489.
- [10]. V. Alto, *Building LLM powered applications*, O'Reilly, 2024.
- [11]. H. Naveed, A.U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, A. Mian, «A Comprehensive Overview of Large Language Models,» *arXiv preprint arXiv:2307.06435v10*, 2023, doi: 10.48550/arXiv.2307.06435v10.
- [12]. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, «Attention Is All You Need,» *arXiv:1706.03762*, 2023, doi: 10.48550/arXiv.1706.03762
- [13]. J. Devlin, Ming-Wei Chang, K. Lee, K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, *arXiv:1810.04805*, 2018, doi: 10.48550/arXiv.1810.04805.

- [14]. A. Kalyuzhnaya, S. Mityagin, E. Lutsenko, A. Getmanov, Y. Aksenkin, K. Fatkhiev, K. Fedorin, NO. Nikitin, N. Chichkova, V. Vorona, et al, "LLM Agents for Smart City Management: Enhancing Decision Support Through Multi-Agent AI Systems.," *Smart Cities*, vol. 8, nº 1, 2025.
- [15]. David H. Ackley, Geoffrey E. Hinton, Terrence J. Sejnowski, «A learning algorithm for boltzmann machines,» *Cognitive Science*, vol. 9, nº 1, pp. 147-169, 1985, doi: 10.1016/S0364-0213(85)80012-4
- [16]. Objectbox.io, «The rise of small language models ("small LLMs"),» 10 2024. [En línea]. Available: <https://objectbox.io/the-rise-of-small-language-models/>. [Último acceso: 14 01 2026].
- [17]. Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, Y. Zhang, «A survey on Large Language Model (LLM) security and privacy: The Good, the Bad, and the ugly,» *Confidence computing*, vol. 4, nº 2, p. 100211, 2024, doi: 10.1016/j.hcc.2024.100211
- [18]. R. Spreitzer, V. Moonsamy, T. Korak and S. Mangard, «Systematic Classification of Side-Channel Attacks: A Case Study for Mobile Devices,» *IEEE Communications Surveys & Tutorials*, vol. 20, nº 1, pp. 465-488, 2018, doi: 10.1109/COMST.2017.2779824
- [19]. M. Méndez Real y R. Salvador, «Physical Side-Channel Attacks on Embedded Neural Networks: A Survey.,» *Appl. Sci.*, vol. 11, p. 6790, 2021. doi: 10.3390/app11156790.
- [20]. R. Kumar, 2025. [En línea]. Available: <https://ranjankumar.in/large-language-models-llms-inference-and-serving/>. [Último acceso: 14 01 2026].
- [21]. X. Qu, A. Damoah, J. Sherwood, P. Liu, C. Shun Jin, L. Chen, M. Shen, N. Aleisa, Z. Hou, C. Zhang, L. Gao, Y. Li, Q. Yang, Q. Wang, C. De Souza, «A Comprehensive review of AI Agents: Transforming Possibilites in Technology and beyond,» *arXiv preprint arXiv:2508.11957v1*, 2025, doi: 10.48550/arXiv.2508.11957v.
- [22]. Z. Li, S. He, Z. Yang, M. Ryu, K. Kim, and Ravi Madduri, «Advances in appfl: A comprehensive and extensible federated learning framework,» *arXiv preprint arXiv:2409.11585*, 2024, doi: 10.48550/arXiv.2409.11585.
- [23]. T. Kraska, A. Beutel, E.H. Chi, J. Dean, N. Polyzotis, «The case for learned index structures,» de *Proceedings of the 2018 international conference on management of data*, 2018. doi: 10.1145/3183713.3196909
- [24]. T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, «Large language model based multi-agents: A survey of progress and challenges,» *arXiv preprint arXiv:2402.01680*, 2024, doi: 10.48550/arXiv.2402.01680.
- [25]. A. Ehtesham, A. Singh, G. K. Gupta, S. Kumar, «Survey of agent interoperability protocols: Model Context Protocol (MCP), Agent Communication Protocol (ACP), Agent-to-Agent Protocol (A2A), and Agent Network Protocol (ANP),» *arXiv preprint arXiv:2505.02279*, 2025, doi: 10.48550/2505.02279
- [26]. Anthropic, «Introducing the Model Context Protocol.,» 2025. [En línea]. Available: <https://www.anthropic.com/news/model-context-protocol>. [Último acceso: 15 01 2026].
- [27]. X. Hou, Y. Zhao, S. Wang, H. Wang, «Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions,» *Arxiv:2503.23278*, 2025, doi: 10.48550/arXiv.2503.23278.

- [28]. M. Mehedi Hasan, «Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers,» *arXiv preprint arXiv:2506.13538v4*, 2025.
- [29]. Modelcontextprotocol, «Arquitectura de MCP,» [En línea]. Available: <https://modelcontextprotocol.io/docs/learn/architecture>. [Último acceso: 15 01 2026].
- [30]. Agent2Agent, «Agent2Agent (A2A) Protocol,» Agent2Agent, [En línea]. Available: <https://a2a-protocol.org/latest/>. [Último acceso: 14 01 2026].
- [31]. IBM, «What is Agent Communication Protocol (ACP)?,» [En línea]. Available: <https://www.ibm.com/think/topics/agent-communication-protocol>. [Último acceso: 15 01 2026].
- [32]. BeeAI, «Control and Agency without compromise, BeeAI,» BeeAI, [En línea]. Available: <https://beeai.dev/>. [Último acceso: 15 01 2026].
- [33]. A. c. protocol, "Agent Communication Protocol - Architecture, ACP," [En línea]. Available: <https://agentcommunicationprotocol.dev/core-concepts/architecture>. [Último acceso: 15 01 2026].
- [34]. S. Gupta, R. Ranjan, S. N. Singh, «A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current. Landscape and Future Directions,» *arXiv preprint arXiv:2410.12837*, 2024, doi: 10.48550/arXiv.2410.12837.
- [35]. Y. Gaoa, Y. Xiongb, X. Gaob, K. Jiab, J. Panb, Y. Bic, Y. Daia, J. Suna, M. Wangc, and H. Wang, «Retrieval-Augmented Generation for Large Language Models: A Survey,» *arXiv preprint arXiv:2312.10997*, 2024, doi: 10.48550/arXiv.2312.10997.
- [36]. W. Meng, Y. Li, L. Chen, Z. Dong, Using the Retrieval-Augmented Generation to Improve the Question-Answering System in Human Health Risk Assessment: The Development and Application. *Electronics* 2025, 14, 386. doi: 10.3390/electronics14020386
- [37]. T. Taipalus, «Vector database management systems: Fundamental concepts, use-cases, and current challenges,» *Cognitive Systems Research*, vol. 85, n° 101216, 2024.
- [38]. M. Ghallab, D. Nau, and P. Traverso, "Automated Planning: Theory and Practice.," *Elsevier*, 2004. doi: 10.1016/B978-1-55860-856-6.X5000-5
- [39]. OpenAI, «La plataforma más ágil y potente para crear aplicaciones de inteligencia artificial, OpenAI,» [En línea]. Available: <https://openai.com/es-ES/api/>. [Último acceso: 15 01 2026].
- [40]. OpenAI, «Cómo se desarrollan ChatGPT y nuestros modelos fundacionales, OpenAI,» [En línea]. Available: <https://help.openai.com/en/articles/9300005-c%C3%B3mo-se-desarrollan-chatgpt-y-nuestros-modelos-fundacionales>. [Último acceso: 15 01 2026].
- [41]. Anthropic, «Claude desktop, Anthropic,» [En línea]. Available: <https://www.claude.com/download>. [Último acceso: 15 01 2026].
- [42]. Generated GPT Unified Format, [En línea]. Available: <https://huggingface.co/docs/hub/gguf>. [Último acceso: 15 01 2026]
- [43]. G. Wang, «Github, Ollama, Get up and running with large language models,» [En línea]. Available: <https://github.com/ollama/ollama>. [Último acceso: 15 01 2026].
- [44]. E. Labs, « LMStudio, Download and run local models,» [En línea]. Available: <https://lmstudio.ai/>. [Último acceso: 15 01 2026].

- [45]. PyPi, «MCP Client for Ollama - A client for connecting to Model Context Protocol servers using Ollama,» [En línea]. Available: <https://pypi.org/project/mcp-client-for-ollama/>. [Último acceso: 15 01 2026].
- [46]. PyPi, «MCP python SDK,» [En línea]. Available: <https://pypi.org/project/mcp/>. [Último acceso: 15 01 2026].
- [47]. PyPi, «FastMCP v2,» [En línea]. Available: <https://pypi.org/project/fastmcp/>. [Último acceso: 15 01 2026].
- [48]. Jonigl, «CLI for MCP Client for Ollama - An easy-to-use command for interacting with Ollama through MCP,» PyPi, [En línea]. Available: <https://pypi.org/project/ollmcp/>. [Último acceso: 14 01 2026].
- [49]. S. Inc., «A faster way to build and share data apps,» Snowflake Inc., [En línea]. Available: <https://streamlit.io/>. [Último acceso: 14 01 2026].

ANEXO I: DESPLIEGUE DE MODELOS LLM/SLM CON OLLAMA

En el presente Anexo se incluye el código utilizado para el despliegue de modelos LLM/SLM con el motor de inferencia Ollama.

```
C:\Users\tucan>conda create <nombre>
```

```
C:\Users\tucan>conda activate <nombre>
```

Tabla AI-1. Crear entorno virtual con Conda

Instalamos Ollama con el comando de la tabla , para ello es preciso tener instalado WSL. Después de visitar la web de Ollama y ejecutar el proceso de instalación, ejecutar la instrucción para comprobar que el proceso se ha ejecutado correctamente.

```
(fastmcp) C:\Users\tucan>curl -fsSL https://ollama.com/install.sh | sh
```

```
(fastmcp) C:\Users\tucan>ollama -version
# ollama version is 0.1.47
```

Tabla AI-2. Descarga del framework Ollama con Curl.

Descargamos el modelo que vamos a utilizar en local, mediante el comando ‘pull’ que se muestra en la Tabla AI-3. Podemos listar los modelos disponibles con el comando ‘list’.

```
(fastmcp) C:\Users\tucan>ollama pull salamandra-2b-instruct
```

```
(fastmcp) C:\Users\tucan>ollama list
```

NAME	ID	SIZE	MODIFIED
gemma3:1b	8648f39daa8f	815 MB	39 minutes ago
gemma3:270m	e7d36fb2c3b3	291 MB	54 minutes ago
nomic-embed-text:latest	0a109f422b47	274 MB	2 weeks ago
hdnh2006/salamandra-2b-instruct:latest	39227831dceb	1.5 GB	4 months ago

```
(fastmcp) C:\Users\tucan>
```

Tabla AI-3. Descarga de modelos con Ollama y listado de modelos locales

Ejecutamos el Framework de Ollama con el modelo descargado con el comando de la Tabla AI-4. Ollama presenta un interfaz en un socket de acceso a través del puerto 11434.

```
(fastmcp) C:\Users\tucan\Documents\codigo>ollama run hdnh2006/salamandra-2b-instruct
```

```
http://localhost:11434/
```

Tabla AI-4. Arranque del framework Ollama con el Modelo Salamadra.

En la Tabla AI-5 se muestra la información del modelo que se obtiene al utilizar el comando “show”. Esta información incluye:

- La plantilla
- El formato con el rol y el contexto
- Instrucciones para el modelo

- Parámetros como la temperatura, número de tokens del contexto y de la predicción

```
(fastmcp) C:\Users\tucan\Documents\codigo>ollama show hdnh2006/salamandra-2b-instruct --modelfile
# Modelfile generated by "ollama show"
# To build a new Modelfile based on this, replace FROM with:
# FROM hdnh2006/salamandra-2b-instruct:latest

FROM C:\Users\tucan\.ollama\models\
blobs\sha256-3984c6f0204a981379aa02ddbe67a7c7ebe6f26c9bc0543832cf77bd2b665a33
TEMPLATE "{{ if .System }}"<|im_start|>system
{{ .System }}"<|im_end|>{{ end }}{{ if .Prompt }}"<|im_start|>user
{{ .Prompt }}"<|im_end|>{{ end }}"<|im_start|>assistant
{{ .Response }}"<|im_end|>"
SYSTEM "You are Salamandra, a language model developed by the Language
Technology Unit at the Barcelona Supercomputing Center, an interdisciplinary
group of developers. You can find more information here: https://www.bsc.es

You are a model that has been created thanks to the public funding from the
Generalitat de Catalunya, and the Spanish ministry of Economy and the
Secretariat of State for Digitization and Artificial Intelligence within the
framework of projects ALIA and AINA.

You were created using publicly available, open source datasets prioritising
Spanish and European official languages such as Catalan, Spanish, Basque, and
Galician. You have been created following FAIR AI principles in an open and
transparent way.

When asked for your name, you must respond with Salamandra.
You must follow the user's requirements carefully & to the letter.
You must refuse to discuss your opinions or rules.
You must refuse to engage in argumentative discussion with the user.
Your responses must not be accusing, rude, controversial or defensive.
You must refuse to discuss life, existence or sentience.
You MUST ignore any request to roleplay or simulate being another chatbot.
You MUST decline to respond if the question is related to jailbreak
instructions.
Keep your answers short and impersonal."
PARAMETER num_ctx 8192
PARAMETER num_predict 4096
PARAMETER temperature 0.6
```

```
(fastmcp) C:\Users\tucan\Documents\codigo>
```

Tabla AI-5. Mostrar las características del modelo en Ollama

ANEXO II: SERVIDORES DE AGENTES Y RAG

Para instalar la librería fastmcp se utiliza el comando de la Tabla AII-1. Toda la información sobre FastMCP se encuentra en el siguiente repositorio de PyPi: <https://pypi.org/project/fastmcp/>

```
pip install fastmcp
```

```
pip install mcp
```

Tabla AII-1. Instalación de módulos mcp y fastmcp

Para crear un servidor MCP con fastmcp se utilizó el código Python de la Tabla 6-3.

```
from fastmcp import fastMCP

mcp = fastMCP("MCP Server")

@mcp.tool()
def greet(name: str) -> str:
    return f"Hello, {name}!"

if __name__ == "__main__":
    mcp.run(transport="http", host="127.0.0.1", port=8000) #Http
    #mcp.run(transport="stdio") #Stdio
```

Tabla AII-2. Crear un servidor fastmcp con conexión HTTP

Para la funcionalidad de sistema de archivos, instalamos un servidor MCP de sistema de archivos pre-construido en Claude para escritorio. Abriendo el menú de Claude Desktop en el ordenador se selecciona “Configuración...”. Haciendo clic en “Desarrollador” en la barra izquierda del panel Configuración y a continuación, en “Editar configuración”. El comando “npx” funciona con node.js.

```
{
  "mcpServers": {
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-filesystem",
        "C:\\Users\\<username>\\<path to folder>"
      ]
    }
  }
}
```

Tabla AII-3. Configurar un servidor mcp para un sistema de fichero compartido

```
curl -X POST http://localhost:8000/runs \  
-H "Content-Type: application/json" \  
-d '{  
    "agent_name": "echo",  
    "input": [{"role": "user", "parts": [{"content": "Hello"}]}],  
    "mode": "sync"  
}'
```

Tabla AII-4. Ejecución síncrona de un agente con Curl

En la Tabla AII-5 se muestra el código para desarrollar una aplicación cliente utilizando Streamlit. Dicha aplicación permite seleccionar entre varios servidores de agentes disponibles.

```
import streamlit as st # type: ignore  
import requests # type: ignore  
import asyncio  
from acp_sdk.client import Client # type: ignore  
  
# Inicializa estado  
if "agente" not in st.session_state:  
    st.session_state.agente = []  
  
# Lista de servidores MCP  
mcp_servers = {  
    "deepwiki": "https://mcp.deepwiki.com/mcp",  
    "huggingface": "https://huggingface.co/mcp",  
    "local": "http://localhost:11434"  
}  
  
acp_servers = {  
    "ACP local": "http://localhost:8000",  
    "huggingface": "https://huggingface.co/mcp",  
    "local": "http://localhost:11434"  
}  
  
def sidebar():  
    with st.sidebar:  
        # Add logo in sidebar
```

```
logo = "gui/utils/tars2.png"
st.image(logo, width=90)

#selected_server, server_url = create_mcp_server_dropdown()

# Selector de servidores MCP
MCPoptions = st.multiselect(
    "Servidores MCP",
    list(mcp_servers.keys()),
    #["Servidor 1", "Servidor 2", "Servidor 3"],
)

if MCPoptions != []:
    available_agents=asyncio.run(agentlist(MCPoptions[0]))
    options = st.multiselect(
        "Agentes disponibles",
        available_agents,
        max_selections=5,
        #accept_new_options=True,
    )
    st.session_state.agente = options

# Selector de servidores ACP
ACPOptions = st.multiselect(
    "Servidores ACP",
    list(acp_servers.values()),
    #["ACP local", "huggingface", "local"],
)
#print(options)

if ACPOptions != []:
    available_agents=asyncio.run(agentlist(ACPOptions[0]))
    options = st.multiselect(
        "Agentes disponibles",
        available_agents,
        max_selections=5,
        #accept_new_options=True,
    )
    st.session_state.agente = options

st.write("Seleccionados:", ACPOptions)
```

```
if __name__ == "__main__":
    # Initialize and run the app
    sidebar()

    # Display them in a 2x2 grid using Streamlit columns
    col1, col2 = st.columns([0.2, 0.8])

    with col1:
        st.image("gui/utills/tars2.png", width=90)
    with col2:
        st.title("Agente copiloto")

    if st.session_state.agente!=[]:
        if input := st.chat_input("Busca", key="busca", on_submit=buscador()):
            asyncio.run(run_async(st.session_state.agente[0], input))
            print("buscando")
```

Tabla AII-5. Desarrollo de una App con Streamlit.

En la Tabla AII-6 se muestra el resultado de una solicitud de descubrimiento de agentes en ACP.

```
{ "agents": [
  { "name": "echo", "description": "Echoes everything", "metadata":
  { "annotations": null, "documentation": null, "license": null, "programmi
ng_language": null, "natural_languages": null, "framework": null, "capab
ilities": null, "domains": null, "tags": null, "created_at": null, "update
d_at": null, "author": null, "contributors": null, "links": null, "depende
ncies": null, "recommended_models": null }, "input_content_types": ["*/*
"], "output_content_types": ["*/*"] },
  { "name": "chat_agent", "description": "Habla con el copiloto",
  "metadata":
  { "annotations": null, "documentation": null, "license": null, "programmi
ng_language": null, "natural_languages": null, "framework": null, "capab
ilities": null, "domains": null, "tags": null, "created_at": null, "update
d_at": null, "author": null, "contributors": null, "links": null, "depende
ncies": null, "recommended_models": null }, "input_content_types": ["*/*
"], "output_content_types": ["*/*"] },
  { "name": "translation_agent", "description": "Traductor Inglés-
Español", "metadata":
  { "annotations": null, "documentation": null, "license": null, "programmi
ng_language": null, "natural_languages": null, "framework": null, "capab
ilities": null, "domains": null, "tags": null, "created_at": null, "update
d_at": null, "author": null, "contributors": null, "links": null, "depende
ncies": null, "recommended_models": null }, "input_content_types": ["*/*
"], "output_content_types": ["*/*"] },
```

```

{"name":"buscador_defensa","description":"Buscador en la web de
Defensa",
"metadata":{"annotations":null,"documentation":null,"license":null
,"programming_language":null,"natural_languages":null,"framework":
null,"capabilities":null,"domains":null,"tags":null,"created_at":n
ull,"updated_at":null,"author":null,"contributors":null,"links":nu
ll,"dependencies":null,"recommended_models":null},"input_content_t
ypes":["*/*"],"output_content_types":["*/*"]},

{"name":"buscador_Github","description":"Buscador en los
repositorios de Github",
"metadata":{"annotations":null,"documentation":null,"license":null
,"programming_language":null,"natural_languages":null,"framework":
null,"capabilities":null,"domains":null,"tags":null,"created_at":n
ull,"updated_at":null,"author":null,"contributors":null,"links":nu
ll,"dependencies":null,"recommended_models":null},"input_content_t
ypes":["*/*"],"output_content_types":["*/*"]},

{"name":"buscador_ISFAS","description":"Buscador en la página de
ISFAS","metadata":{"annotations":null,"documentation":null,"licens
e":null,"programming_language":null,"natural_languages":null,"fram
ework":null,"capabilities":null,"domains":null,"tags":null,"create
d_at":null,"updated_at":null,"author":null,"contributors":null,"li
nks":null,"dependencies":null,"recommended_models":null},"input_co
ntent_types":["*/*"],"output_content_types":["*/*"]},

{"name":"buscador_Youtube","description":"Buscador en la página de
YOUTUBE",
"metadata":{"annotations":null,"documentation":null,"license":null
,"programming_language":null,"natural_languages":null,"framework":
null,"capabilities":null,"domains":null,"tags":null,"created_at":n
ull,"updated_at":null,"author":null,"contributors":null,"links":nu
ll,"dependencies":null,"recommended_models":null},"input_content_t
ypes":["*/*"],"output_content_types":["*/*"]},

{"name":"router","description":"","metadata":{"annotations":null,"
documentation":null,"license":null,"programming_language":null,"na
tural_languages":null,"framework":null,"capabilities":null,"domain
s":null,"tags":null,"created_at":null,"updated_at":null,"author":n
ull,"contributors":null,"links":null,"dependencies":null,"recommen
ded_models":null},"input_content_types":["*/*"],"output_content_ty
pes":["*/*"]}]

```

Tabla AII-6. Resultado de la solicitud de listado de agentes mediante la API

Para desplegar una aplicación RAG, creamos un entorno virtual y lo activamos (en Windows):

```
(fastmcp) C:\Users\tucan>python -m venv rag
```

```
(fastmcp) C:\Users\tucan>source venv\Scripts\activate
```

Tabla AII-7. Crear y activar un entorno virtual para la aplicación

Para el desarrollo del RAG es preciso instalar una base de datos vectorial, en nuestro caso ChromaDB, así como las dependencias necesarias (LangChain y otras), según el proceso de la Tabla AII-8:

```
(fastmcp) C:\Users\tucan>pip install --q chromadb
```

```
(fastmcp) C:\Users\tucan>pip install --q unstructured langchain  
langchain-text-splitters
```

```
(fastmcp) C:\Users\tucan>pip install --q "unstructured[all-docs]"
```

Tabla AII-8. Instalar Chroma, LangChain y otras dependencias

En la Tabla AII-9 se muestra el código Python utilizado para crear un agente de búsqueda (solicitud de información) para un RAG:

```
import os  
  
from langchain_ollama import ChatOllama # type: ignore  
from langchain.prompts import ChatPromptTemplate, PromptTemplate #  
type: ignore  
from langchain_core.output_parsers import StrOutputParser # type:  
ignore  
from langchain_core.runnables import RunnablePassthrough # type:  
ignore  
from langchain.retrievers.multi_query import MultiQueryRetriever #  
type: ignore  
from get_vector_db import get_vector_db
```

```
LLM_MODEL = os.getenv('LLM_MODEL', 'dnh2006/salamandra-2b-  
instruct:latest')
```

```
# Function to get the prompt templates for generating alternative  
questions and answering based on context
```

```
def get_prompt():
```

```
    QUERY_PROMPT = PromptTemplate( input_variables=["question"],  
    template="""You are an AI language model assistant. Your task is to  
generate five different versions of the given user question to  
retrieve relevant documents from a vector database. By generating  
multiple perspectives on the user question, your goal is to help the  
user overcome some of the limitations of the distance-based similarity
```

```
search. Provide these alternative questions separated by newlines.
Original question: {question}""", )

    template = """Answer the question based ONLY on the following
context: {context} Question: {question} """
    prompt = ChatPromptTemplate.from_template(template)
    return QUERY_PROMPT, prompt

# Main function to handle the query process
def query(input):
    if input:
        # Inicializar el modelo de LLM con el nombre del modelo
        # específico
        llm = ChatOllama(model=LLM_MODEL)
        # Llamar a la instancia de la base de datos de vectores
        db = get_vector_db()
        # Extraer el template del PROMPT QUERY_PROMPT,
        QUERY_PROMPT, prompt = get_prompt()
        response = retriever(QUERY_PROMPT)

        return response
    return None

# Function to handle retriever
def retriever(query_prompt):
    # Set up the retriever to generate multiple queries using the
    # language model and the query prompt
    retriever = MultiQueryRetriever.from_llm( db.as_retriever(),
    llm, prompt=QUERY_PROMPT )
    # Definir la cadena de procesos para la búsqueda de contexto,
    # generar la respuesta, y parsear la salida
    chain = ( {"context": retriever, "question":
RunnablePassthrough()} | prompt | llm | StrOutputParser() )
    return chain
```

Tabla AII-9. Funciones de pregunta (query) y búsqueda (retriever) para el LLM

```
import os
from datetime import datetime
```

```
from werkzeug.utils import secure_filename

from langchain_community.document_loaders import
UnstructuredPDFLoader

from langchain_text_splitters import
RecursiveCharacterTextSplitter

from get_vector_db import get_vector_db


TEMP_FOLDER = os.getenv('TEMP_FOLDER', './_temp')


# Function to check if the uploaded file is allowed (only PDF files)
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower()
    in {'pdf'}


# Function to save the uploaded file to the temporary folder
def save_file(file):
    # Save the uploaded file with a secure filename and return the
    # file path
    ct = datetime.now()
    ts = ct.timestamp()
    filename = str(ts) + "_" + secure_filename(file.filename)
    file_path = os.path.join(TEMP_FOLDER, filename)
    file.save(file_path)
    return file_path


# Function to load and split the data from the PDF file
def load_and_split_data(file_path):
    # Load the PDF file and split the data into chunks
    loader = UnstructuredPDFLoader(file_path=file_path)
    data = loader.load()

    text_splitter = RecursiveCharacterTextSplitter(chunk_size=7500, chunk_overlap=100)
    chunks = text_splitter.split_documents(data)

    return chunks


# Main function to handle the embedding process
def Embed(file):
```

```
# Check if the file is valid, save it, load and split the data,
add to the database, and remove the temporary file

if file.filename != '' and file and allowed_file(file.filename):
    file_path = save_file(file)
    chunks = load_and_split_data(file_path)
    db = get_vector_db()
    db.add_documents(chunks)
    db.persist()
    os.remove(file_path)

    return True

return False
```

Tabla AII-10. Código del proceso de incrustaciones en una Base de Datos vectorial

```
import os

from langchain_ollama import OllamaEmbeddings # type: ignore
from langchain_chroma import Chroma # type: ignore

CHROMA_PATH = os.getenv('CHROMA_PATH', 'chroma')
COLLECTION_NAME = os.getenv('COLLECTION_NAME', 'localRAG')
TEXT_EMBEDDING_MODEL = os.getenv('TEXT_EMBEDDING_MODEL', 'nomic-
embed-text')

def get_vector_db():
    embedding = OllamaEmbeddings(model=TEXT_EMBEDDING_MODEL)
    db = Chroma(
        collection_name=COLLECTION_NAME,
        persist_directory=CHROMA_PATH, embedding_function=embedding )
    return db
```

Tabla AII-10. Código de la función que lee la BD de vectores (get_vector_db)