UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

PROYECTO FIN DE GRADO

GRADO EN CIENCIA DE DATOS E INTELIGENCIA ARTIFICIAL

Superando la Rigidez de la Búsqueda Geœspacial - Agentes LLM como Interfaz Inteligente para CartoCiudad

Desarrollado por: Pedro Ortiz Villanueva

Dirigido por: Miguel Ángel Manso Callejo y Francisco Serradilla García

Madrid, 28 de junio de 2025

Superando la Rigidez de la Búsqueda Geoespacial - Agentes LLM como Interfaz Inteligente para CartoCiudad
Desarrollado por: Pedro Ortiz Villanueva
Dirigido por: Miguel Ángel Manso Callejo y Francisco Serradilla García
Proyecto Fin de Grado, 28 de junio de 2025

E.T.S. de Ingeniería de Sistemas Informáticos
Campus Sur UPM, Carretera de Valencia (A-3), km. 7
28031, Madrid, España

Si deseas citar este trabajo, la entrada completa en BibTeX es la siguiente:

```
@mastersthesis{citekey,
    title = {Superando la Rigidez de la Búsqueda Geoespacial - Agentes LLM como
Interfaz Inteligente para CartoCiudad},
    type = {Bachelor's Thesis},
    author = {Ortiz Villanueva, P.},
    school = {E.T.S. de Ingeniería de Sistemas Informáticos},
    year = {2025},
    month = {6},
}
```

Agradecimientos

Llegar a este punto no ha sido un camino en solitario. Este trabajo es el resultado de mucho esfuerzo, pero también del apoyo, la guía y la inspiración de muchas personas a las que deseo expresar mi más sincero agradecimiento.

En el ámbito académico, mi más profundo reconocimiento es para mi tutor profesional, Miguel Ángel Manso. Le agradezco no solo su indispensable guía académica, sino, sobre todo, su calidad humana. Gracias por tu cercanía y por haber luchado por este proyecto y por mí, demostrando un compromiso genuino que va mucho más allá de las obligaciones de un tutor. Ver que alguien cree y se esfuerza por ti de esa manera es una de las lecciones más valiosas.

Asimismo, quiero agradecer a los profesores Alberto Díaz y Carlos Badenes. Ellos me enseñaron que la docencia no es un trámite hacia la investigación, sino un objetivo en sí mismo. Ver en ellos esa pasión, ese brillo en los ojos al enseñar, ha moldeado mi visión de lo que significa ser un buen profesional: alguien curioso, con una auténtica vocación por la enseñanza.

En el plano personal es de donde viene mi auténtica inspiración y la motivación para levantarme todas las mañanas; por ello, aquí las gracias se quedan cortas. A mis padres, por vuestro apoyo incondicional, pero, sobre todo, por haberme dado las herramientas para ser quien soy: una persona curiosa, con el ánimo de aprender, explorar y buscarme la vida por mí mismo. Esa capacidad de reflexión y autocrítica que me habéis inculcado es el motor que me impulsa.

A mi amigo Santi. Desde que pisé este campus, te has convertido en un apoyo esencial, el mejor compañero académico y de vida que podría haber imaginado. La complicidad, las risas y la confianza que hemos construido codo con codo cada día ha sido un auténtico privilegio. Gracias por todo.

A mis compañeros de clase, por acogerme con una amabilidad que ni yo mismo creía merecer en ciertos momentos. Vuestro compañerismo ha hecho este camino mucho más ligero. Y a esos amigos que trascienden las aulas, como Mario, Dani y Cano, gracias por ser mi ancla no solo en la universidad, sino en la vida.

Por último, quiero agradecer a aquello que no se puede tocar, pero que ha sido indispensable para llegar hasta aquí. A mi propia capacidad de esfuerzo, pero por encima de todo, a la pasión que siento por mi campo. Adoro este sector porque me permite ser una persona creativa a la par que técnica, una dualidad que me mantiene completamente estimulado e inspirado. Idear y buscar soluciones es lo que me hace sentir vivo y alerta.

Y sí, aunque pueda parecer una excentricidad, quiero agradecer a la música. Ha sido la banda sonora que me ha ayudado a interiorizar, a relativizar y a estar en el camino de aprender a elegir mis batallas. Porque, como he ido descubriendo, no hay nada más importante que saber qué es importante.

Resumen

La interacción con datos geoespaciales oficiales en España se ve frecuentemente obstaculizada por motores de búsqueda rígidos e intolerantes a los errores del usuario. Este trabajo presenta un enfoque basado en agentes inteligentes para transformar esta interacción, abordando las limitaciones del buscador de CartoCiudad (CNIG). Se diseñaron y evaluaron cuatro arquitecturas de agentes de complejidad creciente (Base, Intención, Validación y Ensemble) utilizando LangChain/LangGraph[1], [2], un modelo LLM cuantizado (Qwen3-30b) y una nueva librería (PyCiudad) que interactúa con la API existente. Contrario a la expectativa de que una mayor sofisticación resultaría en un mejor rendimiento, nuestra evaluación sobre un dataset sintético demuestra que la arquitectura más simple (Agente Base) ofrece el balance óptimo entre calidad y eficiencia. La conclusión principal es que las arquitecturas más avanzadas no justificaron su mayor coste en latencia y recursos, estableciendo al Agente Base como la solución óptima en el balance calidad-eficiencia y el punto de partida más sólido para trabajos futuros.

Palabras clave: Agentes LLM; Búsqueda Geoespacial; CartoCiudad; Generación de Datos Sintéticos; LangGraph

Abstract

Interaction with official geospatial data in Spain is often hindered by rigid search engines that are intolerant to user errors. This work presents an intelligent agent-based approach to transform this interaction, addressing the limitations of the CartoCiudad (CNIG) search engine. Four agent architectures of increasing complexity (Base, Intent, Validation, and Ensemble) were designed and evaluated using LangChain/LangGraph[1], [2], a quantized LLM (Qwen3-30b), and a new library (PyCiudad) that interfaces with the existing API. Contrary to the expectation that greater sophistication would result in better performance, our evaluation on a synthetic dataset demonstrates that the simplest architecture (the Base Agent) offers the optimal balance between quality and efficiency. The main conclusion is that the more advanced architectures did not justify their higher cost in terms of latency and resources, establishing the Base Agent as the optimal solution for the quality-efficiency trade-off and the most solid starting point for future work.

Keywords: LLM Agents; Geospatial Search; CartoCiudad; Synthetic Data Generation; LangGraph

Índice general

1 Introducción **1**

1.1 Limitaciones del Buscador de CartoCiudad 1

1.2 Objetivos del Proyecto 5

1.3 Metodología y Estructura 6

2 Estado de la Cuestión **9**

2.1 Búsqueda Léxica: El Sistema Actual 9

2.2 De la Búsqueda Léxica a la Semántica 11

2.3 Limitaciones de los Enfoques Semánticos Puros 16

2.4 Agentes LLM: Un Nuevo Paradigma 17

3 Herramientas y Decisiones Técnicas **21**

3.1 Selección del Framework de Orquestación 21

3.2 Infraestructura y Estrategia de Modelado 25

3.3 La Librería PyCiudad 27

4 Metodología de trabajo: Evolución Arquitectónica de Agentes **29**

4.1 Interacción Geoespacial Avanzada 29

4.2 La Funcionalidad Base 30

4.3 Captura de la Intención 31

4.4 Ciclos de Auto-Corrección 33
4.5 Estrategias Dinámicas 35
4.6 Evolución y Potencial 37

5 Diseño del Dataset de Evaluación 39

5.1 Un Ground Truth Basado en Datos Reales 39
5.2 La Necesidad de la Generación Sintética 40
5.3 Variaciones Lingüísticas y Complejidad Semántica 41
5.4 La Inyección Adversarial de Errores 43
5.5 Estructura Final del Dataset 46

6 Evaluación y Resultados 47

6.1 Metodología de Evaluación y Protocolo Experimental 47
6.2 Resultados Comparativos de las Arquitecturas 49
6.3 Discusión y Trade-offs 54

7 Síntesis y Lecciones Aprendidas 57

7.1 La Eficacia de la Simplicidad 57
7.2 Análisis de las Arquitecturas Complejas 58

8 Consideraciones para la Puesta en Producción 60

8.1 Viabilidad Técnica y Operativa 60
8.2 Viabilidad Económica y Ambiental 62
8.3 Viabilidad Humana y Organizacional 63

9 Trabajo Futuro y Reflexiones sobre su Impacto 64

9.1 Evolución hacia Arquitecturas de Agentes Complejos 64

9.2 Implicaciones Sociales: La Doble Hoja de la Cognición Artificial 65

9.3 Gobernanza y Seguridad como Contrapeso Necesario 66

9.4 Conclusión: Hacia un Nuevo Paradigma de Interacción 66

A Anexo: Documentación de PyCiudad 68

A.1 Introducción y Objetivos de PyCiudad 68

A.2 Arquitectura General de la Librería 69

A.3 Descripción Detallada del Cliente `CartoCiudad` 69

A.4 Modelos de Datos (`modelos.py`) 70

A.5 Utilidades, Constantes y Excepciones 71

A.6 Ejemplo de Uso Completo 71

Índice de figuras

1.1 Logo oficial del Instituto Geográfico Nacional (IGN), organismo de referencia para la información geográfica en España. 1

1.2 Comparativa conceptual entre el enfoque de búsqueda tradicional de CartoCiudad (arriba), basado en reglas y palabras clave estrictas, y la interfaz inteligente propuesta basada en agentes LLM (abajo), que permite una interacción flexible en lenguaje natural. 4

1.3 Diagrama de flujo de la metodología del proyecto. El proceso abarca desde el análisis inicial y el desarrollo de herramientas hasta el diseño iterativo de arquitecturas de agentes y su evaluación final. 8

2.1 Logo de Elasticsearch, una de las tecnologías de búsqueda léxica más extendidas, basada en la creación de índices invertidos para una recuperación rápida de documentos. 10

2.2 Representación conceptual de un espacio vectorial de embeddings. Palabras o frases con significados similares (ej. rey", reina", "príncipe") se agrupan, mientras que las disímiles se sitúan en regiones distantes del espacio. 12

2.3 Visualización de una búsqueda aproximada de vecinos cercanos (ANN). En lugar de una búsqueda exhaustiva, los algoritmos como HNSW navegan por una estructura de grafos jerárquica para encontrar eficientemente los vectores más próximos. 13

2.4 Arquitectura del modelo Transformer, destacando los componentes de Multi-Head Attention y Feed Forward. Adaptado de"Attention Is All You Nee" (Vaswani et al., 2017). 15

2.5 Esquema del ciclo de razonamiento del patrón ReAct. El agente alterna entre fases de razonamiento verbal (*Thought*), ejecución de acciones con herramientas (*Act*) y la asimilación de los resultados (*Observation*). 19

3.1 Logo del framework SmolAgents. 22

3.2 Logo del framework LlamaIndex. 22

3.3 Logo del framework CrewAI. 23

3.4 Logo de LangChain, el framework de orquestación seleccionado para el desarrollo de los agentes de este proyecto. 24

3.5 Diagrama de la arquitectura de infraestructura implementada. Los agentes, ejecutándose localmente, se comunican a través de un túnel seguro de ngrok con un servidor Ollama remoto, que gestiona la inferencia del modelo LLM en una GPU NVIDIA RTX 3090. 25

3.6 Herramientas clave para la infraestructura del servicio. Ollama (izquierda) para servir los modelos de lenguaje y Ngrok (derecha) para crear un túnel de red seguro y público. 26

4.1 Diagrama de flujo del Agente Base. La consulta del usuario pasa por un pipeline secuencial de interpretación, formulación de parámetros, llamada a la API y reordenamiento de resultados. 30

4.2 Arquitectura del Agente de Intención. Introduce un procesamiento en paralelo que analiza la consulta de forma simultánea para extraer entidades e inferir la intención semántica. Ambos resultados se combinan para una búsqueda más precisa. 32

4.3 Flujo de trabajo del Agente de Validación. Implementa un ciclo de autocorrección donde los resultados son evaluados; si se consideran insuficientes, la consulta se reformula y se reintenta la búsqueda, acumulando candidatos en cada iteración. 34

4.4 Lógica del Agente Ensemble. Actúa como un meta-orquestador que primero clasifica la consulta para seleccionar el pipeline más adecuado (Simple, Intermedio o Complejo). Si el pipeline elegido falla, puede escalar automáticamente la tarea al siguiente nivel de complejidad. 36

5.1 Fase inicial del proceso de generación sintética. A partir de una dirección del ground truth, el agente generador es instruido para crear un conjunto de consultas que cubren diferentes estilos lingüísticos (Directa, Natural, Coloquial) y niveles de dificultad (Fácil, Medio, Difícil). 42

5.2 Fase de inyección adversarial de errores. Un subconjunto de las consultas sintéticas generadas se somete a un proceso de degradación controlada, donde el LLM introduce patrones de error comunes en castellano (omisión de tildes, errores de tecleo, etc.) sin alterar la intención de búsqueda subyacente. 44

6.1 Impacto de la dificultad de la consulta en el rendimiento del Agente Base. Se muestra la Puntuación Media de Relevancia (MRS) y la Tasa de Éxito para cada nivel de dificultad. 50

6.2 Análisis de la robustez del Agente Base ante tipos de error específicos. El gráfico muestra cómo varía el MRS en presencia de diferentes degradaciones en la consulta. 50

6.3 Impacto de la dificultad de la consulta en el rendimiento del Agente de Intención. 51

6.4 Análisis de la robustez del Agente de Intención ante tipos de error específicos. 51

6.5 Impacto de la dificultad de la consulta en el rendimiento del Agente de Validación. 52

6.6 Análisis de la robustez del Agente de Validación ante tipos de error específicos. 52

6.7 Impacto de la dificultad de la consulta en el rendimiento del Agente Ensemble. 53

6.8 Análisis de la robustez del Agente Ensemble ante tipos de error específicos. 53

6.9 Análisis comparativo de la robustez de los agentes. El gráfico izquierdo muestra la caída en la Tasa de Éxito cuando se introducen errores. El derecho clasifica los agentes según su "Índice de Robustez"(Tasa de Éxito con errores / Tasa de Éxito sin errores), donde valores más cercanos a 1 indican mayor robustez. 55

6.10 Frontera de Pareto que ilustra el compromiso entre la Calidad de los resultados (MRS) y la Eficiencia (inverso de la latencia). Los puntos en la frontera (rojo) representan las soluciones óptimas. El Agente Base es la única arquitectura que se sitúa en esta frontera. 55

Índice de tablas

3.1 Comparativa de Frameworks de Orquestación de Agentes. 23

5.1 Taxonomía de Consultas para el Dataset de Evaluación. 45

6.1 Tabla Resumen General de Rendimiento por Agente (Dataset Completo - 747 consultas). 49

6.2 MRS por Tipo de Error (Mayor valor = Mejor rendimiento). 54

Índice de listados

A.1 Ejemplo de uso completo de la librería `PyCiudad`. 71

1. Introducción

Figura 1.1. Logo oficial del Instituto Geográfico Nacional (IGN), organismo de referencia para la información geográfica en España.

El Instituto Geográfico Nacional (IGN), cuyo logo se muestra en la Figura 1.1, es el organismo público de referencia en España para la producción, mantenimiento y difusión de la información geográfica oficial del país. Entre sus múltiples responsabilidades se encuentra la creación y gestión de bases de datos geoespaciales fundamentales, así como el desarrollo de servicios y aplicaciones que facilitan el acceso y uso de dicha información por parte de administraciones, empresas y ciudadanos.

Dentro de este marco, el **Centro Nacional de Información Geográfica (CNIG)**, organismo autónomo dependiente del IGN, desarrolla y ofrece el servicio CartoCiudad. Este proyecto proporciona un conjunto integrado de datos geográficos que abarcan todo el territorio nacional, incluyendo información detallada sobre calles, códigos postales, entidades de población y puntos de interés diversos. CartoCiudad se materializa principalmente a través de servicios web, destacando una API (Application Programming Interface) que permite consultar y localizar estos elementos geográficos de forma programática, incorporando un motor de búsqueda específico para estas tareas. Es precisamente este motor de búsqueda y sus capacidades el foco central del presente trabajo.

1.1. Limitaciones del Buscador de CartoCiudad

A pesar de la riqueza de los datos que gestiona CartoCiudad, su motor de búsqueda actual presenta una serie de limitaciones significativas que merman su eficacia y usabilidad, especialmente para usuarios no expertos. Estas deficiencias, originadas en gran

medida por un enfoque técnico tradicional basado en búsquedas léxicas estrictas y reglas predefinidas, constituyen el principal catalizador de este proyecto. A continuación, se detallan los problemas fundamentales identificados:

1.1.1. Rigidez Heurística y Formato

El funcionamiento interno del motor de búsqueda se apoya en gran medida en un conjunto de heurísticas y filtros, implementados sobre una infraestructura como Elasticsearch. Estas heurísticas representan reglas codificadas manualmente por los desarrolladores para intentar interpretar las consultas de los usuarios y mapearlas(Mapear) a los campos de la base de datos. Si bien este enfoque puede ser eficaz para consultas muy específicas y bien estructuradas, sufre de una fragilidad esencial:

- **Rigidez Estructural:** El sistema espera que las consultas se ajusten a patrones predefinidos (por ejemplo, "Tipo Vía + Nombre Vía + Localidad"). Cualquier desviación de este formato esperado, aunque semánticamente correcta para un humano, puede llevar a que la consulta no sea interpretada correctamente o directamente falle.
- **Conocimiento Implícito Requerido:** El usuario necesita, en la práctica, conocer o intuir la estructura subyacente de los datos y cómo debe formular la consulta para que las heurísticas la procesen con éxito. Esto impone una barrera de entrada significativa.
- **Ordenación Fija de Resultados:** La presentación de resultados suele seguir un orden preestablecido (por ejemplo, priorizando calles sobre puntos de interés), lo cual puede ocultar el resultado más relevante para el usuario si este no pertenece a la categoría priorizada inicialmente.

Esta dependencia de reglas fijas hace que el sistema sea poco adaptable a la variabilidad natural del lenguaje y a las diferentes formas en que los usuarios pueden expresar una misma necesidad de búsqueda.

1.1.2. Intolerancia a Errores y Ambigüedad

Una consecuencia directa de la rigidez mencionada es la escasa tolerancia del motor de búsqueda a los errores y ambigüedades comunes en las entradas de los usuarios:

- **Intolerancia a Errores Ortográficos y Tipográficos:** El sistema carece, en general, de mecanismos efectivos para manejar faltas de ortografía, errores de tecleo, uso inconsistente de acentos o abreviaturas no contempladas. Una simple letra incorrecta ("Plasa Mayor" en lugar de "Plaza Mayor") suele ser suficiente para no obtener ningún resultado relevante.
- **Manejo Deficiente de la Ambigüedad:** El lenguaje humano es inherentemente ambiguo. Una misma palabra puede referirse a distintos tipos de lugares ("Colón" puede ser una plaza, una calle, un monumento) o un mismo nombre puede existir en múltiples localidades ("Calle Real"). El motor actual tiene dificultades para desambiguar estas situaciones, ya sea devolviendo resultados irrelevantes, omitiendo el correcto, o requiriendo una especificidad que el usuario no siempre posee de antemano.
- **Dificultad con Consultas Vagas:** Las consultas que no especifican un lugar exacto sino una zona o una característica ("¿Dónde se encuentra el Oso y el Madroño?", "¿Dónde acaba la calle Arenal en Madrid?") quedan fuera del alcance de un sistema basado en coincidencias léxicas estrictas con nombres de entidades geográficas concretas.

Esta falta de robustez genera frustración en el usuario, quien percibe que el sistema "no entiende" peticiones razonables, obligándole a refinar y corregir su consulta múltiples veces, a menudo sin éxito.

1.1.3. Procesamiento del Lenguaje Natural

Quizás la limitación más evidente desde la perspectiva del usuario moderno es la incapacidad del motor de búsqueda para comprender y procesar consultas formuladas en lenguaje natural. Los usuarios están cada vez más acostumbrados a interactuar con sistemas (buscadores web, asistentes virtuales) utilizando frases completas, preguntas o descripciones conversacionales. El motor de CartoCiudad, sin embargo, opera bajo un paradigma de palabras clave:

- **Ausencia de Comprensión Semántica:** El sistema no interpreta el significado o la intención detrás de las palabras del usuario. No puede entender una pregunta como "¿Cuál es la calle principal de Sevilla?" o una instrucción como "Llévame al estadio Santiago Bernabéu".
- **Necesidad de Sintaxis Artificial:** Se obliga al usuario a "traducir" su necesidad a una secuencia de palabras clave que coincida con los índices de la base de datos, adoptando una sintaxis que puede sentirse antinatural y restrictiva.
- **Barrera de Accesibilidad:** Esta limitación no solo afecta a la comodidad, sino también a la accesibilidad. Usuarios con menor competencia digital o que simplemente prefieren una interacción más fluida se ven disuadidos de usar el servicio eficazmente.

En resumen, la necesidad de adherirse a formatos estrictos, la intolerancia a errores y la incapacidad para entender el lenguaje natural convierten al motor de búsqueda actual de CartoCiudad en una herramienta potente pero poco práctica para el público general, justificando plenamente la exploración de nuevas tecnologías como los LLM para superar estas barreras. La Figura 1.2 ilustra el cambio de paradigma que este proyecto busca investigar.

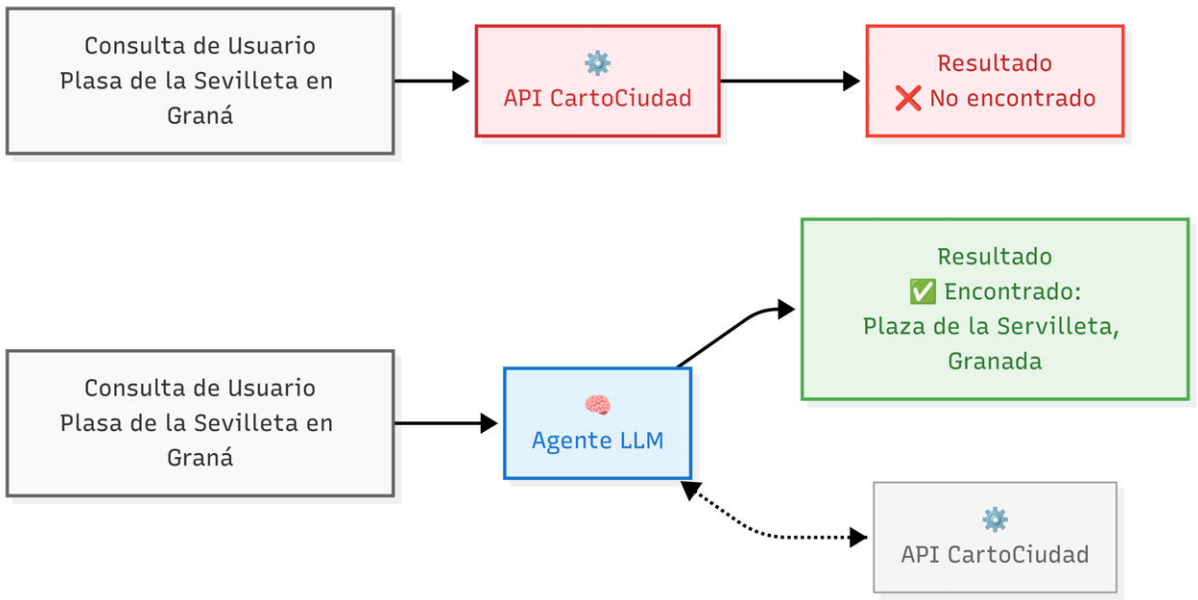

Figura 1.2. Comparativa conceptual entre el enfoque de búsqueda tradicional de CartoCiudad (arriba), basado en reglas y palabras clave estrictas, y la interfaz inteligente propuesta basada en agentes LLM (abajo), que permite una interacción flexible en lenguaje natural.

1.2. Objetivos del Proyecto

Ante las limitaciones identificadas en el motor de búsqueda actual de CartoCiudad, este proyecto se plantea como una exploración enfocada en la aplicación de tecnologías de Inteligencia Artificial de vanguardia, específicamente los LLM y los sistemas de agentes basados en ellos, para abordar dichas deficiencias. Los objetivos se estructuran en un objetivo principal y varios secundarios que lo apoyan:

1.2.1. Objetivo: Robustez mediante Agentes LLM

El propósito central de este trabajo es investigar y demostrar la viabilidad técnica del uso de LLM y arquitecturas de agentes inteligentes como una capa de procesamiento sobre la API existente de CartoCiudad. El fin último es mejorar significativamente la **robustez** del sistema de búsqueda geoespacial. Esta mejora se busca en múltiples dimensiones:

- **Tolerancia a errores:** Capacidad del nuevo enfoque para manejar eficazmente errores ortográficos, tipográficos y variaciones léxicas en las consultas de los usuarios.
- **Manejo de ambigüedad:** Habilidad para interpretar consultas ambiguas, potencialmente interactuando con el usuario para desambiguar o presentando resultados priorizados según la probabilidad semántica.
- **Comprensión del lenguaje natural:** Capacidad para procesar y entender consultas formuladas de manera conversacional, incluyendo preguntas directas o descripciones vagas, extrayendo la intención de búsqueda subyacente.
- **Flexibilidad:** Reducción de la dependencia de formatos de consulta rígidos, permitiendo al usuario expresar su necesidad de forma más libre y natural.

Este objetivo principal no persigue, necesariamente, la creación de un sistema preparado para su despliegue en producción, sino **explorar y evaluar el potencial** de estas tecnologías en el contexto específico de la búsqueda geoespacial del CNIG, identificando tanto sus fortalezas como sus desafíos inherentes.

Para alcanzar el objetivo principal, hemos establecido los siguientes objetivos secundarios que han guiado el desarrollo del trabajo:

- **Evaluar Frameworks de Orquestación de Agentes:** Investigar y comparar diferentes librerías y entornos de desarrollo (frameworks) diseñados para la creación y gestión de agentes basados en LLM (como LangChain/LangGraph[1], [2], CrewAI, etc.), seleccionando el más adecuado con base en criterios de flexibilidad, control, madurez y alineación con los requisitos del proyecto.
- **Desarrollar la Librería PyCiudad:** Crear un componente de software (una librería Python) que abstrae y simplifica la interacción con la API REST de CartoCiudad, facilitando su integración como una herramienta ("tool") dentro de los sistemas de agentes desarrollados.
- **Diseñar e Implementar Arquitecturas de Agentes Prototípicas:** Concebir y construir diferentes modelos de flujo de trabajo (arquitecturas de agentes) utilizando el framework seleccionado. Estas arquitecturas exploran diversas estrategias para procesar las consultas de los usuarios e interactuar con la API de CartoCiudad, variando en complejidad y enfoque (por ejemplo, preprocesamiento, identificación de intención, autorreflexión o metaorquestador).
- **Evaluar Comparativamente los Prototipos:** Definir una metodología de evaluación con métricas claras (precisión, tasa de éxito, latencia, tipo de consulta, etc.) y aplicarla sobre un conjunto de datos de prueba representativo para comparar el rendimiento y la robustez de las diferentes arquitecturas de agentes diseñadas frente a distintos tipos de consultas.
- **Analizar los Trade-offs:** Identificar y discutir las ventajas y desventajas inherentes a cada arquitectura propuesta y al uso de LLM en general para esta tarea. Esto incluye el análisis de la relación entre complejidad computacional, latencia, coste de inferencia, robustez, flexibilidad y controlabilidad de las soluciones.

La consecución de estos objetivos secundarios permitirá obtener una comprensión profunda de las posibilidades y limitaciones de aplicar LLM y agentes a la mejora del buscador de CartoCiudad, proporcionando resultados concretos y conclusiones fundamentadas.

1.3. Metodología y Estructura

El enfoque metodológico de este proyecto ha sido eminentemente práctico y orientado a la ingeniería de soluciones. Tras identificar las carencias funcionales del buscador de

CartoCiudad, se procedió a la evaluación de tecnologías (LLM, agentes) y herramientas (frameworks) con el objetivo de construir prototipos funcionales que abordaran dichas carencias. Esto requirió no solo la selección de un entorno de desarrollo (LangChain/-LangGraph[1], [2]), sino también la resolución de desafíos de infraestructura (acceso a GPU remota) y el desarrollo de un componente de software clave (la librería PyCiudad) para interactuar con la API existente. El flujo de trabajo seguido, esquematizado en la Figura 1.3, consistió en el diseño, implementación y prueba de diferentes arquitecturas de agentes que utilizan PyCiudad para mejorar la robustez de la búsqueda.

La estructura del documento sigue esta lógica de construcción y evaluación: el **Capítulo 2** proporciona el marco conceptual necesario; el **Capítulo 3** describe el *andamiaje* técnico (las herramientas seleccionadas, la infraestructura habilitada y la librería PyCiudad desarrollada); el **Capítulo 4** presenta las *construcciones* (las distintas arquitecturas de agentes implementadas); el **Capítulo 5** detalla la metodología de generación de datasets sintéticos para la evaluación rigurosa de los agentes; el **Capítulo 6** evalúa el *rendimiento* de estas construcciones mediante pruebas específicas y, finalmente, los **Capítulos 7, 8 y 9** ofrecen las conclusiones sobre la viabilidad de estas soluciones y discuten aspectos críticos para una posible implementación futura.

El código fuente de los componentes desarrollados en este trabajo se encuentra disponible públicamente en repositorios de GitHub para fomentar la transparencia y la reproducibilidad de esta investigación:

- **Librería `PyCiudad`:** Repositorio que contiene el wrapper de Python para la API de CartoCiudad.

  `github.com/PedroOrtix/PyCiudad`

- **Agentes y Evaluación:** Repositorio que contiene el código fuente para la implementación de las arquitecturas de agentes, la generación del dataset sintético y los scripts de evaluación con sus resultados y gráficos.

  `github.com/PedroOrtix/Pyciudad_Agents`

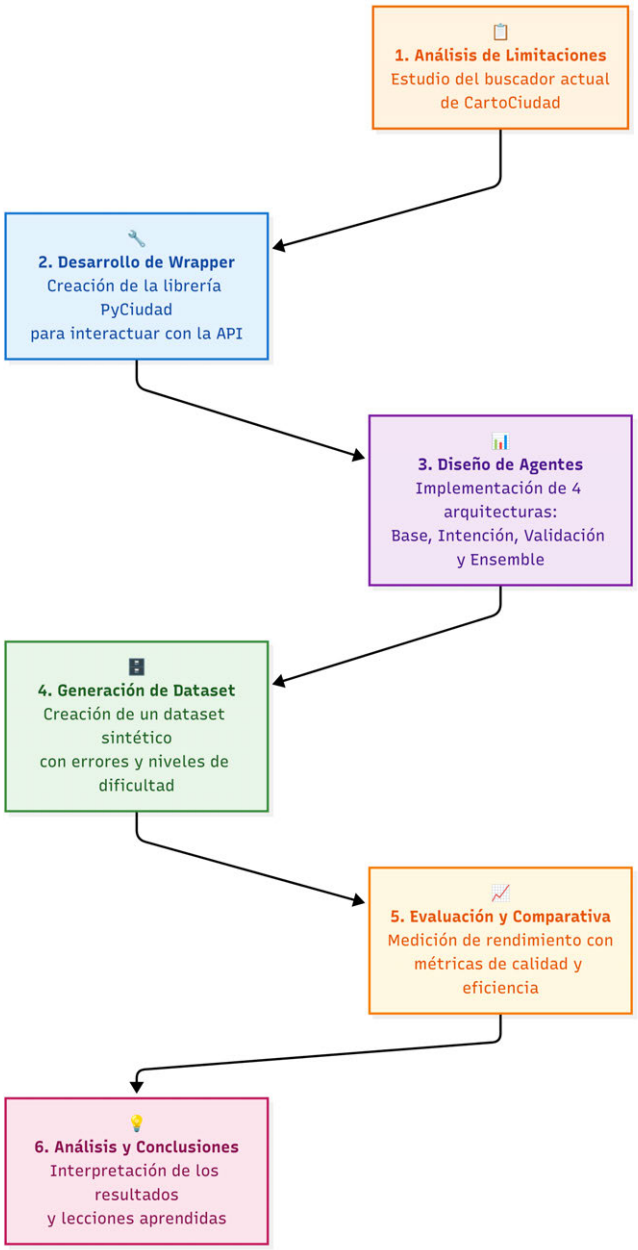

Figura 1.3. Diagrama de flujo de la metodología del proyecto. El proceso abarca desde el análisis inicial y el desarrollo de herramientas hasta el diseño iterativo de arquitecturas de agentes y su evaluación final.

2. Estado de la Cuestión

La búsqueda de información geoespacial, si bien fundamental para innumerables aplicaciones, presenta desafíos únicos cuando se busca robustez y flexibilidad frente a la variabilidad con la que los usuarios expresan sus necesidades. El sistema de búsqueda actual de CartoCiudad, a pesar de su valor como repositorio oficial de datos geográficos españoles, sufre limitaciones significativas en este aspecto, tal como se detalló en la introducción. Estas deficiencias motivan la exploración de paradigmas tecnológicos más avanzados que puedan complementar o superar las aproximaciones tradicionales.

Este capítulo se adentra en el estado de la cuestión relevante, trazando una evolución desde los fundamentos de la búsqueda léxica (la base del sistema actual) hasta las capacidades emergentes de los LLM y los sistemas de agentes. Se analizarán los principios, ventajas y limitaciones de cada etapa tecnológica, con el objetivo de contextualizar y justificar la metodología y las soluciones propuestas en este proyecto, que buscan precisamente aplicar estos avances para potenciar el buscador de CartoCiudad.

2.1. Búsqueda Léxica: El Sistema Actual

Para comprender las oportunidades de mejora en el buscador de CartoCiudad, es esencial primero analizar los fundamentos tecnológicos sobre los que opera su sistema actual y las limitaciones inherentes a dicho enfoque. La búsqueda de información tradicional, especialmente para grandes volúmenes de datos textuales o semiestructurados, se ha apoyado históricamente en la **búsqueda léxica**, también conocida como búsqueda por coincidencia exacta de términos.

La búsqueda léxica opera bajo el modelo de *bag of words* [3], donde tanto los documentos (en este caso, las entradas geográficas de CartoCiudad) como las consultas de los usuarios se tratan como conjuntos de términos independientes, sin considerar en primera instancia el orden o la sintaxis compleja. La eficiencia de este método radica en el uso de **índices invertidos**, estructuras de datos que mapean(Mapear) cada término a una lista de documentos que lo contienen. Al recibir una consulta, el sistema busca

los términos en este índice y devuelve los documentos coincidentes. Tecnologías como **Elasticsearch** (Figura 2.1) son implementaciones robustas de este paradigma, capaces de procesar consultas en milisegundos sobre índices masivos. Estos motores suelen emplear algoritmos de ponderación como **TF-IDF (Term Frequency-Inverse Document Frequency)**[4] o su extensión más refinada, **BM25S (Best Match 25)**[5]. TF-IDF asigna a cada término un peso proporcional a su frecuencia en un documento e inversamente proporcional a su frecuencia en toda la colección, mientras que BM25 introduce mejoras como la normalización por longitud del documento y la saturación de la frecuencia del término, ofreciendo a menudo resultados más precisos. En esencia, estos algoritmos permiten ordenar los resultados devueltos por una relevancia calculada en base a la presencia y distribución de los términos de la consulta.

Figura 2.1. Logo de Elasticsearch, una de las tecnologías de búsqueda léxica más extendidas, basada en la creación de índices invertidos para una recuperación rápida de documentos.

En el contexto específico de CartoCiudad, que maneja datos de direcciones inherentemente irregulares (con abreviaturas, variaciones idiomáticas, errores comunes de usuario, etc.), su motor de búsqueda léxico se complementa con una capa significativa de **heurísticas de normalización y parseo de la consulta**. Este preprocesamiento es crucial: se convierten abreviaturas estándar ("C/" a "Calle"), se manejan homónimos ("Sta." a "Santa") y se aplica tokenización específica para extraer componentes estructurales de la dirección como el tipo de vía, nombre, número y localidad. Este proceso busca "canonizar" la consulta para facilitar la coincidencia con las entradas indexadas. CartoCiudad, de hecho, ofrece opciones de autocompletado y aproximaciones léxicas (*fuzzy matching*)[6] que intentan paliar estas irregularidades.

Sin embargo, y aquí radican las limitaciones fundamentales que este proyecto busca abordar, estas **heurísticas, por extensas que sean, siguen siendo frágiles** ante la riqueza y variabilidad del lenguaje natural. La búsqueda léxica, incluso con preprocesamiento, adolece de:

- **Incapacidad para capturar semántica o contexto:** No entiende que "ordenador" y "computadora" son sinónimos si no existe una regla explícita. Una consulta por "Avenida principal cerca de la catedral" no se mapea(Mapear) fácilmente a una búsqueda por términos exactos.

- **Sensibilidad a la variabilidad lingüística y errores no previstos:** La prevalencia de sinónimos no cubiertos por las heurísticas disminuye el *recall* (exhaustividad) de los resultados[7]. Faltas de ortografía o formatos no contemplados en las reglas pueden llevar a que no se encuentren coincidencias exactas.
- **Problemas con la ambigüedad (polisemia):** Un término como "Colón" puede referirse a múltiples tipos de entidades geográficas o lugares en diferentes ciudades, y un sistema léxico puro tiene dificultades para desambiguar sin información contextual adicional, que no suele procesar.
- **Ignorancia del orden y relaciones sintácticas complejas:** Aunque menos crítico para direcciones simples, la interpretación de consultas más conversacionales se ve impedida.

En definitiva, si bien la búsqueda léxica es eficiente y efectiva para consultas precisas y bien formadas, su dependencia de coincidencias exactas de términos y reglas predefinidas la hace inherentemente limitada en robustez y flexibilidad. Estas carencias justifican la exploración de tecnologías capaces de comprender la intención del usuario y el significado subyacente del lenguaje, más allá de la simple comparación de cadenas de caracteres.

2.2. De la Búsqueda Léxica a la Semántica

Las limitaciones inherentes a la búsqueda léxica, particularmente su incapacidad para comprender el significado más allá de la coincidencia literal de términos, impulsaron la investigación hacia enfoques que pudieran capturar la **semántica** del lenguaje. Esta evolución ha culminado recientemente en el desarrollo y la creciente aplicabilidad de LLM, tecnologías que están redefiniendo las capacidades de los sistemas para procesar y entender el lenguaje natural.

2.2.1. Embeddings y Búsqueda Vectorial

El avance fundamental que posibilitó la búsqueda semántica fue el desarrollo de los **text embeddings**[8]. Un embedding es una representación vectorial densa (es decir, un vector de N dimensiones, donde cada número tiene un significado implícito) de una

unidad de texto (palabra, frase o documento) que busca capturar su significado. A diferencia de las representaciones dispersas (*sparse*) como el *one-hot encoding*[9], en un espacio de embeddings, elementos con significados similares se sitúan geométricamente cerca unos de otros. Esta propiedad permite que operaciones matemáticas, por ejemplo el cálculo de la similitud del coseno entre otras metricas, reflejen relaciones semánticas útiles.

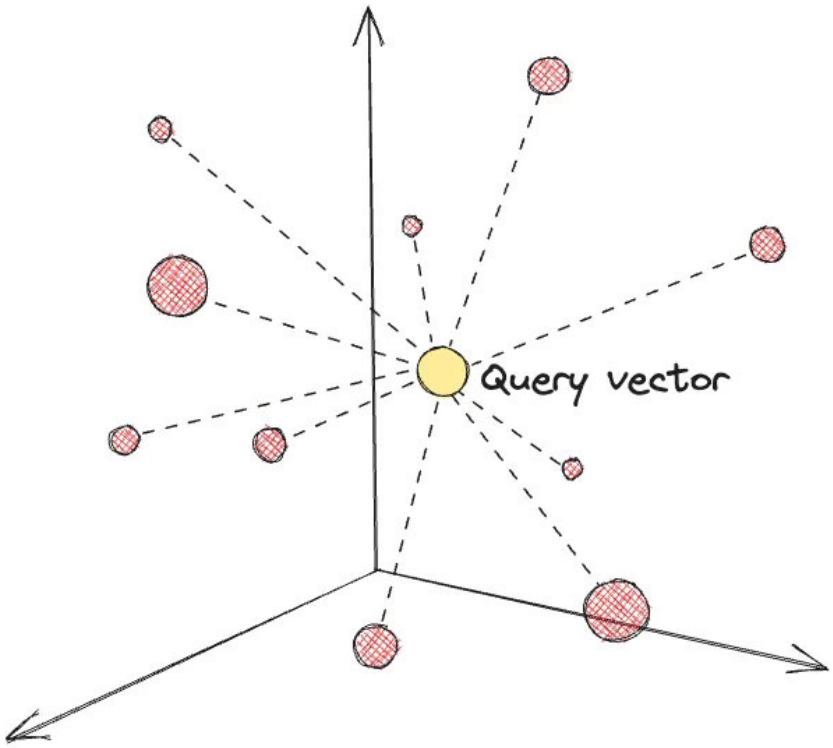

Figura 2.2. Representación conceptual de un espacio vectorial de embeddings. Palabras o frases con significados similares (ej. rey", reina", "príncipe") se agrupan, mientras que las disímiles se sitúan en regiones distantes del espacio.

La generación de embeddings ha evolucionado significativamente. Los modelos iniciales, como Word2Vec[10] y GloVe[11], aprendían representaciones estáticas para palabras basándose en sus contextos de coocurrencia. Si bien supusieron un gran avance al capturar analogías léxicas (por ejemplo, "Rey - Hombre + Mujer = Reina"), tenían la limitación de asignar un único vector a cada palabra, sin poder distinguir la polisemia. El verdadero cambio llegó con los **embeddings contextuales**, popularizados por modelos como ELMo[12] y, de forma más prominente, por aquellos basados en la arquitectura Transformer[13], como BERT[14]. Estos modelos generan vectores diferentes para una misma palabra dependiendo de la oración en la que aparece, capturando así matices de significado mucho más valiosos.

El componente encargado de transformar el texto de entrada en estos embeddings es el **Encoder**. En la búsqueda semántica moderna, los encoders suelen ser redes neuronales basadas en Transformers[13], preentrenadas en grandes corpus textuales y a menudo afinadas (*fine-tuned*) para tareas de similitud de oraciones, como los modelos de la librería `SentenceTransformers`. Estos encoders, como `all-mpnet-base-v2`, pueden tomar una frase o documento y producir un embedding que encapsula su significado semántico.

Una vez que el texto se convierte en embeddings, la búsqueda semántica se realiza mediante la **búsqueda por similitud vectorial**. Dada una consulta convertida a vector, se buscan en una colección de vectores de documentos aquellos que son más "cercanos" según una métrica de distancia o similitud. Para manejar eficientemente millones de estos vectores de alta dimensión, han surgido las **Bases de Datos Vectoriales (VDBs)**[15], como Chroma, Weaviate o Pinecone. Estos sistemas están especializados en almacenar, indexar y buscar vectores mediante algoritmos de **Búsqueda Aproximada de Vecinos Cercanos (ANN)**[16], permitiendo responder a consultas semánticas en milisegundos.

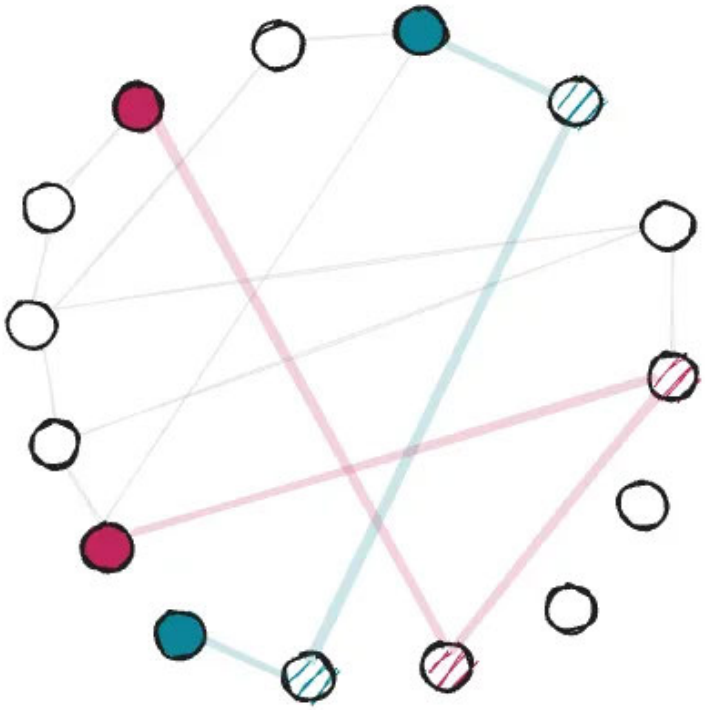

Figura 2.3. Visualización de una búsqueda aproximada de vecinos cercanos (ANN). En lugar de una búsqueda exhaustiva, los algoritmos como HNSW navegan por una estructura de grafos jerárquica para encontrar eficientemente los vectores más próximos.

2.2.2. La Arquitectura Transformer y los LLM

La arquitectura que subyace a la mayoría de los encoders modernos y, de forma más general, a la revolución actual en el Procesamiento del Lenguaje Natural es el **Transformer**[13]. La innovación clave del Transformer [13] es su mecanismo de **autoatención (*self-attention*)**, detallado en la arquitectura que se muestra en la Figura 2.4, que permite al modelo ponderar la importancia de todas las palabras en una secuencia al procesar cada palabra individual. Esto, a diferencia de las arquitecturas recurrentes (RNN) que procesan el texto secuencialmente, permite al Transformer[13] capturar dependencias de largo alcance de manera eficiente y procesar toda la secuencia en paralelo, facilitando el entrenamiento en GPU.

Sobre esta arquitectura se han construido los LLM. Un LLM es una red neuronal (generalmente de tipo Transformer-Decoder-Only) con un gran número de parámetros (desde cientos de millones hasta billones) entrenada con volúmenes masivos de datos textuales. Ejemplos notables incluyen la serie GPT (OpenAI) [17], Gemini (Google) [18], Qwen (Alibaba) [19] o Claude (Anthropic) [20]. Este preentrenamiento masivo, típicamente autosupervisado (por ejemplo, prediciendo la siguiente palabra o palabras enmascaradas), permite a los LLM aprender patrones lingüísticos complejos, conocimiento factual y de "sentido común".

El auge y la viabilidad de los LLM actuales se deben a una confluencia de factores:

- **Escala sin precedentes:** Tanto en el número de parámetros del modelo como en la cantidad de datos de entrenamiento. Las "leyes de escalado" demostraron que un mayor tamaño a menudo se traduce en un mejor rendimiento predecible en diversas tareas.
- **Capacidades Emergentes (*Emergent Abilities*):** Habilidades que no están presentes en modelos más pequeños pero que surgen abruptamente cuando los LLM alcanzan cierta escala[21]. Ejemplos de ello son el razonamiento, la comprensión de instrucciones complejas con pocos ejemplos (*few-shot learning*)[22] o la generación de código.
- **Comprensión Implícita del Mundo (*Internal World Model*):** Aunque no interactúan directamente con el mundo físico, los LLM, al procesar ingentes cantidades de texto que describen el mundo, desarrollan un "modelo interno" implícito de cómo funcionan las cosas, las relaciones entre conceptos y diferentes dinámicas del mundo real[23].

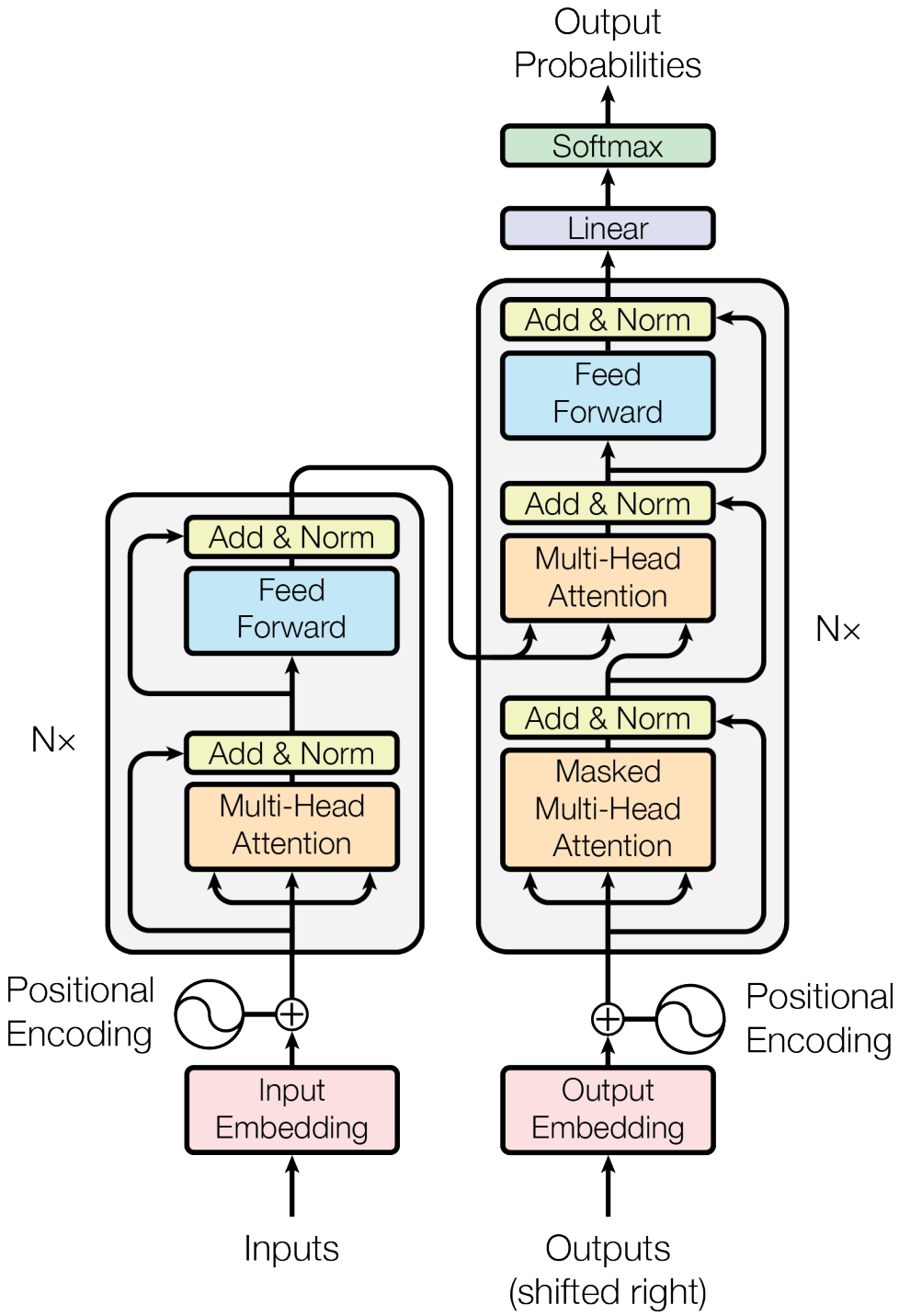

Figura 2.4. Arquitectura del modelo Transformer, destacando los componentes de Multi-Head Attention y Feed Forward. Adaptado de"Attention Is All You Nee" (Vaswani et al., 2017).

- **Computación en Tiempo de Inferencia (*Test-Time Compute*)[24] / Razonamiento:** Los LLM no solo recuperan información, sino que pueden realizar cómputos y razonamientos complejos durante la generación de la respuesta; por ejemplo, descomponiendo un problema en pasos (como en el *Chain-of-Thought prompting*)[25] o incluso estando entrenados para ello[26].
- **Aumento de las Ventanas de Contexto:** Los avances en los *positional embeddings* (por ejemplo, RoPE - *Rotational Positional Embeddings*)[27] y en la eficiencia de los mecanismos de atención (por ejemplo, MQA - *Multi-Query Attention*[28] o GQA - *Grouped-Query Attention*[29]) han permitido a los LLM procesar y "recordar" contextos mucho más largos.
- **Ecosistema de Software y Comunidad:** La disponibilidad de modelos y el desarrollo de librerías y herramientas (como `Hugging Face Transformers`[30], `LangChain`[1], `Ollama`[31] o `vLLM`[32]) han democratizado el acceso y facilitado enormemente la experimentación y la construcción de aplicaciones sobre LLM.

Estos avances han transformado a los LLM de simples completadores de texto a potentes motores de comprensión y generación de lenguaje, capaces de realizar tareas que antes se consideraban exclusivas de la inteligencia humana.

2.3. Limitaciones de los Enfoques Semánticos Puros

La capacidad de los embeddings para capturar el significado y la eficiencia de las Bases de Datos Vectoriales[15] (VDB) para realizar búsquedas semánticas a gran escala representan un avance significativo sobre la búsqueda léxica tradicional. De hecho, una primera aproximación para mejorar un sistema como el de CartoCiudad podría ser indexar todas sus entradas geográficas como embeddings en una VDB[15] y realizar búsquedas de similitud semántica. Este enfoque, a menudo combinado con un LLM en un paradigma conocido como **Generación Aumentada por Recuperación (RAG)**, ha demostrado ser muy efectivo para muchas tareas. En un sistema RAG, se recuperan fragmentos de información relevantes de una base de conocimiento y se proporcionan como contexto a un LLM para que genere una respuesta más precisa.

Este paradigma RAG ofrece ventajas claras, como la capacidad de trabajar con información actualizada y la reducción de "alucinaciones" del LLM al fundamentar sus respuestas en datos concretos. Sin embargo, al considerar su aplicación directa al dominio específico de las direcciones y lugares geográficos de CartoCiudad, surgen limitaciones importantes derivadas de la naturaleza semiestructurada de estos datos. Las direcciones postales poseen una estructura jerárquica inherente (tipo de vía, nombre, número, etc.) que es crucial para su correcta interpretación. De hecho, la Unión Europea ha invertido un esfuerzo considerable en impulsar programas para la estandarización de esta clase de datos, como el programa **INSPIRE**.

El principal desafío radica en que, al convertir una dirección completa en un único embedding vectorial, se tiende a "aplanar" o perder esta valiosa información estructural. Como se ha analizado en la literatura sobre el tema, aunque los embeddings capturan bien la semántica general, los detalles finos pueden diluirse[33]. Por ejemplo, "Calle Mayor, 5, Madrid" y "Calle Mayor, 25, Madrid" son semánticamente muy similares y sus embeddings estarían muy próximos en el espacio vectorial. Un sistema RAG que recupere la dirección más "semánticamente similar" podría fácilmente confundir el número de portal si la consulta del usuario no es perfectamente explícita.

2.4. Agentes LLM: Un Nuevo Paradigma

Frente a las limitaciones de la búsqueda léxica y los desafíos de los enfoques semánticos puros para datos estructurados, emerge una tercera vía: los **sistemas de agentes inteligentes impulsados por LLM**. Este paradigma busca combinar la comprensión del lenguaje y las capacidades de razonamiento de los LLM con la habilidad de interactuar con entornos y herramientas externas.

2.4.1. El Concepto de Agente en Inteligencia Artificial

En el campo de la Inteligencia Artificial, un **agente** se define como cualquier entidad capaz de percibir su entorno, razonar sobre esa percepción y actuar sobre dicho entorno para alcanzar objetivos específicos. Un agente no es un programa pasivo, sino que opera en un ciclo continuo de percepción-razonamiento-acción. Para ser considerado inteligente o racional, un agente debe seleccionar sus acciones de forma que maximice alguna medida de desempeño.

En el contexto de los sistemas modernos, un LLM puede funcionar como el **componente central de razonamiento (el "cerebro")** de un agente. El LLM procesa la información percibida, delibera sobre los posibles pasos a seguir y decide la próxima acción a ejecutar.

2.4.2. Alineamiento y Fiabilidad de los Agentes

Un LLM preentrenado, a pesar de sus notables capacidades, no garantiza por sí solo un comportamiento seguro o alineado con las intenciones del usuario. De hecho, un **LLM base** simplemente predice la siguiente palabra y puede generar contenido irrelevante.

Para que un LLM pueda actuar eficazmente como el cerebro de un agente, es crucial el proceso de **alineamiento**. Alinear un LLM significa orientar su comportamiento mediante técnicas de afinado (*fine-tuning*) para que sus respuestas sean útiles, veraces y seguras. Las técnicas más empleadas son:

- **Afinado Supervisado (SFT) con instrucciones:** Consiste en afinar el modelo preentrenado utilizando ejemplos de instrucciones humanas emparejadas con respuestas deseadas[34].
- **Aprendizaje por Refuerzo con Retroalimentación Humana (RLHF):** Este método utiliza evaluaciones humanas de las respuestas del modelo como señal de recompensa para afinar el LLM principal[35].

Más recientemente, han surgido enfoques como la **Optimización Directa de Preferencias (DPO)**[36], que buscan simplificar el proceso de RLHF. El resultado de estas técnicas es un LLM que se comporta de forma "ad hoc" según las instrucciones dadas en el prompt. Esta capacidad de **respuesta generalizada a prompts (*promptability*)** es precisamente lo que habilita a los LLM alineados a actuar como agentes inteligentes versátiles.

2.4.3. Capacidades de los Agentes LLM

Los agentes modernos basados en LLM se integran en arquitecturas que les otorgan capacidades adicionales:

- **Planificación:** La habilidad de descomponer objetivos complejos en subtareas manejables y trazar un plan de acciones.
- **Memoria:** Mecanismos para almacenar y recuperar información más allá de la ventana de contexto, incluyendo memoria a corto y largo plazo[37], [38].
- **Uso de Herramientas (*Tool Use*):** Facultad para interactuar con programas o servicios externos (APIs, bases de datos, etc.) para obtener información o realizar acciones. En este proyecto, la librería `PyCiudad` se concibe como una herramienta que el agente LLM utilizará para interactuar con la API de CartoCiudad.
- **Patrones de Razonamiento:** Para guiar la toma de decisiones, los agentes emplean patrones específicos. Dos de los más fundamentales son:
  - **Chain-of-Thought (CoT):** Induce al LLM a generar pasos de razonamiento intermedios antes de la respuesta final, obligándole a "pensar en voz alta"[25].
  - **ReAct (Reason → Act → Observe):** Este framework es crucial para agentes que interactúan con herramientas. Permite al LLM intercalar explícitamente trazas de **razonamiento** verbal (*Thought*) con la ejecución de **acciones** (*Act*) y las **observaciones** (*Observation*) resultantes en un bucle iterativo[39].

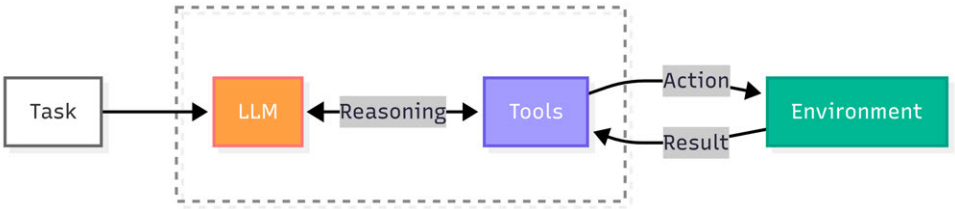

Figura 2.5. Esquema del ciclo de razonamiento del patrón ReAct. El agente alterna entre fases de razonamiento verbal (*Thought*), ejecución de acciones con herramientas (*Act*) y la asimilación de los resultados (*Observation*).

Además de estos, existen otras meta-arquitecturas de razonamiento como **Tree of Thoughts (ToT)**[40] o **Least-to-Most Prompting (L2M)**[41].

2.4.4. Orquestación de Agentes

La construcción de estos agentes se ve facilitada por frameworks de orquestación. Librerías como `LangChain`[1] o `LlamaIndex`[42] proporcionan abstracciones y módulos pre-construidos para definir agentes, integrar herramientas y gestionar los flujos de

interacción. Estos frameworks permiten a los desarrolladores ensamblar agentes complejos con relativa facilidad.

Para concluir, la evolución desde la búsqueda léxica hasta los sofisticados agentes LLM actuales representa un cambio paradigmático. Este enfoque parece particularmente prometedor para superar las limitaciones del buscador actual de CartoCiudad, no intentando reemplazar su base de datos, sino interactuando con ella de una forma mucho más robusta y semánticamente consciente a través de su API. Las decisiones técnicas específicas sobre cómo implementar esta visión se detallarán en los capítulos subsiguientes.

3. Herramientas y Decisiones Técnicas

Tras establecer en el capítulo anterior el potencial de los agentes basados en LLM para abordar las limitaciones del buscador de CartoCiudad, este capítulo se adentra en las decisiones técnicas concretas y las herramientas seleccionadas o desarrolladas para materializar una solución. Un componente crucial en cualquier sistema de agentes LLM es el **framework de orquestación**, la librería o entorno de desarrollo que facilita la definición, gestión y ejecución de los agentes y sus interacciones. La elección de este framework impacta directamente en la flexibilidad, el control, la escalabilidad y la facilidad de desarrollo del proyecto.

3.1. Selección del Framework de Orquestación

Dada la rápida evolución del ecosistema de IA generativa, existen múltiples frameworks para la creación de agentes. Para seleccionar el más adecuado, hemos realizado una exploración preliminar de varias opciones, evaluando no solo sus características técnicas sino también su alineación con los objetivos de aprendizaje y los requisitos específicos del proyecto, como la necesidad de control, la transparencia en el funcionamiento y la capacidad de integración con la API de CartoCiudad.

3.1.1. Evaluacion de Alternativas

La exploración inicial ha considerado varios frameworks populares, basándose tanto en la documentación oficial como en la experiencia práctica.

- **`SmolAgents`[43]:** Este framework, impulsado por Hugging Face, destaca por su

enfoque minimalista y *code-first*. Los agentes escriben acciones directamente en Python, lo que prometía simplicidad y una baja curva de aprendizaje. Su compatibilidad agnóstica con cualquier LLM y la modularidad para usar herramientas externas eran también atractivas. A pesar de su simplicidad, su enfoque en la ejecución de código y las abstracciones reducidas han resultado en una "caja negra" en ciertos aspectos, una desventaja para un proyecto exploratorio donde el objetivo es entender y controlar el flujo de razonamiento.

Figura 3.1. Logo del framework SmolAgents.

- **`LlamaIndex`[42]:** Está fuertemente centrado en la orquestación de datos y la construcción de aplicaciones RAG avanzadas. Sus capacidades para conectar LLM con datos personalizados son robustas. Sin embargo, su fuerte orientación a RAG no se alineaba directamente con el núcleo del problema de CartoCiudad, que requiere más la *interpretación* de la consulta y la *interacción estructurada* con una API existente que la recuperación de grandes volúmenes de documentos.

Figura 3.2. Logo del framework LlamaIndex.

- **`CrewAI`[44]:** Se presenta como un framework orientado a equipos colaborativos de agentes autónomos, con abstracciones claras como `Crews` (equipos) y `Flows` (flujos). Tras una experimentación más profunda, la **latencia** se convirtió en un problema. Hemos observado demoras notables entre la finalización de una tarea y el inicio de la siguiente, atribuibles a las comprobaciones internas del framework. Dado que un motor de búsqueda de direcciones requiere la menor latencia posible, este factor fue decisivo.

Figura 3.3. Logo del framework CrewAI.

Tabla 3.1. Comparativa de Frameworks de Orquestación de Agentes.

Framework	**Ventajas Principales**	**Inconvenientes**
`SmolAgents`	• Minimalista y *code-first* • Compatible con cualquier LLM • Baja curva de aprendizaje	• Caja negra en el razonamiento • Memoria efímera por defecto • Sin RAG nativo
`LlamaIndex`	• Excelente para RAG • Amplio ecosistema de integraciones • Soporte para múltiples fuentes de datos	• Orientado a recuperación de documentos • Curva de aprendizaje más elevada • Overhead innecesario para este caso de uso
`CrewAI`	• Enfoque en colaboración de agentes • Roles y flujos de trabajo claros • Amigable para *prompt-engineering*	• Alta latencia observada • Muy dependiente de la calidad de los prompts • Telemetría activa por defecto
`LangChain/Graph`	• Control granular y total del flujo • Transparencia y depurabilidad • Mínima latencia añadida por el framework	• Mayor complejidad inicial • Curva de aprendizaje para el paradigma de grafos • Más código explícito requerido

3.1.2. Adopción de LangChain y LangGraph

Considerando las limitaciones de las alternativas y los objetivos del proyecto, la elección ha recaído en **LangChain[1]**, complementado con su extensión más reciente para la orquestación de agentes, **LangGraph[2]**.

La introducción de LangGraph[2] ha representado un avance significativo y ha sido el factor determinante. Permite definir flujos de múltiples agentes como un **grafo de estados (*state graph*)**, donde los nodos representan agentes o funciones y las aristas definen las transiciones. Esta aproximación ha ofrecido varias ventajas clave:

- **Control y Flexibilidad:** Proporciona un control granular sobre la lógica del flujo, permitiendo definir explícitamente cada paso y condición de transición.
- **Paradigma de Grafos Intuitivo:** La representación de la lógica como un grafo ha resultado particularmente intuitiva para razonar sobre flujos complejos.
- **Transparencia y Depurabilidad:** Con el apoyo de herramientas como LangSmith[45], es posible trazar la ejecución del agente, inspeccionar el estado en cada paso y depurar problemas de manera efectiva.
- **Ecosistema Robusto de LangChain:** LangGraph[2] se beneficia de todo el ecosistema de LangChain[1], incluyendo su amplia variedad de integraciones.

Figura 3.4. Logo de LangChain, el framework de orquestación seleccionado para el desarrollo de los agentes de este proyecto.

En resumen, LangChain[1] con LangGraph[2] ofrece el mejor equilibrio entre madurez, flexibilidad, control y un rico ecosistema, considerándose la opción más idónea para explorar las arquitecturas de agentes propuestas.

3.2. Infræstructura y Estrategia de Modelado

La implementación y experimentación con modelos de lenguaje han estado intrínsecamente ligadas a las capacidades de la infraestructura hardware disponible.

3.2.1. Acceso Remoto a una GPU

Para la ejecución local de los LLM, se ha dispuesto de acceso a una máquina remota equipada con una tarjeta gráfica NVIDIA GeForce RTX 3090. Esta GPU cuenta con **24 GB de memoria de vídeo (VRAM)**, un factor crítico que determina el tamaño máximo de los modelos que pueden ser cargados.

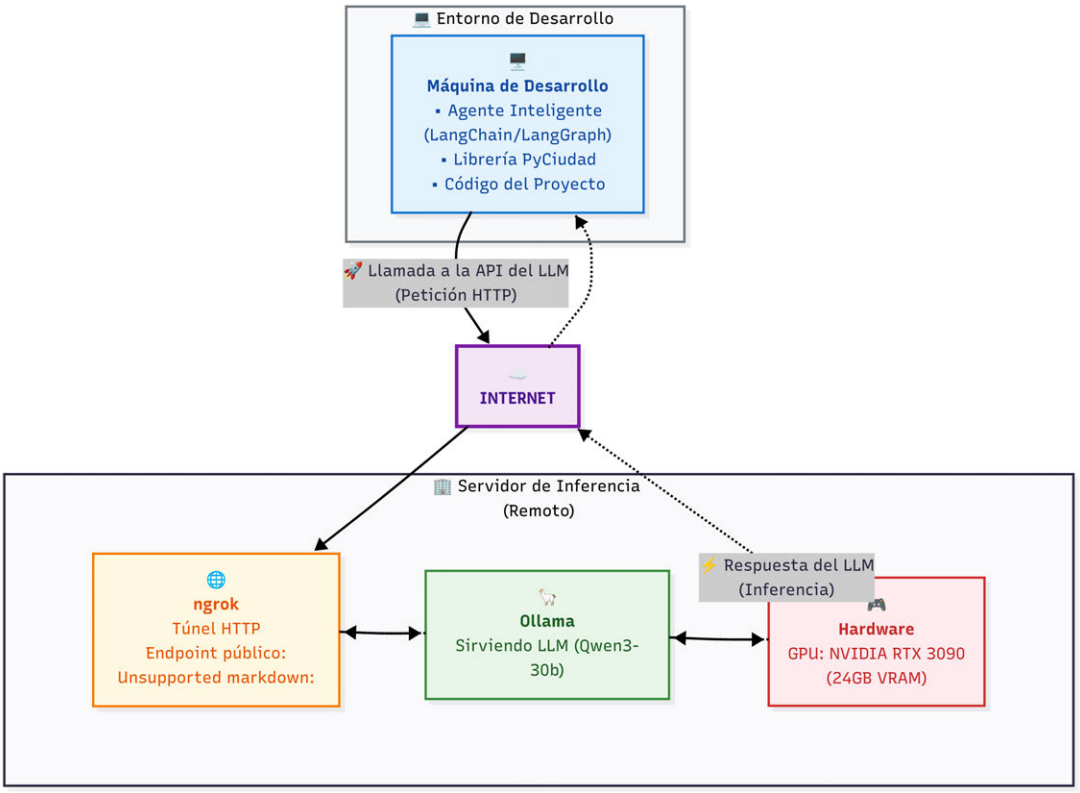

Figura 3.5. Diagrama de la arquitectura de infraestructura implementada. Los agentes, ejecutándose localmente, se comunican a través de un túnel seguro de ngrok con un servidor Ollama remoto, que gestiona la inferencia del modelo LLM en una GPU NVIDIA RTX 3090.

3.2.2. Implementación de un Endpoint con `Ollama` y `ngrok`

Para servir los LLM de manera eficiente en la máquina remota, se optó por utilizar **`Ollama`[31]**. Esta herramienta simplifica la descarga, configuración y ejecución de modelos de código abierto, exponiéndolos a través de una API REST.

Para asegurar una conexión estable y accesible desde cualquier red sin configuraciones complejas de firewall, se implementó una solución utilizando **`ngrok`[46]**. Este servicio crea un túnel seguro desde una red local a un endpoint público en internet, exponiendo la API de `Ollama`[31] y permitiendo que los agentes realizaran inferencias en la RTX 3090 remota de forma transparente.

(a) Ollama (b) Ngrok

Figura 3.6. Herramientas clave para la infraestructura del servicio. Ollama (izquierda) para servir los modelos de lenguaje y Ngrok (derecha) para crear un túnel de red seguro y público.

3.2.3. Criterios para la Selección de Modelos LLM

La disponibilidad de 24 GB de VRAM condicionó la selección de los modelos. El objetivo era encontrar un equilibrio entre el tamaño del modelo y su viabilidad de ejecución. La selección se centró en modelos de código abierto de entre aproximadamente **7 y 32 mil millones de parámetros (7B a 32B)**. Modelos en este rango, como variantes de `Phi-4`, `Gemma-3`, o la familia `Qwen3`, ofrecen un buen compromiso entre capacidad y recursos.

3.2.4. La Necesidad de Cuantización de Modelos

Incluso los modelos en el rango de 7B a 32B, en su precisión original (generalmente `float16` o `bfloat16`), pueden exceder la VRAM disponible. Para hacerlos ejecutables fue imprescindible recurrir a la **cuantización**.

La cuantización es una técnica de compresión que reduce la precisión numérica de los pesos del modelo. En este proyecto, se exploró principalmente la cuantización a 4 bits, que reduce drásticamente el tamaño del modelo en memoria, permitiendo que modelos más grandes y capaces puedan ser cargados en la GPU.

3.3. La Librería PyCiudad

Un requisito fundamental para que los agentes pudieran interactuar con los servicios de CartoCiudad era disponer de una interfaz programática robusta en Python. Ante la ausencia de un wrapper oficial, se tomó la decisión de desarrollar una librería específica para este fin: **`PyCiudad`**.

3.3.1. Ausencia de Wrapper Python para la API REST

La interacción directa con la API REST de CartoCiudad desde el código de los agentes presentaba varias desventajas:

- **Repetición de Código:** Cada llamada a un endpoint requeriría la construcción manual de la URL, el manejo de parámetros y el parseo de la respuesta JSON.
- **Complejidad en el Manejo de Errores:** Gestionar de forma consistente los diversos códigos de estado HTTP y formatos de error incrementaría la complejidad.
- **Falta de Abstracción Orientada a Objetos:** Trabajar con respuestas JSON crudas es menos intuitivo que interactuar con objetos Python que modelen las entidades de CartoCiudad.
- **Integración como Herramienta (*Tool*):** Para que los agentes de `LangChain`[1] pudieran usar los servicios, necesitaban una "herramienta" bien definida que encapsulara la lógica de la API.

3.3.2. Descripción de la Librería `PyCiudad`: Diseño y Funcionalidades

`PyCiudad` es una librería Python desarrollada para proporcionar una interfaz de programación moderna y robusta para los servicios REST de CartoCiudad. El diseño se centra en la simplicidad para el desarrollador y la fiabilidad, utilizando la librería **`Pydantic`** para el modelado de datos, lo que convierte las respuestas de la API en objetos Python con tipos definidos y validación automática.

Las funcionalidades clave que `PyCiudad` ofrece incluyen:

- **Búsqueda Avanzada de Candidatos (`buscar_candidatos`):** Permite realizar consultas textuales para encontrar lugares, soportando un amplio abanico de filtros.
- **Geocodificación Precisa (`geocodificar`):** Transforma descripciones textuales o identificadores en coordenadas geográficas.
- **Geocodificación Inversa (`geocodificacion_inversa`):** Convierte coordenadas geográficas en la dirección postal más relevante.

Además, la librería incorpora un sistema de manejo de errores centralizado y utilidades para la validación de entradas. El desarrollo de `PyCiudad` fue un paso instrumental, ya que permitió a los agentes LLM interactuar de forma programática y fiable con CartoCiudad. Una descripción técnica más detallada se puede encontrar en el **Apéndice A**.

4. Metodología de trabajo: Evolución Arquitectónica de Agentes

4.1. Interacción Geœspacial Avanzada

El motor de búsqueda geoespacial de CartoCiudad, si bien se apoya en una base de datos oficial y de referencia, presenta, como hemos discutido en capítulos anteriores, limitaciones inherentes a su diseño basado en heurísticas y búsqueda léxica estricta. Esta rigidez se traduce en una experiencia de usuario a menudo frustrante, donde la necesidad de formular consultas con una sintaxis precisa y la intolerancia a errores o ambigüedades dificultan la obtención de resultados relevantes.

El foco principal de este capítulo es el diseño e implementación de una serie de arquitecturas de agentes inteligentes, construidas sobre LLM y orquestadas mediante el framework `LangChain/LangGraph`[1], [2]. El objetivo es explorar cómo estas tecnologías pueden superar las deficiencias mencionadas, dotando al sistema de búsqueda de una capa de inteligencia capaz de interpretar la intención del usuario e interactuar de forma más robusta y flexible con la API existente, encapsulada a través de la librería `PyCiudad`.

La metodología adoptada para el desarrollo de estos agentes ha sido incremental y evolutiva. Hemos partido de una solución base, minimalista pero funcional, para luego ir introduciendo progresivamente capas de complejidad. A lo largo de las siguientes secciones, detallaremos la concepción, los componentes y el flujo de ejecución de cada uno de los agentes desarrollados: el Agente Base, el Agente de Intención, el Agente de Validación y, finalmente, el Agente Ensemble.

4.2. La Funcionalidad Base

La interacción directa con la API de CartoCiudad resulta poco práctica para el usuario final. Para abordar esto, se requiere una capa intermedia que permita consultas en lenguaje natural. Como solución, hemos desarrollado el **Agente Base (`app_base`)**, que representa la implementación más directa de un flujo de procesamiento de consultas basado en LLM. Este agente establece un pipeline secuencial que traduce la petición del usuario en una respuesta estructurada, con su lógica principal residiendo en `agent_base.py`.

El Agente Base sirve como una línea de referencia fundamental. Su arquitectura se articula en torno a una secuencia lógica de cuatro etapas cruciales:

1. **Interpretación y Normalización:** Un LLM procesa la entrada del usuario para identificar los elementos geográficos esenciales y refinar la consulta.
2. **Formulación de Parámetros:** El LLM traduce esta comprensión a los parámetros estructurados que espera la API de CartoCiudad.
3. **Interacción con CartoCiudad:** El agente invoca la API externa (a través de `PyCiudad`) y procesa los resultados, eliminando duplicados.
4. **Reordenamiento de Resultados:** Un LLM evalúa los candidatos devueltos en el contexto de la consulta original y los prioriza según la intención inferida del usuario.

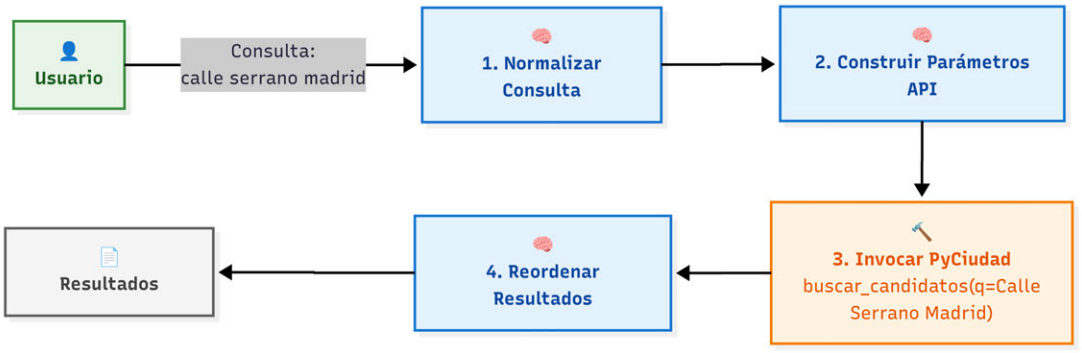

Figura 4.1. Diagrama de flujo del Agente Base. La consulta del usuario pasa por un pipeline secuencial de interpretación, formulación de parámetros, llamada a la API y reordenamiento de resultados.

Este agente, aunque simple, representa una mejora significativa. Sin embargo, su capacidad de interpretación es limitada, siendo vulnerable a la ambigüedad, lo que sienta las bases para arquitecturas más avanzadas.

4.3. Captura de la Intención

El Agente Base, al centrarse en la extracción directa de palabras clave, a menudo falla al interpretar consultas ambiguas. Para superar esta limitación, hemos diseñado el **Agente de Intención (`app_intention`)**. La lógica de este agente, en `agent_intention.py`, introduce una arquitectura que procesa la consulta del usuario de forma paralela para capturar no solo las entidades, sino también la *intención* subyacente.

La clave del Agente de Intención es la descomposición del análisis inicial en dos procesos concurrentes. En paralelo, un proceso se dedica a inferir la intención semántica general de la consulta. La información resultante de ambos procesos (palabras clave e intención) se combina, permitiendo construir una consulta a CartoCiudad más precisa y contextualmente informada. El flujo subsiguiente incluye la llamada a la API y un reordenamiento de candidatos. Este enfoque busca mejorar significativamente la precisión, especialmente ante consultas que requieren "leer entre líneas".

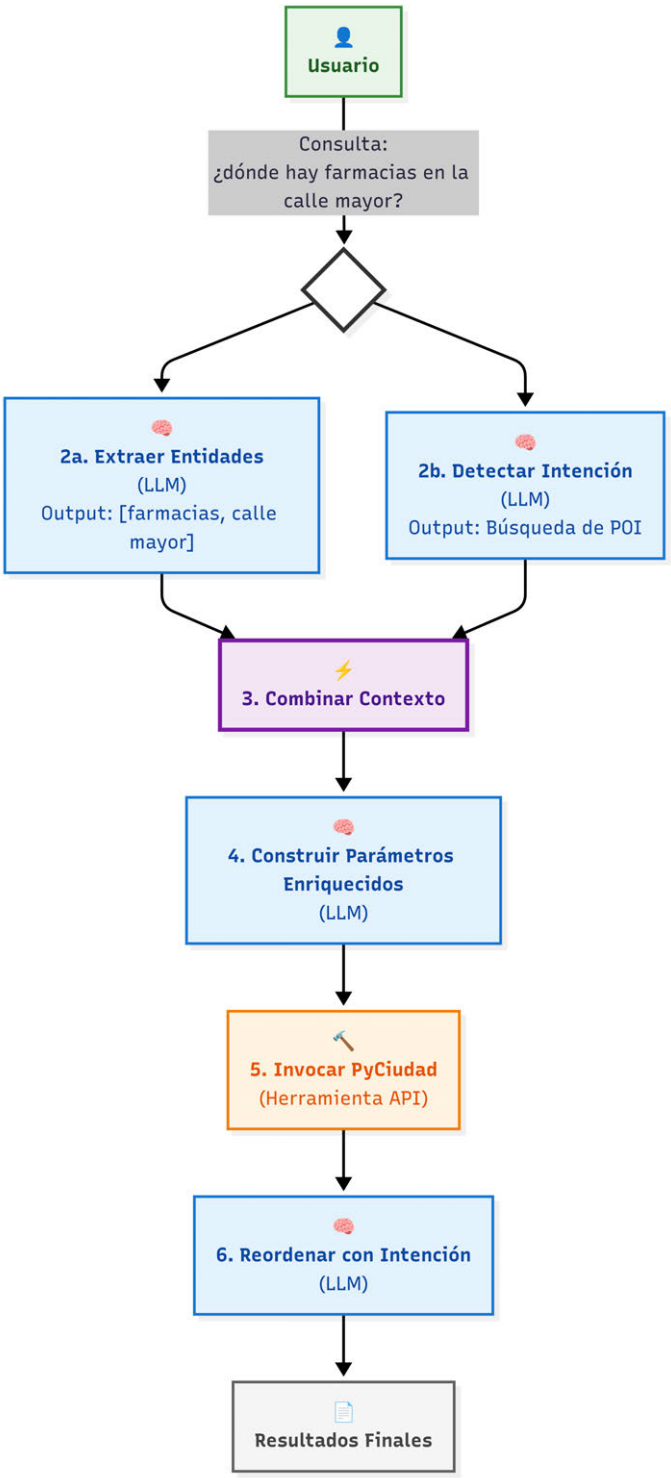

Figura 4.2. Arquitectura del Agente de Intención. Introduce un procesamiento en paralelo que analiza la consulta de forma simultánea para extraer entidades e inferir la intención semántica. Ambos resultados se combinan para una búsqueda más precisa.

4.4. Ciclos de Auto-Corrección

Incluso con una mejor comprensión inicial, los resultados de la API pueden ser insatisfactorios. Un sistema robusto debería ser capaz de reflexionar sobre el resultado e intentar mejorar. Para ello, hemos desarrollado el **Agente de Validación (`app_validation`)**, cuya lógica se implementa en `agent_validation.py`. Este agente implementa un ciclo de auto-mejora:

1. **Búsqueda Inicial:** Se realiza una primera búsqueda para obtener un conjunto inicial de resultados.
2. **Evaluación de Resultados:** Un LLM especializado actúa como validador. Analiza críticamente los resultados y emite una decisión: si son "Suficientes" o si "Necesitan Reformulación".
3. **Decisión de Iterar o Finalizar:** En función de la decisión del validador y un número máximo de reintentos, el sistema decide si continuar.
4. **Reformulación Inteligente:** Si se requiere, otro LLM actúa como reformulador, generando un nuevo conjunto de parámetros para la API basándose en la reflexión del validador.
5. **Nueva Búsqueda y Acumulación:** Se realiza una nueva llamada a la API y los candidatos únicos se acumulan. El ciclo regresa a la etapa de evaluación.
6. **Presentación Final:** Cuando el ciclo concluye, se presentan los candidatos finales y se aplica un último paso de reordenamiento.

Este bucle iterativo, con su capacidad de auto-corrección, dota al sistema de una notable capacidad de adaptación.

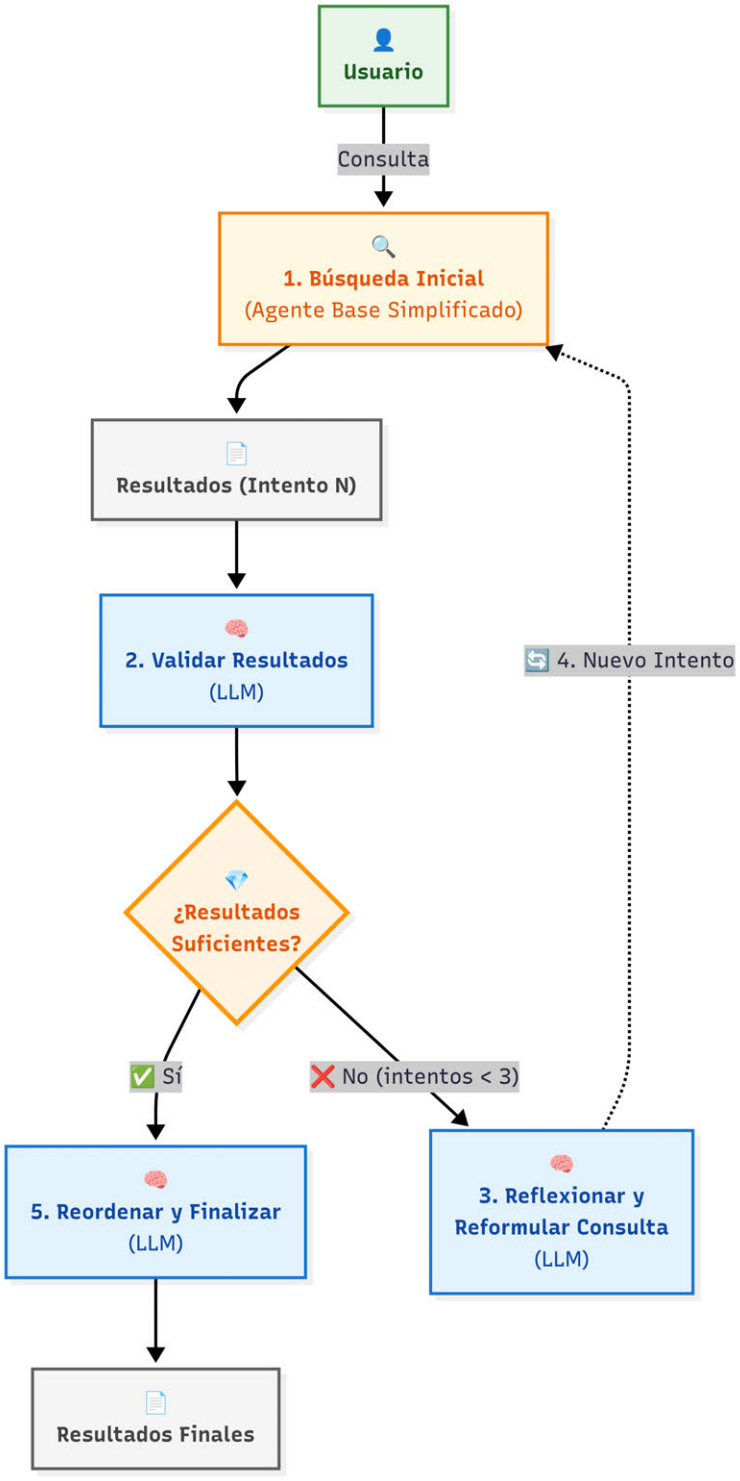

Figura 4.3. Flujo de trabajo del Agente de Validación. Implementa un ciclo de auto-corrección donde los resultados son evaluados; si se consideran insuficientes, la consulta se reformula y se reintenta la búsqueda, acumulando candidatos en cada iteración.

4.5. Estrategias Dinámicas

Los agentes anteriores ofrecen distintos niveles de sofisticación y coste. Aplicar el más complejo a todas las consultas sería ineficiente. Surge así la necesidad de un mecanismo que dirija la consulta al pipeline más apropiado. Como respuesta, se ha diseñado el **Agente Ensemble (`app_ensemble`)**, con su lógica en `agent_ensemble.py`. Actúa como un metagente orquestador que selecciona dinámicamente el pipeline más adecuado, con capacidad de escalado automático.

Este agente introduce una lógica de control de dos etapas:

1. **Selección Estratégica Inicial:** Un LLM, actuando como "Metaevaluador", analiza la consulta y determina si debe ser manejada por el pipeline Simple (Agente Base), Intermedio (Agente de Intención) o Complejo (Agente de Validación).
2. **Validación y Escalado Automático:** Una vez que el pipeline escogido ha procesado la consulta:
   - Si produjo candidatos, el proceso concluye.
   - Si no produjo candidatos, el sistema puede decidir escalar automáticamente la tarea al siguiente pipeline más complejo en la jerarquía.
3. **Salida Final:** El agente presenta los resultados del primer pipeline que tuvo éxito o, en su defecto, los del último intentado.

El Agente Ensemble busca optimizar el rendimiento global del sistema, proporcionando un equilibrio robusto entre eficiencia y efectividad.

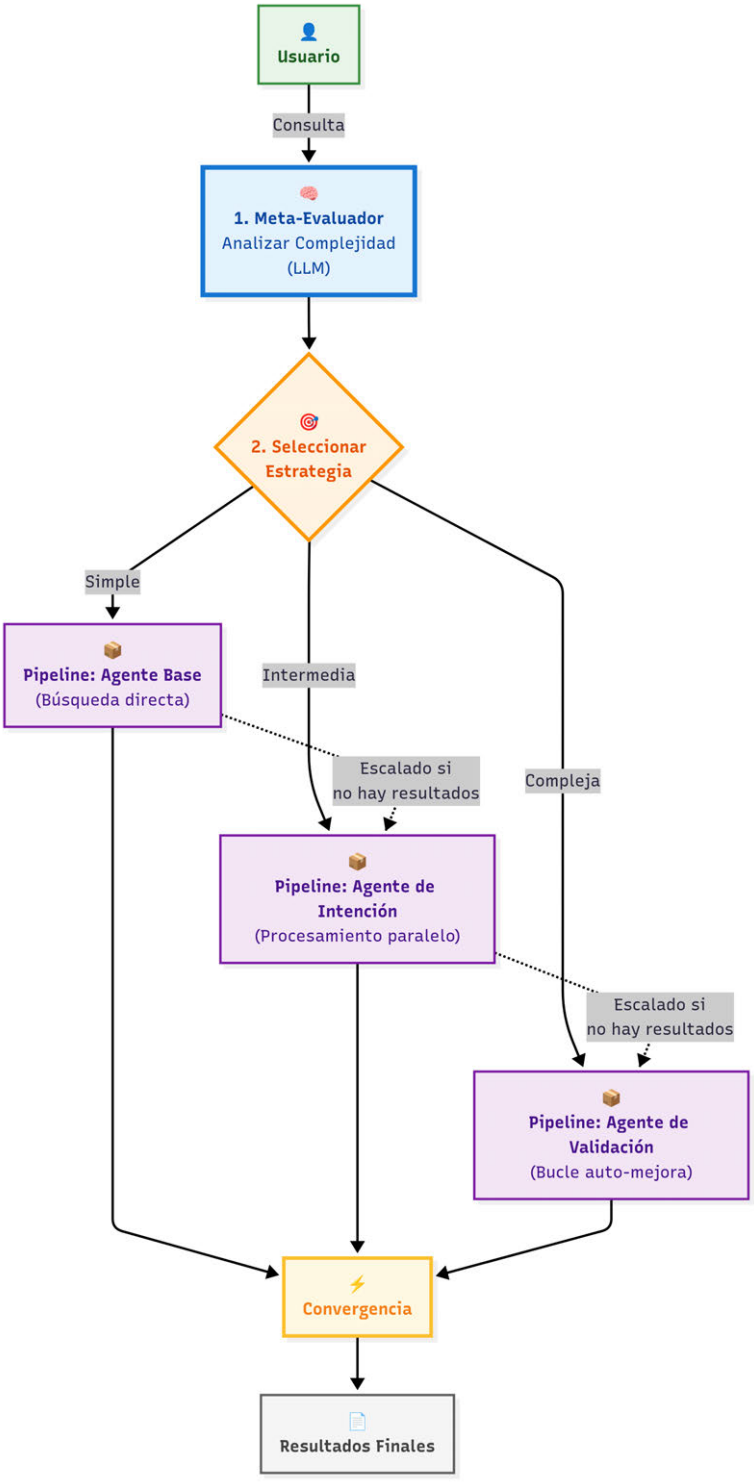

Figura 4.4. Lógica del Agente Ensemble. Actúa como un meta-orquestador que primero clasifica la consulta para seleccionar el pipeline más adecuado (Simple, Intermedio o Complejo). Si el pipeline elegido falla, puede escalar automáticamente la tarea al siguiente nivel de complejidad.

4.6. Evolución y Potencial

El recorrido a través de los agentes Base, de Intención, de Validación y Ensemble ilustra una progresión evolutiva en la búsqueda de una solución más robusta para CartoCiudad.

El **Agente Base (`app_base`)** sentó los cimientos con un pipeline funcional. El **Agente de Intención (`app_intention`)** incorporó el procesamiento paralelo para detectar la intencionalidad, mejorando el manejo de consultas ambiguas. El **Agente de Validación (`app_validation`)** introdujo un ciclo de autorreflexión y mejora iterativa. Finalmente, el **Agente Ensemble (`app_ensemble`)** culminó la evolución actuando como un meta-orquestador inteligente que optimiza el uso de recursos.

A lo largo de esta evolución, han emergido capacidades significativas que transforman la interacción con el buscador:

- Comprensión mejorada del lenguaje natural.
- Manejo progresivo de la ambigüedad.
- Robustez ante errores y resultados subóptimos.
- Adaptabilidad dinámica de la estrategia de procesamiento.
- Modularidad y escalabilidad gracias a la estructura de grafos.

La Paradoja de la Complejidad Incremental. Es fundamental establecer que, si bien cada agente sucesivo introduce mecanismos que, a priori, deberían mejorar el rendimiento, la experiencia práctica demuestra que la adición de sofisticación no siempre se traduce en mejores resultados. Esto puede deberse a la naturaleza estocástica de los LLM, la introducción de nuevos puntos de fallo o la acumulación de latencia.

Limitaciones Técnicas Fundamentales. Además, es crucial reconocer las limitaciones que condicionan el rendimiento de todos los agentes:

- **Capacidades Limitadas del Modelo Base:** Los LLM empleados operan dentro de restricciones de capacidad que afectan a su habilidad para realizar razonamiento complejo.

- **Ecosistema de Herramientas Restringido:** Los agentes carecen de acceso a herramientas externas como motores de búsqueda web que podrían enriquecer su capacidad de contextualización.
- **Grado de Libertad Limitado:** Las arquitecturas operan dentro de restricciones impuestas por la memoria de contexto y las capacidades del framework.
- **Dependencia del Conocimiento Interno del Modelo:** El rendimiento está intrínsecamente vinculado al conocimiento geográfico preentrenado en el modelo base, lo cual es un cuello de botella fundamental.

Esta observación no invalida el valor de la exploración realizada, sino que subraya la importancia de la validación empírica rigurosa frente a las asunciones teóricas.

5. Diseño del Dataset de Evaluación

La evaluación rigurosa y comparativa de las arquitecturas de agentes diseñadas en el capítulo anterior constituye un pilar fundamental de este trabajo. Sin embargo, para que dicha evaluación sea significativa, se requiere de un conjunto de datos (dataset) que no solo sea extenso, sino que también refleje fielmente la diversidad y complejidad de las consultas que un usuario real podría formular.

Ante la ausencia de un dataset público y estandarizado para esta tarea específica en el contexto español, nos enfrentamos a un desafío metodológico clave: la necesidad de construir nuestro propio corpus de evaluación desde cero. Este capítulo detalla el proceso de ingeniería de datos que hemos seguido, una estrategia híbrida que combina la extracción de una base de datos real (Ground Truth) desde la propia API de CartoCiudad con un proceso de generación sintética.

5.1. Un Ground Truth Basado en Datos Reales

La credibilidad de cualquier evaluación descansa sobre la veracidad de su Ground Truth. Por ello, el primer paso en la construcción de nuestro dataset ha sido la extracción sistemática de un conjunto de datos geográficos reales y verificados directamente desde la API de CartoCiudad, utilizando la librería `PyCiudad`.

5.1.1. Extracción y Filtrado de Entidades Geográficas

Nuestro proceso de extracción se diseñó para obtener una muestra representativa de la toponimia urbana española. Hemos enfocado las consultas a la API de CartoCiudad utilizando términos de búsqueda genéricos pero semánticamente ricos, como “calle”, “plaza”, “avenida” y “paseo”.

Para evitar una sobresaturación de resultados de grandes capitales y garantizar la diversidad, hemos implementado una lógica de filtrado y muestreo. El proceso itera sobre

las diferentes Comunidades Autónomas de España, realizando búsquedas controladas dentro de cada una y aplicando un límite al número de resultados extraídos por consulta. De esta manera, nos aseguramos de que el dataset base no esté sesgado y contenga una representación equilibrada de la geografía española. El resultado es un conjunto de datos inicial que contiene entidades geográficas reales, cada una con su identificador único, nombre, tipo, municipio y provincia.

5.2. La Necesidad de la Generación Sintética

El dataset extraído presenta una limitación fundamental: consiste en datos estructurados y nomenclaturas oficiales. No refleja cómo un ser humano buscaría esas mismas direcciones. Un usuario no escribe "CALLE MAYOR, 28013, MADRID"; podría escribir "¿dónde está la calle mayor en madrid?" o "c/mayor madrid".

5.2.1. La Brecha entre el Dato Oficial y la Consulta Real

La principal carencia es la ausencia de registros históricos de consultas de usuarios en CartoCiudad. Al no disponer de un log de búsquedas reales, es imposible analizar los patrones de lenguaje y errores con los que los usuarios interactúan con el sistema. Esta brecha entre el dato oficial y la consulta natural es el problema que la generación sintética busca resolver.

5.2.2. Diseño de un Agente Generador Especializado

Para abordar este desafío, hemos diseñado un agente generador. Este agente, implementado con `LangChain[1]` y un LLM, toma como entrada cada dirección de nuestro Ground Truth y la utiliza como semilla para generar un conjunto de consultas sintéticas. Su arquitectura está pensada para producir variaciones controladas, transformando una entrada de datos estructurada en múltiples salidas de texto no estructurado que simulan diferentes intenciones y estilos de comunicación.

5.2.3. De Dirección a Consulta Natural

El núcleo de nuestro agente generador es un pipeline que, para cada dirección, ejecuta una serie de transformaciones lingüísticas. El LLM recibe la dirección oficial junto con instrucciones específicas (prompts) para reescribirla de múltiples maneras, manteniendo siempre la intención de búsqueda original pero alterando la forma.

5.3. Variaciones Lingüísticas y Complejidad Semántica

Para asegurar que nuestro dataset cubra un amplio espectro de interacciones, hemos definido una taxonomía que clasifica las queries generadas.

5.3.1. Taxonomía de Tipos de Consulta

El agente generador fue instruido para producir variantes que se ajustaran a cuatro categorías principales:

- **Directa:** Consultas que se asemejan mucho al formato de la base de datos (ej. "Calle Alcalá 100 Madrid").
- **Natural:** Formulaciones más cercanas al lenguaje hablado, pero todavía claras (ej. "Busco la dirección Calle de Alcalá número 100 en Madrid").
- **Coloquial:** Incorpora abreviaturas, omisiones o un tono informal (ej. "¿Dónde pilla la c/ alcalá 100 en madrid?").
- **Pregunta:** Consultas formuladas explícitamente como una interrogación (ej. "¿Podrías decirme dónde está la Calle Alcalá 100 en Madrid?").

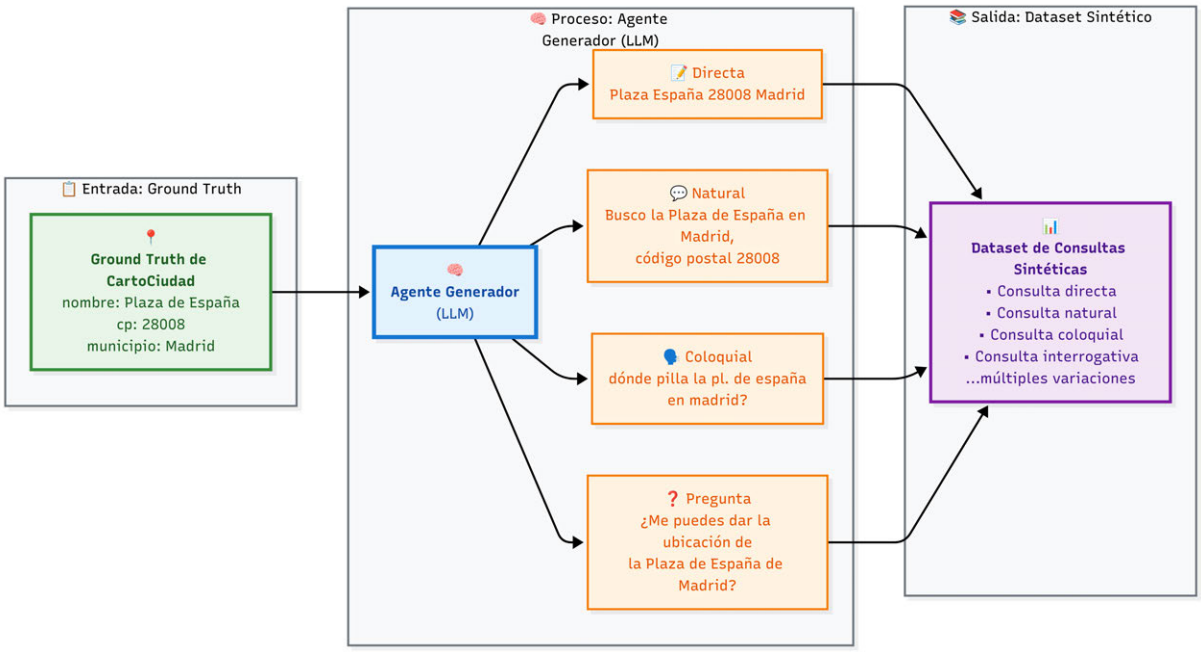

Figura 5.1. Fase inicial del proceso de generación sintética. A partir de una dirección del ground truth, el agente generador es instruido para crear un conjunto de consultas que cubren diferentes estilos lingüísticos (Directa, Natural, Coloquial) y niveles de dificultad (Fácil, Medio, Difícil).

5.3.2. Niveles de Dificultad y Complejidad

Además del estilo, cada consulta generada se clasifica en uno de tres niveles de dificultad:

- **Fácil:** Consultas claras, con pocos elementos ambiguos.
- **Medio:** Introducen ambigüedad moderada, como la omisión de la localidad.
- **Difícil:** Consultas deliberadamente ambiguas, incompletas o que requieren un mayor grado de inferencia.

5.3.3. Prompting Estructurado para una Generación Controlada

La clave para lograr esta generación clasificada reside en el *prompting* estructurado. Para cada dirección del Ground Truth, el agente generador utiliza una plantilla de prompt que le pide al LLM generar una variación para cada categoría y nivel de dificultad, proporcionando directrices claras.

5.4. La Inyección Adversarial de Errores

Un sistema robusto debe ser tolerante a los errores. Para evaluar esta capacidad, hemos realizado una inyección adversarial de errores en una parte de las consultas sintéticas.

5.4.1. Simulación de Errores de Usuario

El objetivo es simular los descuidos más comunes que cometen los usuarios al teclear en castellano. El agente generador fue instruido para aplicar una o más de las siguientes degradaciones:

- **Errores de Tildes:** Omisión o adición incorrecta de acentos (“avenida de andalucia”).
- **Errores Ortográficos:** Errores de tecleo comunes (“calle serano”).
- **Errores de Espaciado:** Falta de espacios o dobles espacios (“paseodela castellana”).
- **Abreviaciones No Estándar:** Uso de abreviaturas coloquiales (“avda.”, “pl.”).
- **Uso Inconsistente de Mayúsculas/Minúsculas.**

5.4.2. Degradación Sintáctica sin Pérdida de Intención

Es fundamental subrayar que este proceso se realiza de forma controlada. La instrucción al LLM es clara: degradar la sintaxis de la consulta, pero preservando siempre la intención semántica original. De este modo, nos aseguramos de que la consulta sigue siendo “resoluble” y que estamos evaluando la robustez del agente, no pidiéndole que adivine una entrada sin sentido.

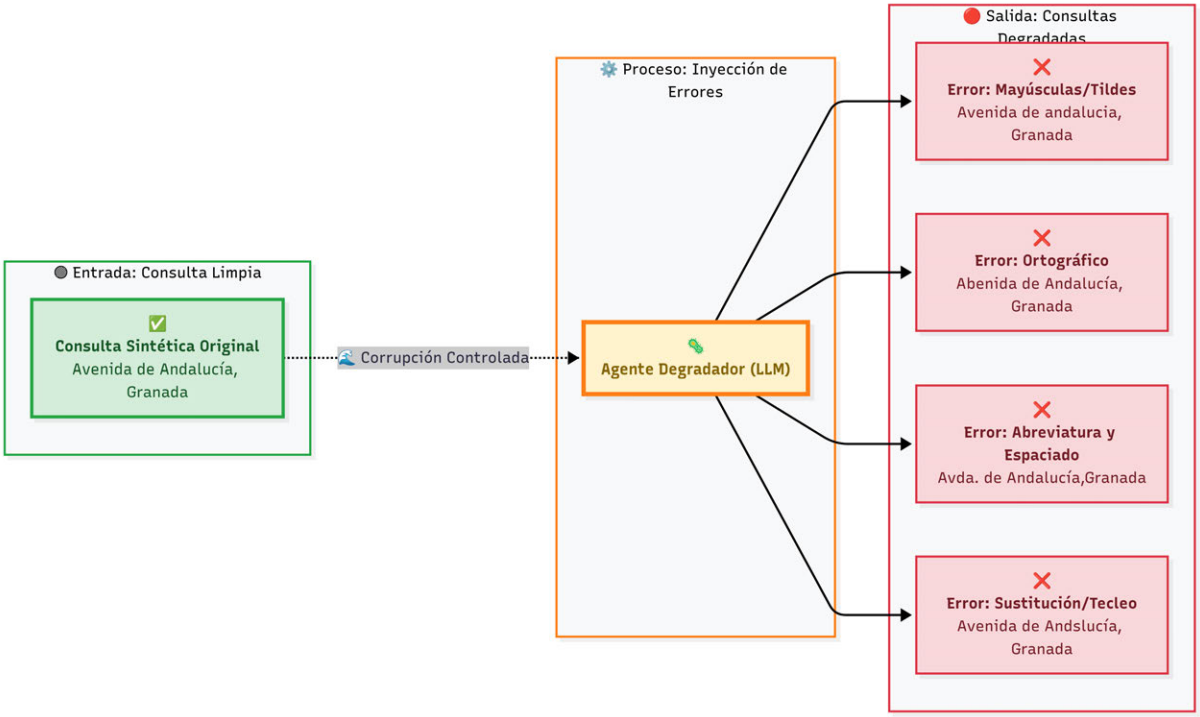

Figura 5.2. Fase de inyección adversarial de errores. Un subconjunto de las consultas sintéticas generadas se somete a un proceso de degradación controlada, donde el LLM introduce patrones de error comunes en castellano (omisión de tildes, errores de tecleo, etc.) sin alterar la intención de búsqueda subyacente.

Tabla 5.1. Taxonomía de Consultas para el Dataset de Evaluación.

Tipo de Consulta	Nivel: Fácil	Nivel: Medio	Nivel: Difícil
Directa	`Calle Serrano 50 Madrid` *Clara, completa, casi como en una BBDD.*	`Calle Real Pozuelo` *Nombre de calle común, falta provincia para desambiguar.*	`Estadio Bernabeu` *Nombre de lugar, sin tipo de vía ni ciudad explícita.*
Natural	`Busco la Avenida de la Constitución 5 en Sevilla` *Lenguaje conversacional pero estructurado.*	`Quiero ir a la plaza principal de Cáceres` *Petición vaga ("plaza principal") que requiere inferencia.*	`Buscando la casa donde nació Picasso en Málaga` *Búsqueda por referencia cultural.*
Coloquial	`c/ gran via madrid` *Abreviaturas comunes, sin ambigüedad.*	`dnd sta la plza mayor?` *Abreviatura, falta de tildes y omisión de localidad.*	`la avda q va a la playa en valencia` *Descripción puramente funcional, sin nombres.*
Pregunta	`¿Dónde está la Sagrada Familia en Barcelona?` *Pregunta directa sobre un POI conocido.*	`¿Cuál es el código postal de la calle Colón?` *Nombre muy común, genera alta ambigüedad.*	`¿Puedes encontrar la calle del ayuntamiento en el pueblo de mi abuela cerca de Lugo?` *Extremadamente vaga, con información no procesable.*

Justificación de los ejemplos elegidos:

- **Fácil:** Consultas que un sistema léxico con buenas heurísticas podría resolver. Sirven para establecer la línea base.

- **Medio:** Introducen ambigüedad o vaguedad, el punto donde se espera que los agentes LLM superen a los sistemas tradicionales.
- **Difícil:** Diseñadas para llevar a los agentes al límite, requiriendo inferencia, conocimiento del mundo o la capacidad de reconocer que una consulta es irresoluble.

5.5. Estructura Final del Dataset

El resultado final es un dataset estructurado y listo para ser consumido por nuestro pipeline de evaluación. Cada entrada se almacena en un formato JSON con tres componentes principales:

1. **Ground Truth:** Un objeto que contiene la información original y verificada de la entidad geográfica, incluyendo su `id_unico_coincidencia`.
2. **Query:** La cadena de texto de la consulta sintética generada, que será la entrada para los agentes a evaluar.
3. **Metadatos:** Un conjunto de etiquetas que describen la consulta, incluyendo su `tipo` (Directa, Natural, etc.), su `dificultad` (Fácil, Media, Difícil) y si contiene `errores` inyectados.

Esta estructura nos permite no solo automatizar la evaluación, sino también realizar análisis de resultados segmentados. Con este dataset de calidad, estamos preparados para proceder a una evaluación exhaustiva y fundamentada de nuestras arquitecturas de agentes.

6. Evaluación y Resultados

Este capítulo presenta la evaluación sistemática del rendimiento de las diversas arquitecturas de agentes inteligentes diseñadas en el Capítulo 4. El objetivo es cuantificar y comparar su eficacia en la mejora de las búsquedas geográficas en CartoCiudad, utilizando el dataset sintético desarrollado en el Capítulo 5 y aplicando la metodología de evaluación detallada a continuación.

6.1. Metodología de Evaluación y Protocolo Experimental

Para valorar de forma rigurosa y objetiva el rendimiento de las distintas arquitecturas de agentes propuestas, se ha diseñado e implementado un framework de evaluación.

Dataset de Prueba y Ground Truth

La piedra angular de la evaluación es el **dataset sintético descrito en el Capítulo 5**, que consta de **747 consultas únicas**. Este conjunto de datos está diseñado para emular la diversidad y complejidad de las interacciones de un usuario real, incluyendo:

- Consultas estructuradas y bien formadas (486 consultas).
- Consultas con errores ortográficos y típicos (261 consultas).
- Consultas vagas, ambiguas, en lenguaje natural y coloquial.
- Variaciones en tipos de consulta y niveles de dificultad.

Métricas de Evaluación Implementadas

Se ha definido un conjunto de métricas multidimensionales para capturar tanto el rendimiento técnico como la calidad semántica de los resultados.

- **Tasa de Éxito Técnico:** Porcentaje de ejecuciones que finalizan sin errores irrecuperables.
- **Tiempo de Respuesta (Latencia):** Latencia total en segundos (media y mediana).
- **Sistema de Puntuación Posicional (Quality Score):** Para las consultas donde el Ground Truth es encontrado, se asigna una puntuación basada en su posición:
  - **1.0 punto (Perfect Match):** Resultado correcto en la primera posición.
  - **0.8 puntos (Top 3):** Resultado correcto entre las posiciones 2 y 3.
  - **0.6 puntos (Top 5):** Resultado correcto entre las posiciones 4 y 5.
  - **0.3 puntos (Found Far):** Resultado correcto más allá de la quinta posición.
  - **0.0 puntos (Not Found):** El resultado correcto no se encuentra en la lista.
- **Puntuación Media de Relevancia (MRS):** El promedio del *Quality Score* sobre todo el dataset.
- **Tasa de Encontrado (Found Rate):** Porcentaje de consultas donde el Ground Truth fue localizado.
- **Distribución por Tramos de Calidad:** Desglose porcentual en las categorías Perfect Match, Top 3, Top 5 y Not Found.

Framework de Evaluación Automatizada y Monitorización

Se ha desarrollado un sistema programático que orquesta la ejecución masiva de consultas contra cada arquitectura. Este framework gestiona la carga del dataset, la instanciación de los agentes, la ejecución de consultas y la captura de métricas, integrándose con **`LangSmith`**[45] para asegurar la trazabilidad completa, la depuración y el análisis de las cadenas de pensamiento de los agentes. Adicionalmente, incorpora puntos de control (checkpoints) y reintentos con espera exponencial (*exponential backoff*).

Configuración del Entorno y Modelo LLM Base

La selección del modelo de lenguaje base fue **`Qwen3-30b-a3b`**. Esta decisión se basó en que, incluso en su versión cuantizada `Q4_K_M`, demostró un equilibrio superior entre capacidad de razonamiento, seguimiento de instrucciones y latencia aceptable. Modelos más pequeños mostraron limitaciones en la comprensión de la ambigüedad y la generación de JSON estructurado. Se optó por una **temperatura baja** para favorecer respuestas deterministas. Todas las pruebas se realizaron sobre la infraestructura descrita en el Capítulo 3: una GPU NVIDIA RTX 3090, sirviendo el modelo vía `Ollama`[31] y `ngrok`[46].

6.2. Resultados Comparativos de las Arquitecturas

A continuación, se presentan los resultados obtenidos por cada una de las arquitecturas de agentes evaluadas.

Tabla 6.1. Tabla Resumen General de Rendimiento por Agente (Dataset Completo - 747 consultas).

Métrica	Base	Intención	Validación	Ensemble
Latencia Media (s)	**3.56**	5.45	24.07	18.00
Latencia Mediana (s)	**3.37**	5.13	17.46	13.52
MRS (Quality Score Promedio)	**0.691**	0.413	0.635	0.638
Tasa de Encontrado (%)	**72.69 %**	42.57 %	65.20 %	67.20 %
% Perfect Match (Pos 1)	**61.8 %**	38.2 %	59.2 %	57.6 %
% Top 3 (Pos 1-3)	**69.2 %**	41.6 %	63.7 %	63.6 %
% Top 5 (Pos 1-5)	**70.2 %**	41.8 %	64.5 %	64.7 %
% Not Found	**27.3 %**	57.4 %	34.8 %	32.8 %

Mejores resultados por fila en negrita.

Análisis Detallado de las Arquitecturas

Agente Base: Establece el punto de referencia con una mediana de **3.37 segundos** de ejecución y el MRS más alto, **0.691**.

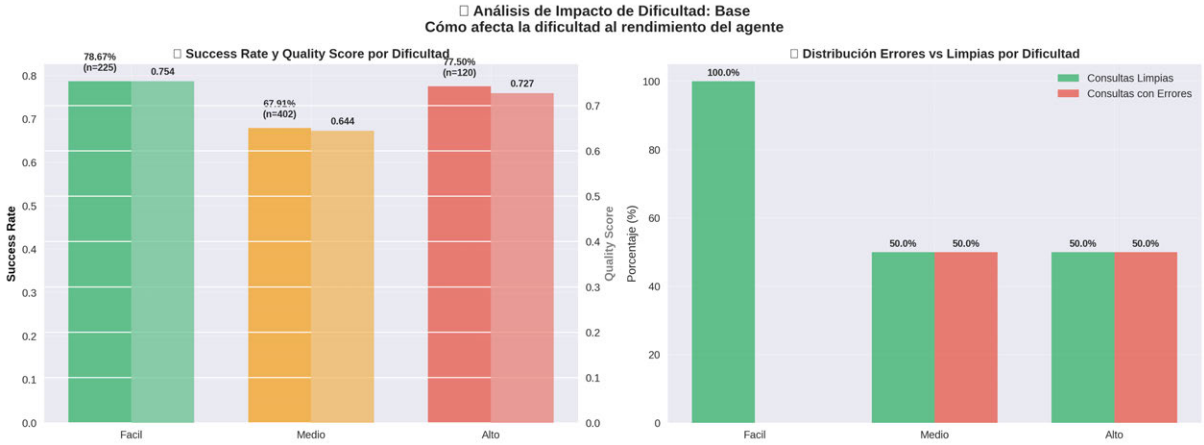

Figura 6.1. Impacto de la dificultad de la consulta en el rendimiento del Agente Base. Se muestra la Puntuación Media de Relevancia (MRS) y la Tasa de Éxito para cada nivel de dificultad.

Al analizar el impacto de la dificultad (Figura 6.1), se observa que el agente es robusto, con un rendimiento en dificultad "Alta" (0.727 MRS) casi tan bueno como en "Fácil" (0.754 MRS). Su principal punto débil es la sensibilidad a los errores de entrada, especialmente las **abreviaciones** y **sustituciones de caracteres**.

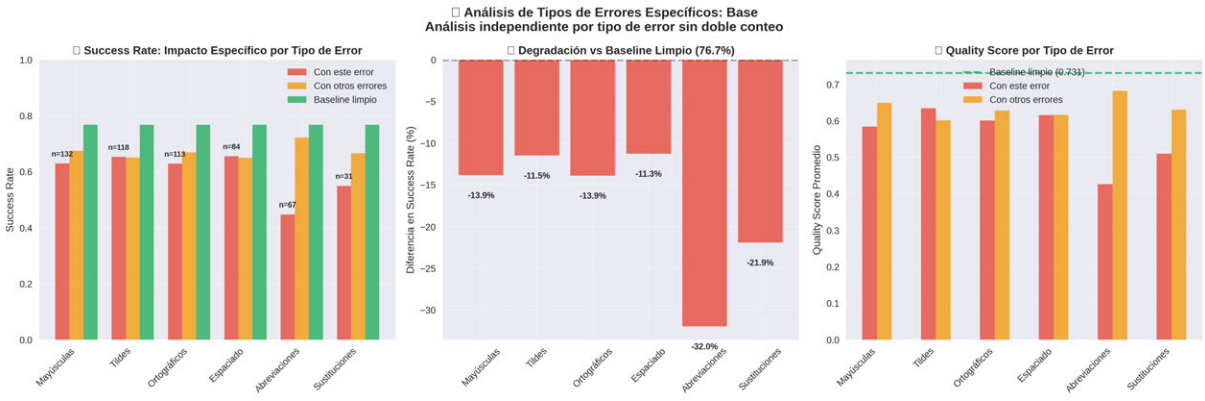

Figura 6.2. Análisis de la robustez del Agente Base ante tipos de error específicos. El gráfico muestra cómo varía el MRS en presencia de diferentes degradaciones en la consulta.

Agente de Intención: Los resultados fueron consistentemente los peores, con el MRS más bajo (0.413) y la tasa de `not_found` más alta (57.4 %). El análisis confirma la de-

gradación del rendimiento con la complejidad, contrario a la hipótesis inicial.

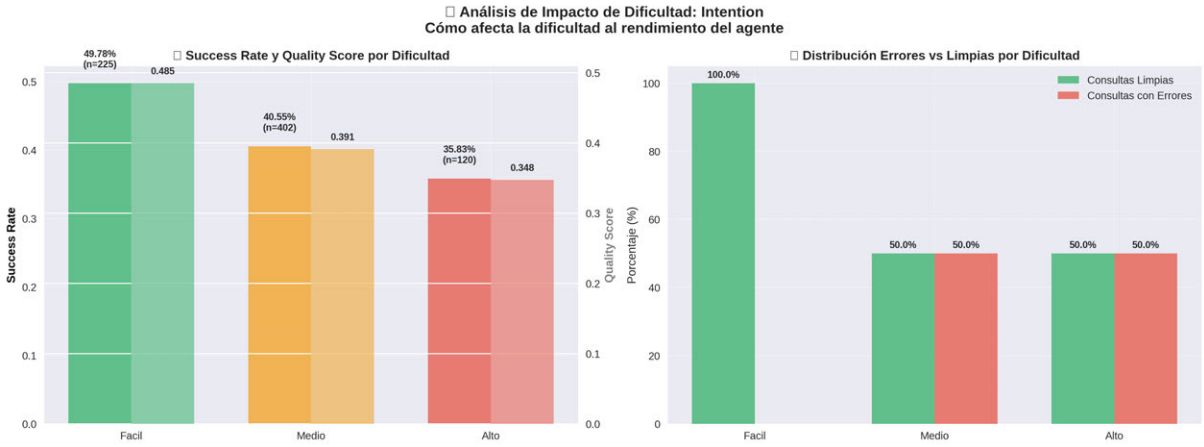

Figura 6.3. Impacto de la dificultad de la consulta en el rendimiento del Agente de Intención.

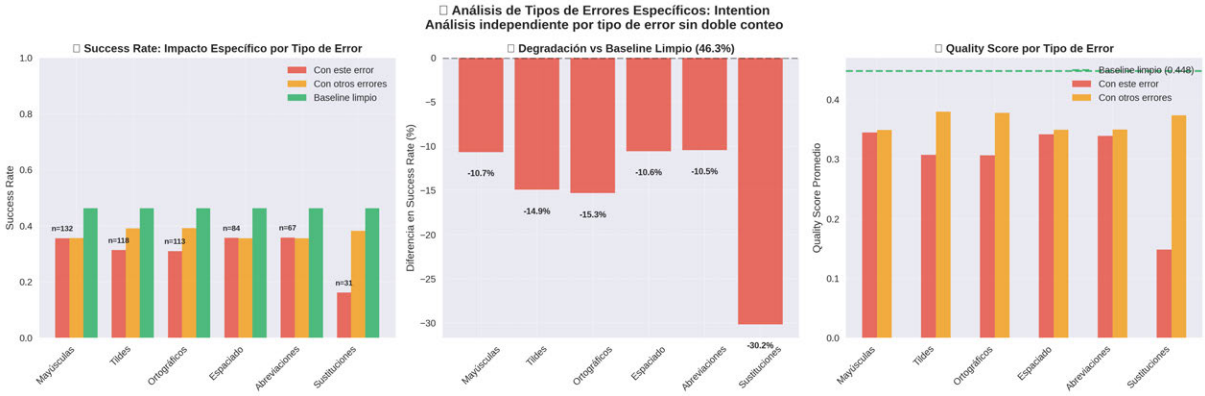

Figura 6.4. Análisis de la robustez del Agente de Intención ante tipos de error específicos.

Agente de Validación: Logra una calidad respetable (0.635 MRS) a costa de una latencia muy elevada (media de 24.07s). Muestra una tendencia interesante: el rendimiento es mejor en consultas de dificultad “Fácil” y “Alta”, pero sufre en “Media”.

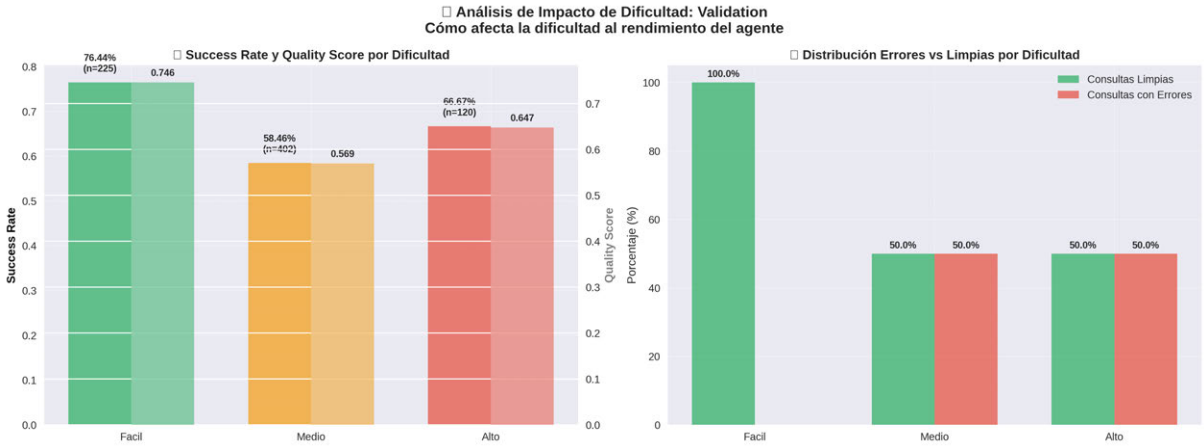

Figura 6.5. Impacto de la dificultad de la consulta en el rendimiento del Agente de Validación.

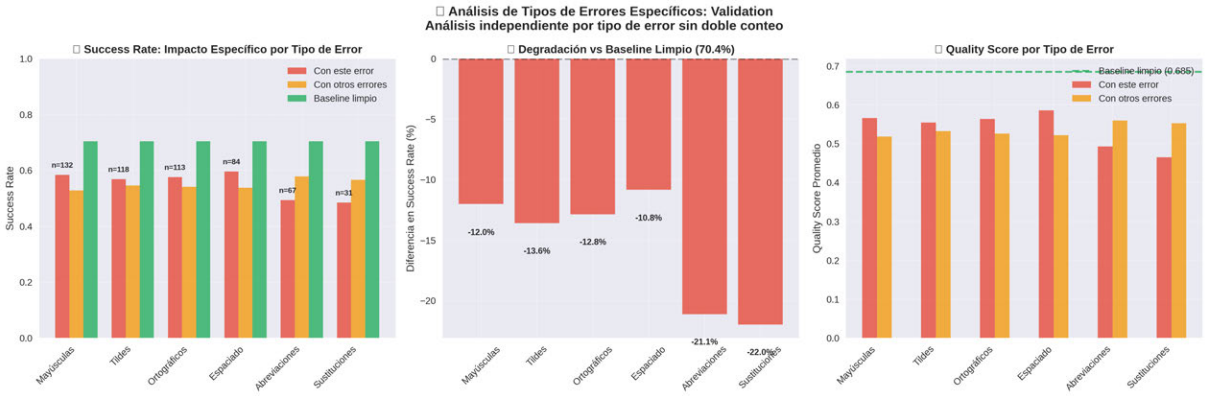

Figura 6.6. Análisis de la robustez del Agente de Validación ante tipos de error específicos.

Agente Ensemble: Actúa como un meta-orquestador. Es más rápido que el Agente de Validación (18.0s) pero mucho más lento que el Base. Su MRS (0.638) es casi idéntico al del Agente de Validación. A diferencia de este, muestra una degradación más lineal con la dificultad.

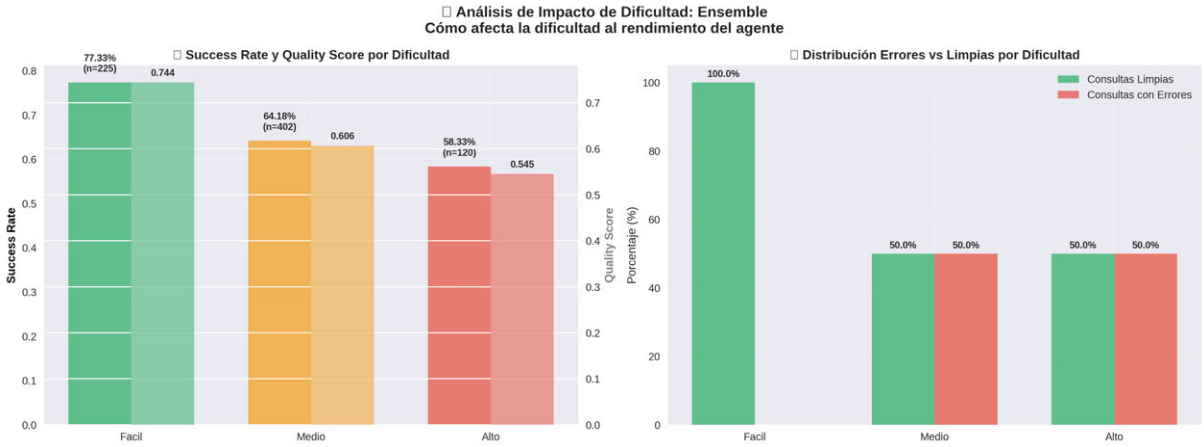

Figura 6.7. Impacto de la dificultad de la consulta en el rendimiento del Agente Ensemble.

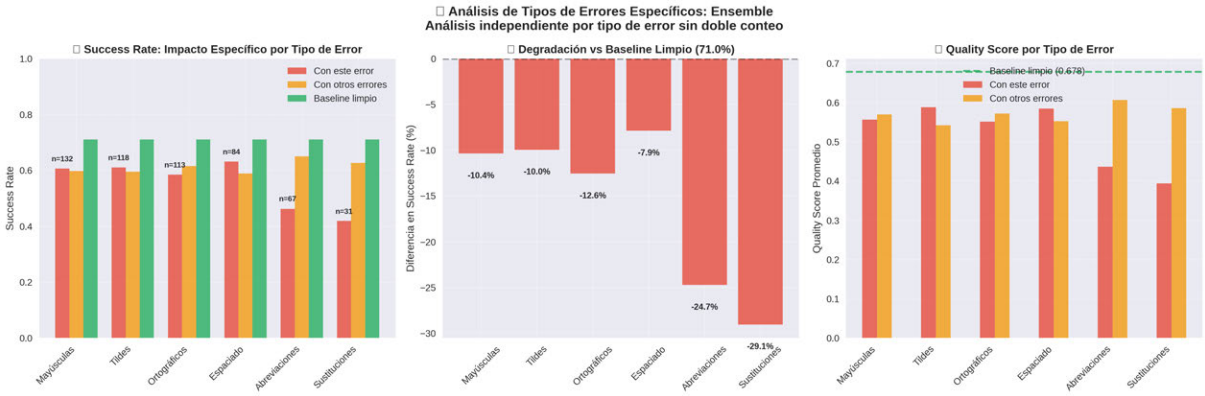

Figura 6.8. Análisis de la robustez del Agente Ensemble ante tipos de error específicos.

Robustez Comparativa ante Errores Específicos

Para entender cómo cada arquitectura maneja las imperfecciones en la entrada, se compara directamente su MRS ante diferentes tipos de error.

Tabla 6.2. MRS por Tipo de Error (Mayor valor = Mejor rendimiento).

Tipo de Error	Base	Intención	Validación	Ensemble
Mayúsculas	0.584	0.345	**0.566**	0.556
Tildes	**0.634**	0.307	0.554	0.588
Ortográficos	**0.601**	0.306	0.564	0.551
Espaciado	**0.615**	0.342	0.586	0.585
Abreviaciones	0.425	0.339	**0.493**	0.436
Sustituciones	**0.510**	0.148	0.465	0.394

Este análisis confirma que, si bien el **Agente de Validación** es el mejor manejando abreviaciones y mayúsculas, el **Agente Base** es superior en los demás tipos de error.

6.3. Discusión y Trade-offs

El análisis comparativo revela una serie de trade-offs críticos entre robustez, calidad y eficiencia.

Robustez General vs. Sofisticación

Un objetivo clave era mejorar la robustez. La Figura 6.9 ofrece una visión comparativa.

Sorprendentemente, el **Agente Base** y el **Agente Ensemble** se muestran como los más robustos, manteniendo aproximadamente el 85 % de su efectividad en consultas con errores. El **Agente de Validación**, a pesar de su ciclo de reformulación, es significativamente menos robusto.

El Balance Óptimo: Calidad vs. Eficiencia (Frontera de Pareto)

Para visualizar el trade-off fundamental, se utiliza un análisis de Frontera de Pareto.

La Figura 6.10 es concluyente. El **Agente Base** se sitúa en la Frontera de Pareto, lo que significa que no existe ninguna otra arquitectura que lo supere simultáneamente en calidad y eficiencia. Los agentes **Validación** y **Ensemble** son "dominados".

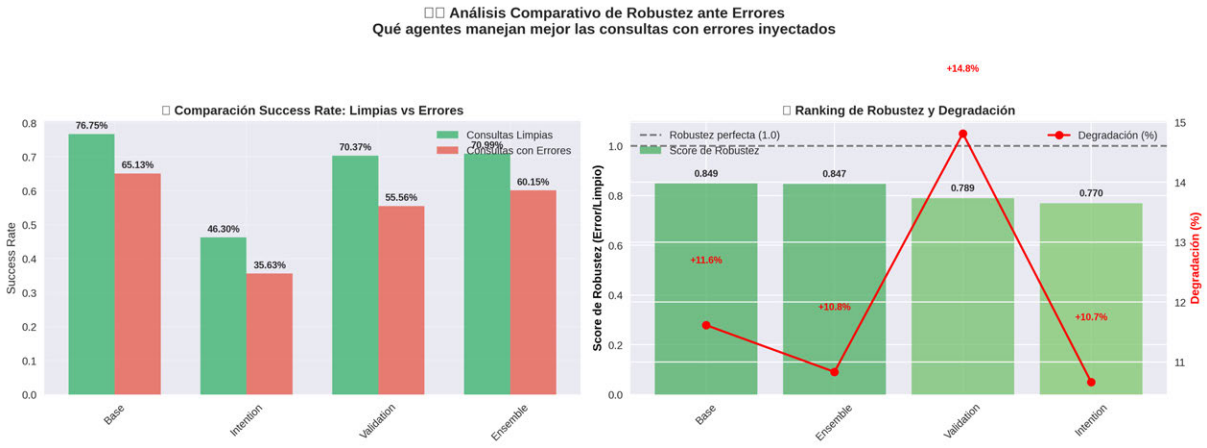

Figura 6.9. Análisis comparativo de la robustez de los agentes. El gráfico izquierdo muestra la caída en la Tasa de Éxito cuando se introducen errores. El derecho clasifica los agentes según su "Índice de Robustez"(Tasa de Éxito con errores / Tasa de Éxito sin errores), donde valores más cercanos a 1 indican mayor robustez.

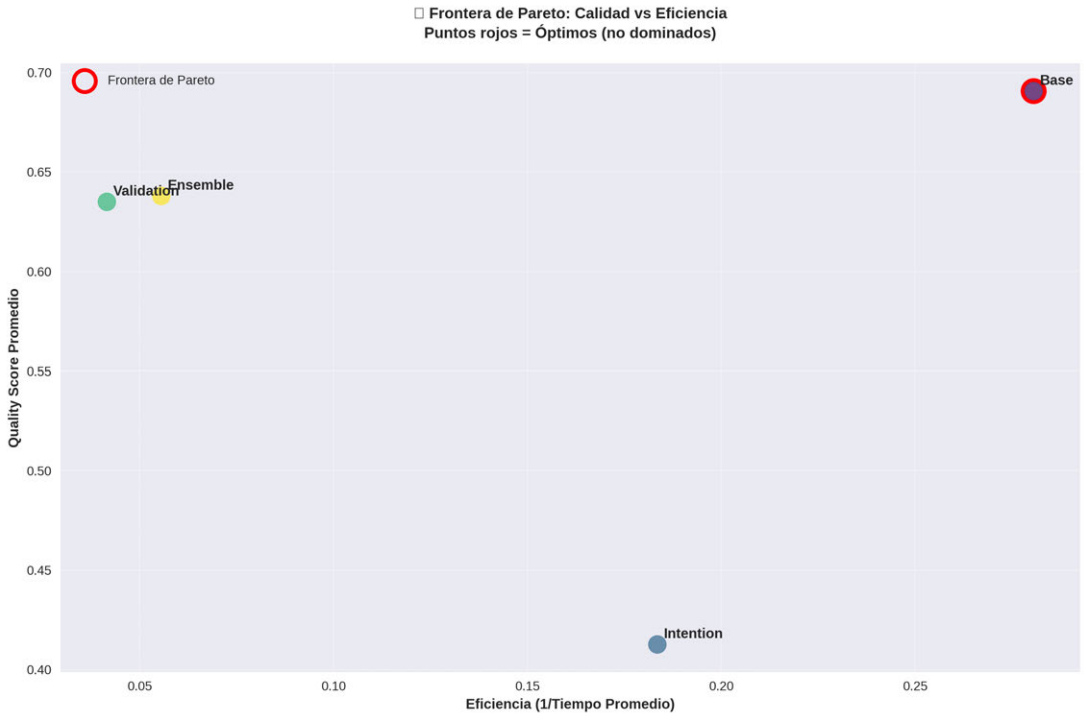

Figura 6.10. Frontera de Pareto que ilustra el compromiso entre la Calidad de los resultados (MRS) y la Eficiencia (inverso de la latencia). Los puntos en la frontera (rojo) representan las soluciones óptimas. El Agente Base es la única arquitectura que se sitúa en esta frontera.

Conclusión Principal de la Discusión

Para el contexto de este estudio, el **Agente Base representa la solución más pragmática y con mejor rendimiento general**. Las arquitecturas más complejas no sólo no ofrecieron las mejoras esperadas, sino que en algunos casos degradaron el rendimiento. Esto subraya la importancia de la validación empírica y la cautela al añadir complejidad a los sistemas LLM.

Patrones Comunes de Fallo: Un hallazgo consistente es la dificultad para manejar errores de **abreviaciones y sustituciones**. Otro patrón es que las consultas de **dificultad "Media"** son las que concentran la mayor parte de los fallos, sugiriendo que la ambigüedad sutil es un problema difícil de resolver.

Impacto de las Capacidades del LLM: Los resultados sugieren que el modelo `Qwen3-30b`, aunque potente, tiene dificultades con la inferencia de "intención" como tarea aislada y la ejecución consistente de ciclos de reflexión que siempre mejoren el resultado. La dependencia del conocimiento geográfico interno del modelo y las capacidades de la API de CartoCiudad siguen siendo factores limitantes.

7. Síntesis y Lecciones Aprendidas

La aportación principal de este proyecto no reside únicamente en las métricas de rendimiento obtenidas, sino en las lecciones extraídas del proceso iterativo de diseño, implementación y evaluación. Lejos de ser un camino lineal hacia lo complejo, el viaje a través de las distintas arquitecturas de agentes ha revelado verdades contraintuitivas sobre la interacción entre los LLM, la orquestación de flujos de trabajo y la naturaleza del problema geoespacial, hallazgos que resuenan con la literatura emergente en el campo.

7.1. La Eficacia de la Simplicidad

La hipótesis inicial que guio este trabajo se alineaba con una intuición común en ingeniería: una mayor complejidad arquitectónica conduciría a un rendimiento superior. Los resultados empíricos, sin embargo, refutaron esta premisa. El **Agente Base** se alzó como la arquitectura óptima, no solo logrando la latencia más baja y el MRS más alto (**Tabla 6.1**), sino también posicionándose como la única solución en la frontera de Pareto (**Figura 6.10**).

Este hallazgo invoca el principio de la **Navaja de Ockham** y se alinea con las directrices prácticas de referentes en la industria. OpenAI, por ejemplo, recomienda explícitamente validar primero las arquitecturas más directas, aconsejando a los desarrolladores que comiencen con sistemas de un solo agente y solo escalen a modelos multi-agente cuando sea estrictamente necesario ("begin with single-agent systems and graduate to multi-agent models only when necessary" [47]). De forma similar, la investigación académica de Bhatt et al. [48] formaliza esta intuición, demostrando que la orquestación de múltiples agentes solo es ventajosa si existen diferencias claras de rendimiento o coste entre ellos. En ausencia de dichas diferencias, la complejidad añadida no justifica la mejora y puede ser contraproducente.

7.2. Análisis de las Arquitecturas Complejas

Calificar de "fracaso" el rendimiento de las arquitecturas avanzadas sería impreciso; más bien, su suboptimidad constituye un resultado de gran valor pedagógico. El análisis de su rendimiento, a la luz de la investigación actual, revela varios factores interconectados que explican por qué la complejidad resultó perjudicial.

7.2.1. La Fragilidad Inherente de las Cadenas de Razonamiento

El "efecto dominó" observado en el prototipo, donde un error en una etapa temprana se propagaba por el resto del flujo, no es una anomalía, sino una manifestación de la fragilidad intrínseca de los LLM. Investigaciones como la de Li et al. [49] demuestran que los modelos son inherentemente sensibles al ruido y propensos al sobreajuste de patrones más que a la abstracción ("inherently sensitive to noise and prone to pattern overfitting"). Esto significa que cada paso en una cadena de razonamiento (CoT) o en un bucle de validación introduce un nuevo punto de fallo estocástico, donde una ligera desviación puede hacer que el agente se desvíe por completo de la tarea. La complejidad, por tanto, no solo aumenta la latencia, sino que incrementa exponencialmente la superficie de ataque para el error.

7.2.2. La Deuda Técnica y el Coste Oculto de la Orquestación

La orquestación manual de agentes, si no se gestiona con un framework robusto, introduce una forma de **deuda técnica**. El código de enlace o *"glue code"* necesario para conectar los distintos nodos del agente tiende a proliferar, creando lo que Sculley et al. [50] denominaron *"pipeline jungles"*. Artículos técnicos recientes [51], [52] confirman que este anti-patrón es común en sistemas LLM y advierten que lo que empieza como una solución rápida puede convertirse en un sistema que resiste la iteración y castiga la experimentación ("a system that resists iteration and punishes experimentation" [52]).

A esto se suma el **coste oculto de la orquestación** en términos de recursos. Un post-mortem de ingeniería de Anthropic [53] revela que sus sistemas multi-agente, aunque más potentes, consumen aproximadamente 15 veces más tokens que una interacción de chat simple. Este sobrecoste masivo de tokens y la latencia asociada deben ser justi-

ficados por un aumento de rendimiento proporcional, algo que en este proyecto no se materializó.

7.2.3. La Paradoja de la Autorreflexión

La inclusión de un ciclo de validación en el Agente de Validación se basaba en una premisa teóricamente sólida. De hecho, estudios como el de Shinn et al. [54] sobre la autorreflexión (*self-reflection*) demuestran que, en condiciones controladas, añadir bucles de verificación mejora el rendimiento de los agentes ("all types of self-reflection improve the performance of LLM agents").

Sin embargo, en la práctica de este proyecto, se manifestó una paradoja: el beneficio teórico de la reflexión fue superado por los costes de la fragilidad y la deuda técnica. El ciclo de validación, ejecutado por un modelo con capacidades limitadas, a menudo introducía más ruido del que corregía, ilustrando que la autorreflexión solo es útil si la capacidad de razonamiento del modelo es lo suficientemente alta como para no cometer nuevos errores durante el proceso de corrección.

En resumen, la bibliografía y la práctica coinciden: aunque las arquitecturas complejas son teóricamente prometedoras, su viabilidad depende de superar los escollos de la fragilidad del razonamiento, la deuda técnica y los costes ocultos, algo que, en el contexto de este trabajo, solo la simplicidad del Agente Base logró eficazmente.

8. Consideraciones para la Puesta en Producción

El éxito de un prototipo en un entorno controlado es solo el primer paso hacia un servicio robusto en producción. La transición exige un análisis holístico que abarca la operativa, la economía, la sostenibilidad y el impacto organizacional y humano. Este capítulo profundiza en estas áreas, integrando buenas prácticas de la industria y análisis estratégicos relevantes para una institución pública en el marco europeo.

8.1. Viabilidad Técnica y Operativa

La puesta en marcha de un sistema basado en LLM se enmarca en la disciplina de **LLMOps**, que comprende los procesos y herramientas para gestionar el ciclo de vida completo de estos modelos en producción [55], [56].

8.1.1. Estrategias de Despliegue y Optimización

Una implementación exitosa requiere una estrategia bien definida que abarca desde los datos hasta la monitorización continua. Las buenas prácticas extraídas de guías técnicas recientes incluyen [57]:

- **Gestión de datos:** Es fundamental utilizar datos de alta calidad, representativos del dominio y correctamente gobernados. Esto implica un proceso riguroso de recolección, limpieza y catalogación.
- **Entrenamiento y ajuste:** Se debe elegir la técnica de ajuste óptima (p. ej., *fine-tuning* o *prompt tuning*) y monitorizar la convergencia del modelo para asegurar su rendimiento.
- **Despliegue e infraestructura:** La elección de la infraestructura, *on-premise*, nube

pública o API externa, es una decisión crítica. La infraestructura local ofrece máximo control y puede ser rentable para cargas de trabajo grandes y previsibles, mientras que la nube aporta flexibilidad. Las APIs gestionadas (p. ej., OpenAI) agilizan el desarrollo inicial pero reducen la personalización y pueden generar costes elevados. Tras el despliegue, es crucial optimizar la latencia y el escalado mediante técnicas como el cacheo de resultados y el balanceo de carga.

- **Monitorización y evaluación:** Se deben definir indicadores clave (precisión, latencia, uso de recursos) y emplear herramientas de observabilidad específicas para LLM que permitan rastrear los prompts y las cadenas de razonamiento para detectar sesgos o errores de forma proactiva [56].

8.1.2. Escalabilidad y Soberanía Digital

En el contexto europeo, la viabilidad técnica está intrínsecamente ligada a la **soberanía digital**. A pesar de liderar en regulación con el AI Act [58], [59], la Unión Europea mantiene una fuerte dependencia tecnológica externa, con una parte sustancial de su infraestructura digital gubernamental dependiendo de proveedores fuera de la UE. Esta situación expone a las instituciones a riesgos estratégicos [60].

Para contrarrestarlo, se han impulsado iniciativas como el plan **EuroStack**, que busca desarrollar una infraestructura de supercomputación y nube europea. La elección entre un modelo de código abierto auto-alojado y una API propietaria se convierte, por tanto, en una decisión estratégica:

- Los **modelos de código abierto** (p. ej., Mistral, Llama) refuerzan la soberanía digital al dar control total sobre los datos y la infraestructura.
- Los **modelos propietarios** (p. ej., OpenAI, Google) ofrecen un rendimiento de vanguardia y facilidad de despliegue, pero a costa de una mayor dependencia y potenciales conflictos con la normativa de protección de datos europea.

Algunos analistas advierten que una regulación estricta podría ralentizar la innovación interna, por lo que se debate un enfoque de "misiones"que alinee la inversión en IA con objetivos estratégicos del bloque, como la transición verde o la cohesión social [60].

8.2. Viabilidad Económica y Ambiental

8.2.1. Análisis del Coste Total de Propiedad (TCO)

La viabilidad económica de un agente LLM va más allá del coste por token. El verdadero **Coste Total de Propiedad (TCO)** depende del modelo de despliegue:

- **Modelos de código abierto:** Aunque no tienen coste de licencia, exigen una fuerte inversión inicial en hardware (GPUs, almacenamiento) y en equipos técnicos especializados para su mantenimiento y operación. Suelen ser más rentables para cargas de trabajo intensas y continuas.
- **Modelos propietarios (vía API):** Operan bajo un modelo de "pago por uso", con bajas barreras de entrada pero costes recurrentes que pueden escalar rápidamente con la demanda, encareciendo la solución a largo plazo.

La decisión final depende de un análisis que pondere el volumen de consultas esperado, la necesidad de personalización, los costes de transferencia de datos y los recursos humanos dedicados.

8.2.2. La Huella de Carbono de la Inferencia

Los LLM tienen un impacto ambiental considerable. Se estima que el entrenamiento de un modelo como GPT-3 produjo tanto CO_2 como **cinco automóviles en toda su vida útil** [61]. Este efecto se magnifica en producción, donde millones de consultas diarias pueden multiplicar por decenas la energía consumida durante el entrenamiento.

En respuesta, el borrador del AI Act de la UE exige a los proveedores de modelos de propósito general (GPAI) documentar su consumo eléctrico [62], [63]. Para mitigar esta huella, se deben adoptar estrategias como el uso de centros de datos "verdes", la optimización algorítmica (p. ej., usando RAG para evitar procesar datos innecesarios) y el aprovechamiento de hardware más eficiente energéticamente [64].

8.3. Viabilidad Humana y Organizacional

8.3.1. El Coste de la Capacitación y el Mantenimiento

Un servicio basado en agentes LLM requiere un nuevo conjunto de habilidades para su mantenimiento. El CNIG necesitaría contar con profesionales capacitados en ingeniería de LLM, orquestación de agentes y monitorización de sistemas de IA, lo que podría implicar un coste en formación o en la contratación de nuevo talento.

8.3.2. Impacto en la Experiencia de Usuario y la Accesibilidad

El beneficio más tangible para el ciudadano es una transformación radical en la **accesibilidad** de los servicios públicos. Los informes de la iniciativa AI-Watch de la Comisión Europea documentan numerosos casos de uso en administraciones europeas donde la IA está eliminando barreras de comunicación [65]-[67]. Destacan:

- **Simplificación del lenguaje burocrático:** Sistemas de IA que procesan textos oficiales y generan versiones en lenguaje claro y comprensible.
- **Asistentes virtuales inclusivos:** Chatbots que guían a los usuarios en trámites públicos, adaptando la interacción a distintos perfiles y necesidades.
- **Mejora de la accesibilidad cognitiva y sensorial:** En España, por ejemplo, se emplea IA para generar versiones en ”lectura fácil”de documentos complejos y convertirlos a voz sintetizada de alta calidad, facilitando el acceso a la información a personas con dificultades de comprensión o discapacidad visual.

En conjunto, estas soluciones impulsan una atención pública más clara, personalizada e inclusiva, permitiendo que todos los ciudadanos ejerzan sus derechos en igualdad de condiciones.

9. Trabajo Futuro y Reflexiones sobre su Impacto

La implementación de un agente funcional como el desarrollado en este trabajo no es un mero avance técnico; es la constatación de un cambio de paradigma inminente. Las siguientes secciones no solo exploran la evolución tecnológica previsible, sino que profundizan en las profundas implicaciones que esta transformación conlleva para la ciencia, la seguridad y la sociedad en su conjunto.

9.1. Evolución hacia Arquitecturas de Agentes Complejos

El futuro no reside tanto en la evolución aislada de los modelos, sino en el desarrollo de **arquitecturas de agentes** que expriman las capacidades emergentes de los modelos de lenguaje que vayan surgiendo. El verdadero potencial se desbloquea al combinar el razonamiento del LLM con marcos de acción complejos.

Este enfoque, conocido como **Mezcla de Agentes (MoA)**[68], se inspira en conceptos clásicos de la computación y el aprendizaje automático, como los métodos *ensemble* (similares a un Random Forest), la inteligencia de enjambre (*swarm intelligence*) o las búsquedas en árbol de Montecarlo que permitieron a sistemas como AlphaGo[69] desarrollar estrategias sobrehumanas, incomprensibles a priori para los expertos. La idea es simple pero poderosa: si un agente puede razonar, un enjambre de agentes puede explorar, debatir y converger en soluciones que un único actor, sea humano o artificial, jamás podría alcanzar.

Las implicaciones de estos "enjambres de agentes" son profundas, especialmente en el ámbito científico, que se basa en el ciclo de prueba y error. Imaginemos agentes que:

- **Aceleran el descubrimiento científico:** En campos como el *drug discovery* o el

diseño de nuevos materiales, enjambres de agentes podrían explorar de forma autónoma millones de combinaciones moleculares o de compuestos, simulando resultados y aprendiendo de ellos, reduciendo drásticamente los tiempos de investigación.

- **Fortalecen la ciberseguridad:** Agentes especializados podrían interactuar con sistemas críticos durante horas, buscando vulnerabilidades con una paciencia y una perspectiva no humana, descubriendo fallos que equipos de expertos humanos no habían detectado.
- **Se programan y mejoran a sí mismos:** La frontera final es la de agentes que no solo utilizan herramientas, sino que las crean, se auto-reparan y se optimizan, generando un ciclo de mejora exponencial.

9.2. Implicaciones Sociales: La Doble Hoja de la Cognición Artificial

Esta tecnología es una herramienta de un poder inmenso y, como tal, representa una hoja de doble filo. La capacidad de un agente para actuar como un "gemelo digital", usurpando capacidades cognitivas humanas, plantea desafíos fundamentales para nuestra sociedad.

El actual paradigma social y laboral se fundamenta en la productividad humana, a menudo medida en términos monetarios. La irrupción de agentes capaces de realizar no solo tareas automatizadas, sino cognitivas y creativas, entra en conflicto directo con este sistema. No se trata de una simple automatización; es la externalización de la cognición, lo que obliga a una reflexión profunda sobre cómo adaptar nuestras estructuras sociales y laborales.

El debate futuro no puede limitarse a la tecnología en sí, sino que debe abarcar cómo construir una sociedad que pueda asimilarla. Se necesita un nuevo contrato social para un mundo donde el trabajo cognitivo no sea la principal fuente de valor del ser humano, permitiendo que lo verdaderamente importante esté cubierto y los humanos puedan centrarse en aquello que, si es que algo queda, sea innatamente nuestro.[70]

9.3. Gobernanza y Seguridad como Contrapeso Necesario

Ante este potencial disruptivo, la necesidad de un marco de gobernanza robusto y de medidas de seguridad proactivas no es una mera formalidad, sino un contrapeso esencial para asegurar que esta ”hoja de doble filo”se utilice para cortar y no para herir.

- **Marco Regulatorio Europeo:** La Ley de IA de la UE (Reglamento UE 2024/1689) es un primer paso indispensable. Clasificar los sistemas por riesgo y exigir transparencia, robustez y supervisión humana son las bases para una IA fiable y alineada con los valores europeos.
- **Mitigación de Sesgos Lingüísticos y Culturales:** Como se ha demostrado, los modelos multilingües son un punto de partida menos sesgado. Sin embargo, la responsabilidad ética exige una auditoría continua y la creación de benchmarks específicos para las lenguas y culturas de una sociedad, garantizando una equidad real en el trato y la representación.
- **Seguridad Proactiva:** Los riesgos identificados por el OWASP Top 10 para LLMs (inyección de prompts, fugas de datos, etc.) deben ser el punto de partida de cualquier estrategia de seguridad. La defensa no puede ser reactiva; debe ser una parte intrínseca del diseño del sistema.

9.4. Conclusión: Hacia un Nuevo Paradigma de Interacción

Este proyecto no solo resuelve un problema técnico de búsqueda geoespacial; sirve como una demostración práctica de un cambio de paradigma fundamental en la interacción entre humanos y máquinas. Durante décadas, el paradigma dominante ha forzado a los humanos a adaptarse a la máquina, aprendiendo a comunicarse a través de interfaces toscas y artificiales. Este trabajo demuestra que la tecnología ha irrumpido con la fuerza suficiente para invertir esa dinámica: ahora, son las máquinas las que se adaptan al ser humano, utilizando el lenguaje natural como interfaz principal.

La sorprendente eficacia de nuestro Agente Base, en su forma más rudimentaria, es la prueba más contundente de este cambio. Si una arquitectura simple puede aportar tanto valor, la proyección de sistemas más sofisticados, mejor entrenados y monitorizados, es inmensa. El futuro está marcado por una tendencia clara: los agentes LLM razonan cada vez mejor, su conocimiento se expande, y el coste de la cognición se abarata, haciendo la inteligencia artificial accesible a una escala sin precedentes.

Este futuro no se basa en intenciones abstractas, sino en la creación de estándares técnicos concretos. Iniciativas como los **Model-Context Protocols (MCPs)**[71] son un ejemplo de cómo se está construyendo el andamiaje para una nueva era digital. Representan un esfuerzo por estandarizar cómo los agentes, independientemente del framework o modelo subyacente, interactúan con los servicios digitales. El futuro ya no pasa por desarrollar una API para que un programador la utilice, sino por diseñar servicios con un MCP para que un agente los consuma de forma nativa. El agente se erige como el intermediario universal para la interacción del usuario, e incluso del desarrollador, con el mundo digital.

Ahí radica el valor último de este trabajo y su visión a futuro: ser una llamada a la acción. Es el momento de dejar de desarrollar tecnología para el ”internet de ayer” y empezar a crear de manera nativa aplicaciones para el ”internet de mañana”; un entorno digital diseñado para ser navegado e interpretado por agentes inteligentes. Equilibrar esta innovación con una gobernanza responsable es el gran desafío y la mayor oportunidad que tenemos por delante.

A. Anexo: Documentación de PyCiudad

A.1. Introducción y Objetivos de PyCiudad

`PyCiudad` es una librería Python desarrollada como parte de este proyecto para facilitar la interacción con la API REST del servicio CartoCiudad del CNIG. La motivación principal fue la ausencia de un wrapper Python oficial que ofreciera una interfaz moderna y robusta.

Los objetivos de diseño de `PyCiudad` fueron:

- **Abstracción y Simplicidad:** Ocultar la complejidad de las llamadas HTTP y el parseo de JSON.
- **Robustez:** Implementar un manejo de errores consistente con excepciones personalizadas.
- **Modelado de Datos:** Utilizar `Pydantic` para definir modelos de datos claros y validados.
- **Facilidad de Integración:** Diseñar la librería para ser integrada como una "herramienta" (*tool*) en sistemas de agentes como los desarrollados con `LangChain/LangGraph`[1], [2].
- **Cobertura Funcional:** Implementar los principales servicios de CartoCiudad (búsqueda, geocodificación y geocodificación inversa).
- **Mantenibilidad:** Estructurar el código de forma modular.

A.2. Arquitectura General de la Librería

`PyCiudad` está organizada en varios módulos Python para asegurar una separación de conceptos clara:

- `cliente.py`: Contiene la clase principal `CartoCiudad`, que actúa como el cliente de la API y encapsula toda la lógica de peticiones y manejo de errores.
- `modelos.py`: Define los modelos de datos `Pydantic` que representan las entidades y respuestas de la API (`Candidato`, `Ubicacion`, `Direccion`, etc.).
- `constantes.py`: Centraliza las URLs de los endpoints, valores por defecto, cabeceras HTTP y definiciones de filtros.
- `utils.py`: Agrupa funciones de utilidad como la validación de coordenadas y la construcción de parámetros.
- `excepciones.py`: Define las clases de excepciones personalizadas que la librería puede lanzar (`APIError`, `PeticionInvalidaError`).

A.3. Descripción Detallada del Cliente `CartoCiud`

Constructor (`__init__`)

Inicializa el cliente con parámetros para el `timeout`, la verificación de SSL y un modo `debug` que activa un logging más detallado de las peticiones.

Método Privado `_realizar_peticion`

Método interno que ejecuta las peticiones HTTP GET usando la librería `requests`. Maneja la construcción de la petición, la gestión de errores de red y de la API (lanzando excepciones personalizadas) y el parseo de la respuesta JSON.

Método buscar_candidatos

Busca candidatos geográficos que coincidan con una consulta textual, soportando múltiples filtros (límite, exclusión de tipos, filtros geográficos). Devuelve una lista de objetos Candidato.

Método geocodificar

Obtiene la información detallada y coordenadas de una entidad a partir de una consulta textual o un identificador. Devuelve un objeto Ubicacion.

Método geocodificacion_inversa

Convierte coordenadas geográficas (longitud y latitud) en la dirección postal más cercana. Devuelve un objeto Direccion.

A.4. Modelos de Datos (modelos.py)

PyCiudad utiliza Pydantic para un modelado y validación de datos robusto.

- TipoEntidad(str, Enum): Enumeración que define los tipos de entidades reconocidos.
- EntidadBase(BaseModel): Clase base con campos comunes (ID, tipo, dirección, municipio, etc.).
- Candidato(EntidadBase): Modelo para los resultados de buscar_candidatos.
- Ubicacion(EntidadBase): Modelo para los resultados de geocodificar.
- Direccion(EntidadBase): Modelo para los resultados de geocodificacion_inversa.

Los modelos incluyen propiedades (@property) para un acceso más intuitivo a los datos y validadores (@field_validator) para asegurar la consistencia de los datos de entrada.

A.5. Utilidades, Constantes y Excepciones

- **`utils.py`:** Contiene helpers como `validar_coordenadas` y `construir_parametros_fi`
- **`constantes.py`:** Centraliza todas las constantes como las URLs de la API y valores por defecto.
- **`excepciones.py`:** Define las excepciones personalizadas `CartoCiudadError`, `APIError` y `PeticionInvalidaError` para un manejo de errores granular.

A.6. Ejemplo de Uso Completo

El siguiente bloque de código muestra un ejemplo completo de cómo utilizar las principales funcionalidades de la librería `PyCiudad`.

Listado A.1. Ejemplo de uso completo de la librería `PyCiudad`.

```
from pyciudad import CartoCiudad, APIError,
   PeticionInvalidaError

# Inicializar cliente en modo debug para ver más detalles
cliente = CartoCiudad(debug=True)

try:
    print("--- Buscando candidatos para 'Museo del Prado,
       Madrid' ---")
    candidatos_prado = cliente.buscar_candidatos(
        consulta="Museo del Prado, Madrid",
        limite=3,
        excluir_tipos=["Codpost"]
    )
    if candidatos_prado:
        for i, candidato in enumerate(candidatos_prado):
            print(f"Candidato {i+1}:")
            print(f"  Nombre: {candidato.address}")
```

```
            print(f"  Tipo: {candidato.type}")
            print(f"  Coords: Lat={candidato.latitud}, Lon={
               candidato.longitud}")

        # Geocodificar el primer candidato
        primer_candidato_id = candidatos_prado[0].id
        if primer_candidato_id:
            print(f"\n--- Geocodificando ID: {
               primer_candidato_id} ---")
            ubicacion_prado = cliente.geocodificar(id_entidad=
               primer_candidato_id, tipo=candidatos_prado[0].
               type)
            print(f"Direccion geocodificada: {ubicacion_prado.
               address}")

    print("\n--- Geocodificacion inversa para Lon: -3.70325,
       Lat: 40.41700 ---")
    direccion_sol = cliente.geocodificacion_inversa(longitud
       =-3.70325, latitud=40.41700)
    print(f"Via: {direccion_sol.via}, N: {direccion_sol.numero}
       ")
    print(f"Municipio: {direccion_sol.municipio}")

except PeticionInvalidaError as e:
    print(f"\nERROR de Peticion Invalida: {e}")
except APIError as e:
    print(f"\nERROR de API: {e}")
except Exception as e:
    print(f"\nUn ERROR INESPERADO ocurrio: {e}")
```

Referencias

[1] LangChain, Inc. «LangChain Documentation: Introduction.» [Online; accessed 31-May-2024]. (2024), dirección: https://python.langchain.com/docs/introduction/ (visitado 31-05-2024).

[2] LangChain, Inc. «LangGraph Documentation: Why LangGraph?» [Online; accessed 22-June-2025]. (2025), dirección: https://langchain-ai.github.io/langgraph/concepts/why-langgraph/ (visitado 22-06-2025).

[3] Wikipedia contributors. «Bag-of-words model — Wikipedia, The Free Encyclopedia.» [Online; accessed 31-May-2024]. (2024), dirección: https://en.wikipedia.org/wiki/Bag-of-words_model (visitado 31-05-2024).

[4] D. Krishnan. «Search with TF-IDF algorithm in Information Retrieval systems.» [Online; accessed 31-May-2024]. (dic. de 2023), dirección: https://medium.com/@dhanyakrishnan8109/search-with-tf-idf-algorithm-in-information-retrieval-systems-d18e7d1e25c3 (visitado 31-05-2024).

[5] J. C. S. de Souza, J. L. A. D. M. Junior, G. G. H. de Oliveira, T. A. Almeida y D. D. de Campos, *BM25s: A Family of Simple and Effective Learning to Rank Functions*, 2024. arXiv: 2407.03618 [cs.IR].

[6] D. Weldon. «What Is Fuzzy Matching?» [Online; accessed 22-June-2025]. (mayo de 2024), dirección: https://www.devx.com/terms/fuzzy-matching/ (visitado 22-06-2025).

[7] R. J. Tiggemann, A. S y A. N S, «A Comparative Study of Search Result Clustering Algorithms,» *International Journal of Engineering Research and Applications (IJERA)*, vol. 6, n.º 7, págs. 20-22, jul. de 2016.

[8] B. Popper. «An intuitive introduction to text embeddings.» [Online; accessed 22-June-2025]. (nov. de 2023), dirección: https://stackoverflow.blog/2023/11/09/an-intuitive-introduction-to-text-embeddings/ (visitado 22-06-2025).

[9] A. Chaudhary. «One Hot Encoding Explained.» [Online; accessed 22-June-2025]. (ago. de 2020), dirección: https://medium.com/@heyamit10/one-hot-encoding-explained-0b0130ccd78e (visitado 22-06-2025).

[10] ZILLIZ. «From Words to Vectors: Understanding Word2Vec in Natural Language Processing (NLP).» [Online; accessed 22-June-2025]. (abr. de 2024), dirección: https://zilliz.com/glossary/word2vec (visitado 22-06-2025).

[11] ZILLIZ. «GloVe: A Machine Learning Algorithm for Decoding Word Connections.» [Online; accessed 22-June-2025]. (abr. de 2024), dirección: https://zilliz.com/glossary/glove (visitado 22-06-2025).

[12] M. E. Peters, M. Neumann, M. Iyyer et al., «Deep contextualized word representations,» en *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, jun. de 2018, págs. 2227-2237. dirección: https://aclanthology.org/N18-1202.

[13] A. Vaswani, N. Shazeer, N. Parmar et al., «Attention Is All You Need,» en *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, I. Guyon, U. V. Luxburg, S. Bengio et al., eds., Curran Associates, Inc., 2017, págs. 5998-6008. dirección: https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845Paper.pdf.

[14] J. Devlin, M.-W. Chang, K. Lee y K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,» en *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, jun. de 2019, págs. 4171-4186. dirección: https://aclanthology.org/N19-1423.

[15] Wikipedia contributors. «Vector database — Wikipedia, The Free Encyclopedia.» [Online; accessed 22-June-2025]. (2025), dirección: https://en.wikipedia.org/wiki/Vector_database (visitado 22-06-2025).

[16] B. Soni. «The Potential of Approximate Nearest Neighbors (ANN) in High-Dimensional Spaces.» [Online; accessed 22-June-2025]. (jul. de 2023), dirección: https://medium.com/@brijesh_soni/the-potential-of-approximate-nearest-neighbors-ann-in-high-dimensional-spaces-579567e4f1a7 (visitado 22-06-2025).

[17] OpenAI. «Models - OpenAI API.» [Online; accessed 22-June-2025]. (2025), dirección: https://platform.openai.com/docs/models (visitado 22-06-2025).

[18] Google. «Gemini models - Google for Developers.» [Online; accessed 22-June-2025]. (2025), dirección: https://ai.google.dev/gemini-api/docs/models (visitado 22-06-2025).

[19] J. Bai, S. Bai, Y.-H. Sung y T.-J. Weng, *Qwen3 Technical Report*, 2024. arXiv: 2406.04821 [cs.CL].

[20] Anthropic. «Model overview - Anthropic Docs.» [Online; accessed 22-June-2025]. (2025), dirección: https://docs.anthropic.com/en/docs/about-claude/models/overview (visitado 22-06-2025).

[21] J. Wei, Y. Tay, R. Bommasani et al., «Emergent Abilities of Large Language Models,» *Transactions on Machine Learning Research*, oct. de 2022, ISSN: 2835-8856. dirección: https://openreview.net/forum?id=yzkSU5zdwD.

[22] T. B. Brown, B. Mann, N. Ryder et al., «Language Models are Few-Shot Learners,» en *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan y H. Lin, eds., Curran Associates, Inc., 2020, págs. 1877-1901. dirección: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[23] A. Brohan, N. Brown, J. Carbajal et al., *RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control*, 2023. arXiv: 2307.15818 [cs.RO].

[24] K. Se y A. Vert. «What is test-time compute and how to scale it?» [Online; accessed 23-June-2025]. (feb. de 2025), dirección: https://huggingface.co/blog/Kseniase/testtimecompute (visitado 23-06-2025).

[25] J. Wei, X. Wang, D. Schuurmans et al., *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*, 2022. arXiv: 2201.11903 [cs.CL].

[26] Y. Zheng, Y. He, Q. Li et al., *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*, 2025. arXiv: 2501.12948 [cs.CL].

[27] J. Su, Y.-a. Lu, S. Pan, B. Wen e Y. Liu, *RoFormer: Enhanced Transformer with Rotary Position Embedding*, 2021. arXiv: 2104.09864 [cs.CL].

[28] N. Shazeer, *Fast Transformer Decoding: One Write-Head is All You Need*, 2019. arXiv: 1911.02150 [cs.LG].

[29] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón y S. Sanghai, *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*, 2023. arXiv: 2305.13245 [cs.LG].

[30] Hugging Face. «Transformers Documentation.» [En línea; consultado el 23-junio-2025]. (2025), dirección: https://huggingface.co/docs/transformers/es/index (visitado 23-06-2025).

[31] Ollama Inc. «Ollama.» [Online; accessed 23-June-2025]. (2025), dirección: https://ollama.com/ (visitado 23-06-2025).

[32] The vLLM Team. «vLLM Documentation.» [Online; accessed 23-June-2025]. (2025), dirección: https://docs.vllm.ai/en/latest/ (visitado 23-06-2025).

[33] AUTOR. «Unstructured Data Isn't Just for Embeddings: Hidden Structure can Improve RAG.» [Online; accessed 23-June-2025]. (AÑO), dirección: https://ai.gopubby.com/unstructured-data-isnt-just-for-embeddings-hidden-structure-can-improve-rag-167f8d66e61c (visitado 23-06-2025).

[34] AUTOR. «Supervised Fine-Tuning (SFT) with Large Language Models.» [Online; accessed 23-June-2025]. (2024), dirección: https://medium.com/data-science/supervised-fine-tuning-sft-with-large-language-models-0c7d66a26788 (visitado 23-06-2025).

[35] OpenAI. «Aligning language models to follow instructions.» [Online; accessed 23-June-2025]. (ene. de 2022), dirección: https://openai.com/index/instruction-following/ (visitado 23-06-2025).

[36] Y. Chen, S. Qian, H. Tang et al., *LongLoRA: Efficient Fine-tuning of Long-Context Large Language Models*, 2023. arXiv: 2309.12307 [cs.CL].

[37] LangChain. «LangMem SDK for agent long-term memory.» [Online; accessed 23-June-2025]. (feb. de 2025), dirección: https://blog.langchain.dev/langmem-sdk-launch/ (visitado 23-06-2025).

[38] Mem0 AI, *Mem0: The Memory Layer for Personalized AI*, https://github.com/mem0ai/mem0, [Online; accessed 23-June-2025], 2025. (visitado 23-06-2025).

[39] S. Yao, J. Zhao, D. Yu et al., *ReAct: Synergizing Reasoning and Acting in Language Models*, 2022. arXiv: 2210.03629 [cs.CL].

[40] IBM. «What is tree of thoughts? A new prompting technique for LLMs.» [Online; accessed 23-June-2025]. (2025), dirección: https://www.ibm.com/think/topics/tree-of-thoughts (visitado 23-06-2025).

[41] D. Zhou, N. Schärli, L. Hou et al., *Least-to-Most Prompting Enables Complex Reasoning in Large Language Models*, 2022. arXiv: 2205.10625 [cs.CL].

[42] LlamaIndex. «LlamaIndex Documentation.» [Online; accessed 23-June-2025]. (2025), dirección: https://docs.llamaindex.ai/en/stable/ (visitado 23-06-2025).

[43] Hugging Face. «smol-agents Documentation.» [Online; accessed 23-June-2025]. (2025), dirección: https://huggingface.co/docs/smolagents/index (visitado 23-06-2025).

[44] CrewAI. «CrewAI.» [Online; accessed 23-June-2025]. (2025), dirección: https://www.crewai.com/ (visitado 23-06-2025).

[45] LangChain. «LangSmith.» [Online; accessed 23-June-2025]. (2025), dirección: https://www.langchain.com/langsmith (visitado 23-06-2025).

[46] ngrok. «ngrok | API Gateway, Kubernetes Ingress, Webhook Gateway.» [Online; accessed 23-June-2025]. (2025), dirección: https://ngrok.com/ (visitado 23-06-2025).

[47] OpenAI, *Practical Guide for LLM Agent Development*, Internal practical guide and industry recommendations, Cited and summarized in technical blogs such as BABL.AI, 2025.

[48] S. Bhatt, K. Valmeekam, M. Hoffman, S. Shah, C. Baral e Y. Yang, *When Should We Orchestrate Multiple Agents?* arXiv preprint arXiv:2503.13577, 2025. arXiv: 2503.13577 [cs.AI]. dirección: https://arxiv.org/abs/2503.13577.

[49] Z. Li, N. Li, B. Wang, Z. Li y H. Yao, *Patterns Over Principles: The Fragility of Inductive Reasoning in LLMs under Noisy Observations*, arXiv preprint arXiv:2502.16169, 2025. arXiv: 2502.16169 [cs.CL]. dirección: https://arxiv.org/abs/2502.16169.

[50] D. Sculley, G. Holt, D. Golovin et al., «Hidden technical debt in machine learning systems,» en *Advances in neural information processing systems 27 (NIPS 2014)*, 2014. dirección: https://papers.nips.cc/paper/2014/file/8b49b8f1825b6e4e7b6d2b
Paper.pdf.

[51] Various Authors, *Hidden Technical Debt in LLM Systems*, Medium, Represents a common topic in technical articles on the platform discussing anti-patterns in LLM system design, 2024.

[52] Hypermode, *The Hidden Cost of DIY Integrations*, Hypermode Engineering Blog, Analysis from a technical industry blog post on agent orchestration, 2025.

[53] Anthropic, *Engineering Post-mortem on Multi-Agent System Token Costs*, Anthropic Engineering Blog, Internal analysis on the high token consumption of multi-agent systems compared to single-agent interactions, 2025.

[54] N. Shinn, B. Labash y A. Gopinath, *Self-Reflection in LLM Agents: Effects on Problem-Solving Performance*, arXiv preprint arXiv:2405.06682, 2024. arXiv: 2405.06682 [cs.AI]. dirección: https://arxiv.org/abs/2405.06682.

[55] G. Liu. «A Guide to Effective LLMOps Practices.» (jul. de 2023), dirección: https://gyliu513.medium.com/a-guide-to-effective-llmops-practices-7177419176c2 (visitado 26-10-2024).

[56] Weights & Biases, «LLMOps explained: Managing large language model operations,» Weights & Biases, inf. téc., 2024. dirección: https://wandb.ai/onlineinference/llm-evaluation/reports/LLMOps-explained-Managing-large-language-model-operations--VmlldzoxMjM2MDM4MQ (visitado 26-10-2024).

[57] IJSA Technologies, «A Comprehensive Guide to Deploying Large Language Models in Production Environments,» *International Journal of Scientific Advancements*, vol. 1, n.º 1, pág. 2412, 2025. dirección: https://www.ijsat.org/papers/2025/1/2412.pdf.

[58] Comisión Europea. «EU AI Act.» (2024), dirección: https://digital-strategy.ec.europa.eu/en/factpages/ai-act (visitado 26-10-2024).

[59] White & Case. «AI Watch: Global regulatory tracker – European Union.» (abr. de 2025), dirección: https://www.whitecase.com/insight-our-thinking/ai-watch-global-regulatory-tracker-european-union (visitado 26-10-2024).

[60] Carnegie Endowment for International Peace, «The EU's AI Power Play: Between Deregulation and Innovation,» *Carnegie Endowment Research*, mayo de 2025. dirección: https://carnegieendowment.org/research/2025/05/the-eus-ai-power-play-between-deregulation-and-innovation?lang=en.

[61] AlgorithmWatch. «Sustainable AI: A Contradiction in Terms?» (2024), dirección: https://algorithmwatch.org/en/sustainable-ai-explained/ (visitado 26-10-2024).

[62] White & Case. «Energy efficiency requirements under the EU AI Act.» (2024), dirección: https://www.whitecase.com/insight-alert/energy-efficiency-requirements-under-eu-ai-act (visitado 26-10-2024).

[63] C. Alder et al., «AI, Climate, and Transparency: Operationalizing and Improving the AI Act,» *arXiv preprint arXiv:2409.07471*, sep. de 2024. arXiv: 2409.07471 [cs.CY].

[64] A. Ebert, P. Hacker et al., «AI, Climate, and Regulation: From Data Centers to the AI Act,» *arXiv preprint arXiv:2410.06681*, oct. de 2024. arXiv: 2410.06681 [cs.CY].

[65] Joint Research Centre (JRC), «AI Watch – European landscape on the use of Artificial Intelligence by the Public Sector,» European Commission, inf. téc., feb. de 2023, JRC129301. dirección: https://interoperable-europe.ec.europa.eu/sites/default/files/custom-page/attachment/2023-02/JRC129301_01_AI_watch.pdf.

[66] G. Misuraca y C. van Noordt, «AI Watch – Artificial Intelligence in public services,» JRC, European Commission, inf. téc., 2020, JRC118474. dirección: https://ec.europa.eu/jrc/en/publication/eur-scientific-and-technical-research-reports/ai-watch-artificial-intelligence-public-services.

[67] Joint Research Centre (JRC), «AI Watch. Beyond pilots: Sustainable implementation of AI in public services,» European Commission, inf. téc., 2021. DOI: 10.2760/440212.

[68] J. Wang, J. Wang, B. Athiwaratkun, C. Zhang y J. Zou, *Mixture-of-Agents Enhances Large Language Model Capabilities*, 2024. arXiv: 2406.04692 [cs.CL]. dirección: https://arxiv.org/abs/2406.04692.

[69] J. Hui, «Monte Carlo Tree Search (MCTS) in AlphaGo Zero,» *Medium*, dic. de 2017. dirección: https://jonathan-hui.medium.com/monte-carlo-tree-search-mcts-in-alphago-zero-8a403588276a (visitado 25-06-2024).

[70] D. Amodei, *Machines of Loving Grace*, Accessed on 2024-06-25, oct. de 2024. dirección: https://www.darioamodei.com/essay/machines-of-loving-grace.

[71] Model-Context Protocol Contributors, *Model-Context Protocol: Introduction*, https://modelcontextprotocol.io/introduction, [Online; accessed 24-June-2025], 2024.

Índice de términos

Glosario

API Interfaz de Programación de Aplicaciones (*Application Programming Interface*), un conjunto de reglas, métodos y protocolos que permiten la comunicación entre diferentes aplicaciones o componentes de software.. 1, 5–7, 19–22, 26–31, 33, 39, 56, 61, 67, 68

CNIG Centro Nacional de Información Geográfica, organismo autónomo dependiente del Instituto Geográfico Nacional (IGN) responsable del servicio CartoCiudad. 1, 5, 6, 63, 68

cuantización a 4 bits Del inglés *4-bit Quantization*. Técnica de compresión de modelos de aprendizaje profundo que consiste en reducir la precisión numérica de los pesos del modelo a solo 4 bits. Esto disminuye drásticamente el tamaño del modelo en memoria y el coste computacional, permitiendo su ejecución en hardware con recursos limitados.. 27

GPU Unidad de Procesamiento Gráfico, un procesador especializado para renderizar gráficos y realizar cálculos en paralelo.. 25, 27, 49

Ground Truth Término inglés que se refiere al conjunto de datos de referencia, considerados como la verdad absoluta o la fuente de información correcta, contra el cual se evalúa el rendimiento de un modelo o sistema. Se traduce como ”verdad terreno” o ”datos de referencia”.. 39, 40, 42, 48

LLM Un modelo de lenguaje con un gran número de parámetros, entrenado con volúmenes masivos de datos textuales.. 5–7, 9, 11, 14, 16–23, 25, 26, 28–30, 33, 35, 37, 40–43, 46, 56–58, 60–63

Mapear Del inglés *to map*. En el contexto de este trabajo, se refiere a la acción de establecer una correspondencia entre los datos extraídos de la consulta de un usuario y los campos específicos esperados por una API o una base de datos.. 2, 9, 10

Navaja de Ockham Principio metodológico y filosófico que postula que, en igualdad de condiciones, la explicación más simple suele ser la más probable. En ingeniería, se traduce en preferir la solución más sencilla que cumpla con los requisitos.. 57

parseo Del inglés *to parse*. Proceso de analizar sintácticamente una cadena de texto o datos para descomponerla en sus componentes estructurales. En este trabajo, se refiere a la acción de interpretar la consulta del usuario para extraer sus partes significativas (tipo de vía, nombre, localidad, etc.).. 10, 27, 68, 69

Procesamiento del Lenguaje Natural Campo de la inteligencia artificial que se ocupa de la interacción entre las computadoras y el lenguaje humano.. 14

RAG Del inglés *Retrieval-Augmented Generation*. Es una arquitectura de software que combina un modelo de lenguaje generativo con un sistema de recuperación de información externo, para que las respuestas generadas se basen en un corpus de conocimiento específico y actualizado.. 16, 17, 22, 23

VRAM Memoria de acceso aleatorio utilizada para almacenar los datos de imagen y vídeo que procesa una GPU.. 25, 26