Universidad de Valladolid

Escuela Técnica Superior de Ingenieros de Telecomunicación

Trabajo de Fin de Grado

Grado en Ingeniería de Tecnologías de Telecomunicación

Título

Generación de tareas de aprendizaje ubicuo con apoyo de herramientas de Inteligencia Artificial Generativa

Autor:
Luis Francisco Sánchez Turrión
Tutores:
Dr.Juan Ignacio Asensio Pérez
Dr.Miguel Luis Bote Lorenzo

Valladolid, septiembre 2025

Título:	**Generación de tareas de aprendizaje ubicuo con apoyo de herramientas de Inteligencia Artificial Generativa**
Autor:	**Luis Francisco Sánchez Turrión**
Tutores:	**Dr.Juan Ignacio Asensio Pérez** **Dr.Miguel Luis Bote Lorenzo**
Departamento:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

Tribunal

Presidente:	**Dr.Eduardo Gómez Sánchez**
Vocal:	**Dr.Yannis Dimitriadis**
Secretario:	**Dr.Guillermo Vega Gorgojo**

Fecha:	**15 de septiembre de 2025**
Calificación:	

Agradecimientos

Quiero agradecer al grupo de investigación GSIC/EMIC por abrirme las puertas y acompañarme en este camino. Su orientación y confianza han sido decisivas para culminar este trabajo. A mis tutores, por su paciencia y claridad en cada etapa. Y, por supuesto, a mi familia y amistades, cuyo apoyo diario me ayuda a seguir adelante.

Resumen

Múltiples organizaciones publican datos abiertos —especialmente de patrimonio cultural— que no están listos para su uso directo en educación. Trabajos previos del grupo GSIC/EMIC abordaron esta brecha con Casual Learn, una aplicación distribuida (servidor+cliente Android) que, apoyándose en un almacén de triplas RDF y consultas SPARQL, ofrece tareas educativas basadas en plantillas validadas por docentes para aprendizaje ubicuo, con experiencias reales en más de cien estudiantes de 4.º ESO y 1.º de Bachillerato. Aunque eficaz, este enfoque depende del refinamiento previo de plantillas y de la disponibilidad de LOD, lo que limita la expresividad y la portabilidad a dominios distintos del patrimonio.

Este Trabajo Fin de Grado propone una alternativa generativa para tareas de aprendizaje ubicuo: una herramienta para generar tareas de aprendizaje ubicuo con modelos de lenguaje y Generación Aumentada por Recuperación (RAG) a partir de Wikipedia y materiales aportados por el docente. La solución se expone mediante API REST en un backend monolítico que procesa localmente la información para proteger la privacidad, integrando *embeddings* (*SentenceTransformers*), almacenamiento vectorial (*ChromaDB*) y ejecución local de LLM (*Ollama*). Las salidas se fuerzan en JSON estructurado, facilitando su consumo por clientes móviles o flujos de autoría.

En la comparación cualitativa frente a tareas tipo Casual Learn, el sistema genera ítems más ricos y contextualizados (distractores plausibles, preguntas abiertas interpretativas y actividades situadas), sin depender de plantillas rígidas. En la evaluación cuantitativa, las latencias promedio para el *hardware* disponible en el grupo GSIC/EMIC fueron 42–43 s para generación simple (con o sin RAG) y 365 s para generación compleja, identificando la inferencia del LLM como principal cuello de botella y acotando la concurrencia práctica a 2–5 usuarios en el hardware evaluado. En conjunto, el trabajo complementa y trasciende el enfoque basado en LOD al habilitar generación bajo demanda y portabilidad entre dominios, sentando bases para su integración futura con metadatos LOD, técnicas de caché y optimización de inferencia.

Palabras clave—aprendizaje ubicuo; inteligencia artificial generativa; RAG; Wikipedia; LangChain; ChromaDB; Ollama; Linked Open Data.

Abstract

Open data—particularly in the cultural heritage domain—are abundant yet rarely ready for direct educational use. Prior work by GSIC/EMIC addressed this gap with Casual Learn, a distributed application (server + Android client) backed by an RDF triple store and SPARQL that delivers teacher-validated, template-based ubiquitous learning tasks. Deployed with over one hundred high-school students, this approach proved effective but inherently relies on pre-authored templates and LOD availability, which constrains expressiveness and portability beyond heritage scenarios.

This Bachelor's Thesis proposes a generative alternative for ubiquitous learning tasks: a tool for ubiquitous learning task generation using Large Language Models with Retrieval-Augmented Generation (RAG) from Wikipedia and teacher-provided materials. The system is exposed via a REST API on a privacy-preserving monolithic backend, combining SentenceTransformers embeddings, ChromaDB vector storage, and local LLM inference (Ollama). Outputs are enforced as structured JSON to streamline integration with mobile clients and authoring pipelines.

In a qualitative comparison against Casual Learn-style tasks, our system produces richer, more contextualized items (plausible MCQ distractors, interpretive open questions, and situated activities) without rigid templates. The quantitative evaluation shows average latencies for the hardware available in the GSIC/EMIC group were 42–43 s for simple (with/without RAG) and 365 s for complex generations, identifying LLM inference as the main bottleneck and limiting practical concurrency to 2–5 users on the tested hardware. Overall, this work complements and goes beyond LOD-based pipelines by enabling on-demand generation and cross-domain portability, paving the way for future integration with LOD metadata, caching strategies, and inference optimizations.

Keywords— ubiquitous learning; generative AI; RAG; Wikipedia; LangChain; ChromaDB; Ollama; Linked Open Data.

Índice general

Índice de Figuras

Índice de Tablas

Índice de Códigos

Capítulo 1

Introducción

A lo largo de la historia, ciertos avances tecnológicos han supuesto verdaderos puntos de inflexión que han transformado profundamente no solo la economía, sino también las estructuras sociales y culturales. Un ejemplo es el de la mecanización de la agricultura durante la primera mitad del siglo XX. Si bien esta innovación permitió automatizar tareas antes realizadas manualmente, aliviando en gran medida la carga física del trabajo agrícola, sus implicaciones trascendieron el ámbito laboral. La creciente eficiencia productiva redujo la necesidad de mano de obra en el campo, lo que provocó un éxodo masivo hacia las ciudades en busca de nuevas oportunidades [Sierra, 2019]. Este proceso no solo transformó la economía agraria, sino que modificó profundamente los modos de vida tradicionales, aceleró la urbanización y sentó las bases para el surgimiento de una sociedad industrial y predominantemente urbana.

En la actualidad, nos encontramos inmersos en una revolución de naturaleza comparable: la irrupción de la inteligencia artificial (IA). De manera análoga a como las máquinas agrícolas liberaron al ser humano del trabajo físico más exigente, la IA está comenzando a asumir tareas cognitivas complejas, tales como la conducción autónoma, la traducción automática o la asistencia robótica, entre otros. Todo indica que, en las próximas décadas, la automatización basada en IA transformará de forma significativa no solo el mercado laboral [Hui et al., 2024; Moreno-Izquierdo, Torres Penalva et al., 2025], sino también los modos en que aprendemos [Fernández, 2023], nos comunicamos [Herrera-Ortiz et al., 2024] y tomamos decisiones [Jiménez Cardona, 2023; Peñalver-Higuera y Isea-Argüelles, 2024].

Esta transformación está siendo posible gracias a una serie de avances tecnológicos interrelacionados. En primer lugar, la evolución del *hardware*, destacando el uso de unidades de procesamiento gráfico (*Graphics Processing Units*, GPUs) y, más recientemente, de unidades de procesamiento tensorial (Tensor Processing Units, TPUs), diseñadas específicamente para acelerar operaciones matriciales propias del aprendizaje profundo [Chellapilla et al., 2006]. A ello se suma la expansión de la computación en la nube (*cloud computing*), que proporciona acceso flexible y escalable a infraestructuras de alto rendimiento, así como la disponibilidad masiva de grandes volúmenes de datos tanto estructurados como no estructurados, que son esenciales para el entrenamiento efectivo de modelos a gran escala. Además, el desarrollo de marcos de programación de código abierto, como TensorFlow o

PyTorch, ha democratizado el acceso a herramientas avanzadas de modelado. Por último, la introducción de nuevas arquitecturas, en particular los modelos basados en transformadores [Vaswani et al., 2017], ha revolucionado el tratamiento de secuencias, permitiendo capturar dependencias a largo plazo y mejorar significativamente el rendimiento en tareas como el procesamiento del lenguaje natural, la visión por computador y la generación automática de contenidos.

El ámbito educativo no permanece ajeno a los cambios que está propiciando la actual transformación tecnológica impulsada por la IA. La capacidad de los modelos de IA generativa para producir contenidos de manera autónoma y contextualizada abre nuevas posibilidades para el diseño de experiencias de aprendizaje más flexibles, personalizadas y adaptadas al entorno del estudiante [Eager y Brunton, 2023]. El presente Trabajo Fin de Grado se enmarca precisamente en esta línea de innovación y lleva por título **«Generación de tareas de aprendizaje ubicuo con apoyo de herramientas de Inteligencia Artificial Generativa»**. Para comprender el alcance del estudio, resulta pertinente desglosar los principales conceptos incluidos en su título.

Por **aprendizaje ubicuo** se entiende un enfoque educativo en el que el aprendizaje puede ocurrir en cualquier momento y lugar, apoyado por tecnologías móviles, sensores, geolocalización y conectividad permanente. Este paradigma busca extender el aprendizaje más allá del aula, integrándolo en la vida diaria del estudiante de forma continua, contextual y personalizada [Muñoz-Cristóbal et al., 2014]. Esta modalidad rompe con las limitaciones espaciales y temporales de la educación tradicional, facilitando experiencias de aprendizaje más flexibles y adaptativas. La Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura (UNESCO) reconoce el potencial de la tecnología para transformar los entornos educativos, destacando que, según sus defensores, esta permite ampliar las posibilidades del aprendizaje ubicuo y facilitar el desarrollo de nuevas formas de aprendizaje y competencias [UNESCO, 2023b].

Las **tareas de aprendizaje** son actividades diseñadas con fines pedagógicos concretos, que pueden incluir desde ejercicios tradicionales hasta experiencias interactivas, investigaciones en el entorno o resolución de problemas reales [Ruiz-Calleja et al., 2021]. En el marco del aprendizaje ubicuo, estas tareas se vinculan con el contexto físico o digital inmediato del estudiante, haciéndolas más significativas y motivadoras. Este enfoque conecta con la idea de tareas de aprendizaje como actividades centradas en el aprendiz y orientadas a la acción, que promueven una participación activa y situada, especialmente valiosa cuando se vinculan con entornos reales o simulados mediante tecnologías inteligentes.

Por último, la **inteligencia artificial generativa** hace referencia a modelos capaces de producir contenido nuevo a partir de datos existentes o instrucciones, como ocurre con los grandes modelos de lenguaje [Banh y Strobel, 2023]. Estos modelos permiten generar textos, imágenes, código o incluso actividades educativas, ajustándose a parámetros como el nivel del estudiante, su localización, sus intereses o sus interacciones previas. En este contexto, la UNESCO [UNESCO, 2023a] ha señalado el notable incremento del uso de la IA en el ámbito educativo, con especial énfasis en los modelos de lenguaje de gran escala

(*Large Language Models*, LLM), que utilizan técnicas como la generación aumentada por recuperación (Retrieval-Augmented Generation, RAG) para obtener información fiable y generar respuestas actualizadas y contextualizadas. La UNESCO subraya tanto el potencial transformador de estas tecnologías como los desafíos que plantean, especialmente en lo relativo a una gestión ética y responsable, así como a la necesidad de formación adecuada para el profesorado y el estudiantado, con el fin de garantizar un uso beneficioso y equitativo [UNESCO, 2023a].

La convergencia de estos tres elementos —aprendizaje ubicuo, tareas educativas y modelos generativos— constituye el eje central de este proyecto. Para comprender la contribución específica de este trabajo, es fundamental analizar primero el estado actual de la generación de tareas para el aprendizaje ubicuo. Uno de los enfoques usados en aprendizaje ubicuo se basa en la explotación de *Linked Open Data* (LOD), una metodología que combina grandes bases de conocimiento con plantillas pedagógicas.

Un ejemplo paradigmático es el sistema *Casual Learn* [Ruiz-Calleja et al., 2021, 2023], que ha demostrado la viabilidad de esta aproximación. Este enfoque basado en LOD y plantillas ha probado ser notablemente eficaz para generar a gran escala tareas geolocalizadas y contextualizadas. Utilizando bases de conocimiento abiertas como DBpedia, el sistema extrae información estructurada sobre puntos de interés (como monumentos históricos) y aplica plantillas predefinidas para transformar esos datos en enunciados de tareas. Este método permite una producción masiva —generando miles de actividades— y garantiza, gracias a la colaboración de expertos, una alta pertinencia pedagógica, alineando las tareas con los currículos escolares y distintos niveles cognitivos [Ruiz-Calleja et al., 2022].

Sin embargo, y a pesar de sus evidentes fortalezas, este modelo presenta una limitación fundamental que motiva el presente estudio: su dependencia de plantillas predefinidas y rígidas. Esta rigidez estructural impone serias restricciones a la escalabilidad pedagógica. Por un lado, la variedad lingüística y estilística de las tareas es muy escasa, ya que todas las actividades generadas a partir de una misma plantilla siguen un patrón casi idéntico. Por otro lado, la creatividad y la complejidad pedagógica quedan acotadas por el diseño inicial de la plantilla, haciendo imposible generar dinámicamente nuevas tipologías de tareas o adaptarlas a matices contextuales no previstos.

Es precisamente en la superación de esta rigidez donde la inteligencia artificial generativa, y en particular los *Large Language Models* (LLM), emerge como una alternativa disruptiva. A diferencia de los sistemas que se limitan a "rellenar" una estructura fija, los LLM pueden generar texto original, variado y contextualmente coherente a partir de instrucciones complejas, ofreciendo una flexibilidad inalcanzable para los sistemas anteriores. La principal motivación de este proyecto es, por tanto, explorar la hipótesis de que los LLM pueden superar las limitaciones de las plantillas, no solo para aumentar la variedad y el atractivo de las tareas, sino también para introducir nuevas dimensiones de personalización y sofisticación pedagógica.

En consecuencia, este Trabajo de Fin de Grado, se centra en diseñar y evaluar un

sistema que utiliza LLM para trascender el paradigma de las plantillas. El objetivo es demostrar que es posible avanzar hacia un modelo de creación de contenido educativo más dinámico, diverso y adaptativo, sentando así las bases para una nueva generación de herramientas de aprendizaje ubicuo.

1.1 Objetivos

El objetivo principal de este Trabajo Fin de Grado es diseñar y implementar una herramienta multidominio capaz de generar tareas en contextos de aprendizaje ubicuo, así como de ofrecer retroalimentación automatizada basada en IA generativa a las respuestas del estudiantado.

Objetivos específicos que se plantean:

- **Seleccionar un modelo de lenguaje adecuado:** Buscar información y hacer pruebas para encontrar un **LLM** que sea adecuado para la generación de las tareas.
- **Diseñar e implementar una API:** Proporcionar un servicio de generación de tareas a través de herramientas de inteligencia artificial generativa de forma contextualizada.
- **Incorporar la generación de distintos tipos de tareas relevantes desde el punto de vista pedagógico:** Con el fin de adaptarse a los requerimientos individuales, el sistema genera una variedad de tareas. Este enfoque busca la pertinencia pedagógica de cada actividad.
- **Evaluar la eficacia de la herramienta:** Comprobamos que la herramienta realmente está aumentando la calidad de las tareas generadas, se compararán las tareas generadas por modelos comerciales de código cerrado y también una comparativa del uso del modelo sin contexto, modelo con RAG y modelo con todos los datos en el contexto.

1.2 Metodología

El desarrollo del presente Trabajo Fin de Grado se ha estructurado siguiendo los principios de la metodología ágil **Scrum** [Schwaber y Sutherland, 2013], adaptada al contexto de un proyecto individual. Scrum es un marco de trabajo ágil para la gestión y desarrollo de productos que se basa en ciclos iterativos e incrementales denominados *sprints*. Esta metodología permite organizar el trabajo de forma flexible, fomentando la planificación continua, la revisión frecuente de avances y la mejora progresiva del producto.

Durante el proyecto se llevaron a cabo 14 *sprints* de dos semanas de duración. En cada uno de ellos se establecieron objetivos específicos relacionados con el desarrollo de funcionalidades concretas de la herramienta. Al inicio de cada *sprint* se realizó una planificación (*Sprint Planning*), donde se definieron las tareas a abordar y los entregables

correspondientes. Al final de cada ciclo se efectuó una revisión (*Sprint Review*) del trabajo realizado y una reflexión crítica (*Sprint Retrospective*) sobre los aspectos que podían mejorarse en la siguiente iteración.

Los roles de Scrum se adaptaron a la estructura del proyecto de la siguiente manera:

- Los tutores académicos asumieron el rol de **Propietarios del Producto** (*Product Owners*), orientando la dirección del desarrollo, priorizando funcionalidades y validando los entregables parciales.
- Algunos compañeros del departamento, con experiencia técnica, actuaron como **Maestros de Scrum** (*Scrum Masters*), ofreciéndome apoyo puntual para resolver dudas o superar dificultades técnicas.
- El rol de **Equipo de Desarrollo** fue desempeñado por mí como autor del TFG, encargándome de implementar los requisitos funcionales establecidos en cada *sprint*.

Este enfoque me permitió avanzar de forma estructurada, recibir retroalimentación periódica y ajustar el desarrollo según las necesidades emergentes del proyecto.

1.3 Estructura del documento

A lo largo de este documento se proporciona una visión coherente y progresiva del proceso completo, desde los antecedentes hasta las conclusiones y anexos, con el fin de facilitar al lector una comprensión integral y ordenada del proyecto.

En primer lugar, se explora el **Estado del arte**, abordando la generación de tareas de aprendizaje con datos abiertos y la inteligencia artificial generativa, con énfasis en los Modelos de Lenguaje de Gran Tamaño y en las técnicas clave que permiten aprovechar, adaptar y perfeccionar sus capacidades, como la Generación Aumentada por Recuperación y el Ajuste Fino (Fine-Tuning). Este apartado establece las bases conceptuales y metodológicas esenciales para entender la generación dinámica y precisa de contenidos educativos mediante IA.

Seguidamente, el capítulo de **Análisis** expone las necesidades educativas actuales y resalta las limitaciones de las soluciones tradicionales ante la creciente demanda de materiales didácticos personalizados. Aquí se justifica la relevancia del proyecto, definiendo claramente los objetivos y requisitos del sistema propuesto para solucionar las carencias identificadas.

A continuación, en el capítulo de **Diseño**, se describe en detalle la arquitectura lógica del sistema, especificando cómo se coordinan sus distintos componentes. Se explica cómo tecnologías como las bases de datos vectoriales se usan para hacer RAG para recuperar información relevante, y generan salidas estructuradas en formato JSON que alimentan el generador de tareas. Además, se detalla el diseño del motor de creación de ejercicios, el corrector automático y la integración en un flujo de trabajo unificado.

Posteriormente, en el capítulo **Implementación**, se abordan los aspectos prácticos del desarrollo del proyecto, incluyendo la organización del código, la selección de librerías y *frameworks*, configuración del entorno, instalación de dependencias y ejemplos concretos de ejecución. Se destacan también las decisiones tecnológicas adoptadas y los protocolos de prueba utilizados para garantizar la fiabilidad, reproducibilidad y escalabilidad del sistema.

El capítulo **Análisis de resultados** evalúa la eficacia de las tareas generadas mediante una comparación con métodos convencionales. Se presentan métricas relativas a la calidad pedagógica y al rendimiento computacional, detallando tiempos de respuesta y estudios de casos prácticos, además de discutir las principales limitaciones detectadas, sugiriendo posibles vías de mejora.

En el capítulo de **Conclusiones y líneas de trabajo futuro** se resumen las contribuciones más significativas del proyecto, destacando el valor añadido que aporta al ámbito educativo, la originalidad en la integración de RAG y Fine-Tuning, y la capacidad para adaptar contenidos a diferentes perfiles de estudiantes. Asimismo, se proponen líneas futuras de trabajo, tales como la incorporación de retroalimentación en tiempo real, la expansión hacia otras áreas del conocimiento o la integración con nuevos modelos multimodales.

Finalmente, los **Anexos** incluyen documentación detallada de la API desarrollada y el código empleado, permitiendo al lector disponer de los recursos necesarios para comprender profundamente cada componente y replicar el sistema en su propio entorno.

Capítulo 2

Estado del arte

La IA ha avanzado rápidamente, transformando el campo de la IA mediante modelos como la serie *Generative Pre-trained Transformer* (GPT)[Brown et al., 2020]. Gracias a redes neuronales de gran tamaño, nuevos algoritmos de Aprendizaje Automático y extensos conjuntos de datos de entrenamiento, estos modelos destacan por su capacidad para generar texto coherente con características humanas. Su accesibilidad y los marcos de código abierto han democratizado el uso de LLM, facilitando su integración en sectores como los chatbots, la salud y la educación. En este estado del arte se presentan los principios de funcionamiento y las metodologías actuales empleadas en el ámbito de la IA. Además, se va a tratar la metodología anterior en la generación de tareas usando LOD.

2.1 Introducción

El presente Trabajo de Fin de Grado, se enmarca en un contexto donde converge el aprendizaje ubicuo (*u-learning*) y los modelos de lenguaje basados en inteligencia artificial generativa (IA Generativa). El aprendizaje ubicuo se define como aquella modalidad educativa que permite el acceso a conocimiento en cualquier lugar y momento, gracias al uso de tecnologías móviles y ubicuas, promoviendo entornos de aprendizaje distribuidos en tiempo y espacio [Gallego-Lema, 2016]. Este enfoque trasciende las aulas convencionales, posibilitando la interacción espontánea del estudiante con su entorno y favoreciendo el aprendizaje sin que éste sea siempre consciente de estar ocurriendo.

Los dispositivos móviles, ordenadores y otros medios tecnológicos no solo facilitan el acceso a recursos modernos, sino que también integran medios tradicionales en formato digital (por ejemplo, libros de Historia del Arte), ampliando así los escenarios del aprendizaje [Gallego-Lema, 2016]. El modelo se caracteriza por la permanencia, accesibilidad, inmediatez, interactividad y adaptabilidad del contenido educativo, lo que permite diseñar experiencias altamente personalizadas y ecológicamente pertinentes.

Gracias al empleo de Datos Enlazados (Linked Data) y de la Web Semántica, la información sobre tareas educativas puede estructurarse para ser accesible tanto para humanos como para máquinas [Berners-Lee, 2006; Gallego-Lema, 2016]. El grupo GSIC/EMIC ya

avanzó en esta dirección con la creación de un repositorio de triplas sobre patrimonio cultural, publicadas en formato abierto y enlazado, lo que permite su reutilización en sistemas de aprendizaje ubicuo [Ruiz-Calleja et al., 2021] y facilita su integración en plataformas como Casual Learn mediante consultas SPARQL.

Por otra parte, el auge de la IA Generativa, especialmente los modelos de lenguaje enormes (LLM), ofrece una nueva dimensión para la generación automatizada de tareas de aprendizaje. Estos modelos, entrenados con enormes volúmenes de texto, son capaces de generar contenido educativo adaptado, como explicaciones, preguntas y sugerencias pedagógicas. Su uso en la educación permite personalizar rutas de aprendizaje, generar feedback en tiempo real y ampliar el alcance del aprendizaje ubicuo.

No obstante, su integración requiere un diseño cuidadoso. Es fundamental garantizar la precisión, evitar sesgos y fomentar la alfabetización en IA entre estudiantes y docentes, así como asegurar que los modelos sirvan como herramientas de apoyo y no sustitutos del docente. Además, la orientación mediante gráficos de conocimiento (como los repositorios de triplas) y la tecnología RAG (Retrieval-Augmented Generation) puede ayudar a que la IA genere tareas más fundamentadas y contextualizadas en patrimonio cultural.

El presente estado del arte tiene como objetivo establecer el marco teórico y tecnológico en el que se inscribe este trabajo. Se comenzará analizando los enfoques previos para la generación de tareas de aprendizaje ubicuo, con especial atención a los sistemas basados en Datos Abiertos, que si bien son efectivos, dependen de plantillas predefinidas. A continuación, el capítulo se adentrará en el paradigma de la Inteligencia Artificial Generativa, explorando los fundamentos de los LLM. Se describirán sus arquitecturas y, de manera crucial, las técnicas clave para su adaptación a contextos específicos. Este análisis permitirá justificar la transición desde los métodos tradicionales hacia un enfoque más dinámico y flexible, sentando las bases para el sistema desarrollado en este proyecto.

2.2 Generación de tareas de aprendizaje con Datos Abiertos

La Web Clásica, o Web de documentos, está diseñada para ser interpretada por humanos, quienes navegan a través de hipervínculos entre páginas. Este modelo presenta serias dificultades para que las máquinas procesen la información de manera autónoma, ya que carece de una estructura semántica comprensible para ellas [García Zarza, 2021]. Para resolver esta limitación, se propuso la Web Semántica, una extensión de la web donde la información posee una estructura bien definida, permitiendo una cooperación más eficiente entre personas y máquinas [García Zarza, 2021].

El pilar de la Web Semántica son los Datos Abiertos Enlazados (Linked Open Data – LOD), un conjunto de buenas prácticas para publicar y conectar datos estructurados en la Web [García Zarza, 2021]. Estas prácticas se apoyan en tecnologías estándar como RDF (Resource Description Framework) para modelar la información en tripletas

(sujeto–predicado–objeto) y SPARQL como lenguaje de consulta, creando un grafo de conocimiento global y legible por máquinas [García Zarza, 2021]. Este ecosistema de datos interconectados abre nuevas oportunidades para la creación automática de contenido, incluyendo las tareas de aprendizaje.

La creación manual de tareas de aprendizaje, especialmente aquellas contextualizadas para escenarios informales, es un proceso tedioso, costoso y difícil de escalar [Ruiz-Calleja et al., 2021]. Esto ha impulsado la investigación en métodos de generación automática. Las primeras aproximaciones se centraron en el uso de texto no estructurado u ontologías de dominio. Sin embargo, el primero suele generar preguntas superficiales, mientras que el segundo depende de la costosa creación y mantenimiento de ontologías, restringiendo las tareas a un dominio específico y, a menudo, limitándose a evaluar conocimiento factual [Ruiz-Calleja et al., 2021].

El uso de LOD se presenta como una solución para superar estas limitaciones, al explotar la vasta cantidad de conocimiento ya disponible en la Web. Existen estudios pioneros que utilizan DBpedia para generar automáticamente preguntas o ejercicios para entornos educativos Ruiz-Calleja et al., 2021.

A pesar de estos avances, el estado del arte presenta carencias significativas que el trabajo de [Ruiz-Calleja et al., 2021] busca solventar:

- **Dependencia de una única fuente de datos:** La mayoría de los trabajos se basan casi exclusivamente en una fuente, principalmente DBpedia, sin explotar el potencial de integrar múltiples fuentes para obtener descripciones más ricas.
- **Enfoque en conocimiento factual:** Las tareas generadas suelen ser preguntas de bajo nivel cognitivo (como preguntas de opción múltiple) que evalúan la memorización. Rara vez promueven el pensamiento de orden superior como el análisis, la comparación o la creación.
- **Falta de contextualización física:** Las tareas generadas no se vinculan con el contexto físico del estudiante, lo que dificulta su aplicación en escenarios de aprendizaje ubicuo o móvil.

El repositorio *Casual Learn SPARQL* es el resultado de un trabajo previo que aborda directamente estas limitaciones [García Zarza, 2021]. Se trata de un punto de acceso SPARQL que contiene tareas educativas sobre el patrimonio cultural de Castilla y León, generadas de forma semiautomática. Este enfoque representa un avance en el estado del arte por varias razones:

- **Integración de múltiples fuentes:** Para obtener una base de conocimiento rica, el contenido fue generado combinando información de distintas fuentes de datos abiertos, como DBpedia, Wikidata y datos del gobierno regional .
- **Tareas de mayor nivel cognitivo:** Utilizando plantillas diseñadas por expertos educadores, se generaron tareas que van más allá de la simple memorización, solicitando al usuario comparar edificios de estilos diferentes o reflexionar sobre la concentración de un estilo arquitectónico en una zona.

- **Contextualización explícita:** La ontología de Casual Learn está diseñada para la contextualización. Cada tarea (`clo:task`) está vinculada explícitamente a un espacio físico (`clo:physicalSpace`) a través de la propiedad `clp:hasContext`. Estos espacios físicos están geoetiquetados con coordenadas de latitud y longitud, permitiendo que una aplicación recomiende tareas relevantes según la ubicación del usuario.

2.3 Fundamentos de los modelos LLM

A principios de la década de 2010, las Redes Neuronales Recurrentes (RNNs) demostraron su eficacia en el procesamiento secuencial, al capturar dependencias contextuales y generar texto coherente [Banh y Strobel, 2023]. Sin embargo, presentaban dificultades para manejar dependencias a largo plazo, así como problemas de desvanecimiento o explosión del gradiente y lentitud en el procesamiento. Los modelos basados en transformadores revolucionaron la generación de texto al introducir mecanismos de atención capaces de capturar el contexto a lo largo de secuencias completas de manera simultánea [Vaswani et al., 2017]. Modelos como GPT superaron a las RNNs clásicas gracias a su paralelización, mejor manejo de dependencias a largo plazo y una modelización lingüística mejorada mediante atención automática multilateral que es una técnica utilizada para enfocar selectivamente ciertas partes de la entrada al procesar datos. Las capacidades de los LLM han crecido exponencialmente debido a los avances en arquitecturas basada en transformadores, el uso de conjuntos masivos de datos textuales y el aumento del poder computacional [Devlin et al., 2018]. Estos desarrollos, junto con el incremento del número de parámetros, permiten a los LLM destacar en tareas complejas de procesamiento del lenguaje natural.

Los LLM han encontrado aplicación en numerosos ámbitos gracias a su versatilidad para comprender y generar lenguaje natural. En el procesamiento del lenguaje, destacan en tareas como generación de texto coherente, traducción automática, resumen de documentos y análisis de sentimientos. En entornos conversacionales, potencian chatbots y asistentes virtuales capaces de mantener diálogos fluidos y contextualmente relevantes.

En educación, permiten experiencias de aprendizaje personalizado mediante tutores virtuales que responden a consultas en tiempo real y adaptan explicaciones a las necesidades individuales de cada alumno [García-Méndez et al., 2025; Giannakos et al., 2024]. Asimismo, en el desarrollo de *software*, herramientas como asistentes de codificación sugieren fragmentos de código, detectan errores y optimizan flujos de trabajo [Moradi et al., 2025].

2.3.1 Arquitecturas de Modelo y Objetivos de Preentrenamiento

Los LLM se preentrenan típicamente mediante aprendizaje autosupervisado, sin necesidad de etiquetas manuales, es decir, no se necesitan anotadores a diferencia del aprendizaje supervisado. La selección de los objetivos de preentrenamiento define sus capacidades y depende de la arquitectura diseñada [Gholami y Omar, 2023]. En los modelos basados

en transformadores, la arquitectura puede incorporar un codificador, un decodificador o ambos, cada uno con ventajas y limitaciones propias.

Modelos Codificador–Decodificador

También llamados secuencia-a-secuencia (del inglés *sequence-to-sequence*,*seq2seq*), combinan un codificador que procesa la secuencia de entrada con un decodificador que genera la salida. Su preentrenamiento suele alternar entre objetivos de predicción de palabras ocultas en medio de frases (*masked language modeling*, MLM)y la reconstrucción de frases parciales o codificadas, favoreciendo tanto la comprensión contextual como la generación precisa. Ejemplos destacados incluyen BART [M. Lewis et al., 2019] y T5 [Raffel et al., 2020]. Aunque sobresalen en tareas como traducción y resumen, escalar estas arquitecturas a miles de millones de parámetros plantea retos de eficiencia y memoria.

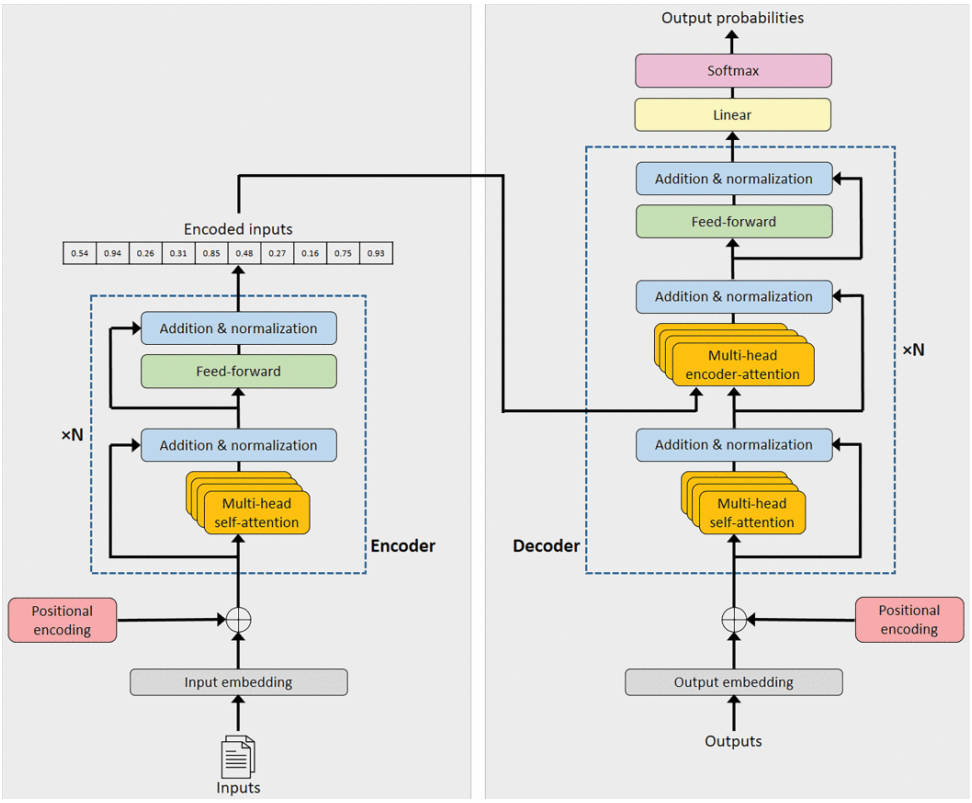

Figura 2.1: Esquema de un modelo codificador–decodificador: atención multicabeza, normalización y generación autorregresiva [Vaswani et al., 2017].

Modelos Solo Codificador

Los modelos de solo codificador toman texto como entrada y a la salida un vector de alta dimensión del texto también conocido como *embeddings* del texto. Los casos de uso de estos modelos son la clasificación de texto, búsqueda semántica y el análisis de texto. BERT [Devlin et al., 2018] y All-MiniLM son referentes en esta categoría.

Modelos Solo Decodificador

Los modelos decodificadores se centran en generar un nuevo *token* para completar la secuencia basándose en los tokens anteriores, solo un token cada vez. Este conjunto de tokens que pueden manejar prestándole atención a todos es lo que se llama ventana de contexto [Yang et al., 2019]. Son líderes en generación de texto fluido y coherente, con ejemplos como GPT [Brown et al., 2020], Deepseek [et al., 2024], Gemma [Gemma Team, 2024] y Llama [Grattafiori et al., 2024] . Sin embargo, demandan grandes volúmenes de datos para entrenamiento y recursos computacionales, y pueden mostrar incoherencias o repeticiones en secuencias extensas que superan el tamaño de la ventana de contexto.

Las arquitecturas de codificador–decodificador, solo codificador y solo decodificador se diferencian fundamentalmente en su enfoque y propósito: la primera está concebida para transformar una entrada en una salida distinta, procesando profundamente la información entrante antes de reconstruirla; la solo codificador se centra exclusivamente en analizar y extraer representaciones del contenido de entrada sin generar nueva información, mientras que la solo decodificador está diseñada para generar secuencias de salida de forma autónoma, basándose únicamente en estímulos previos, continuando así la producción de texto de manera coherente y fluida. Por eso es que en la actualidad son los más usados.

2.3.2 Técnica RAG: *Retrieval-Augmented Generation*

Aunque los LLM ofrecen un rendimiento excepcional en multitud de tareas, su dependencia exclusiva de los datos de entrenamiento puede inducir *alucinaciones*, es decir, respuestas incorrectas presentadas con elevada confianza. Para mitigar este problema y mejorar la exactitud factual, se integran los LLM con motores de recuperación de información externa, dando lugar a la técnica de RAG [P. Lewis et al., 2020].

Ante una consulta, un sistema RAG realiza primero la búsqueda de documentos relevantes en fuentes como wikis, bases de datos o repositorios web. A continuación, incorpora los fragmentos recuperados como contexto adicional al *prompt* del LLM, lo que incrementa la fiabilidad y actualidad de la respuesta generada. De este modo, RAG actúa como puente entre el aprendizaje estadístico y la necesidad de acceder a hechos en tiempo real.

Función de las bases de datos vectoriales en RAG

Las bases de datos vectoriales son esenciales en RAG, pues permiten indexar y recuperar documentos de forma eficiente [Han et al., 2023]. Cada fragmento de texto se representa como un vector de alta dimensión (*embedding*) que captura su significado semántico. Al transformar la consulta en un vector, el sistema identifica rápidamente los vectores más afines y extrae los documentos más relevantes. Este mecanismo optimiza la precisión del contexto proporcionado al LLM y reduce el coste computacional de la búsqueda.

Ventajas

El enfoque RAG amplía las capacidades de los LLM [P. Lewis et al., 2020]:

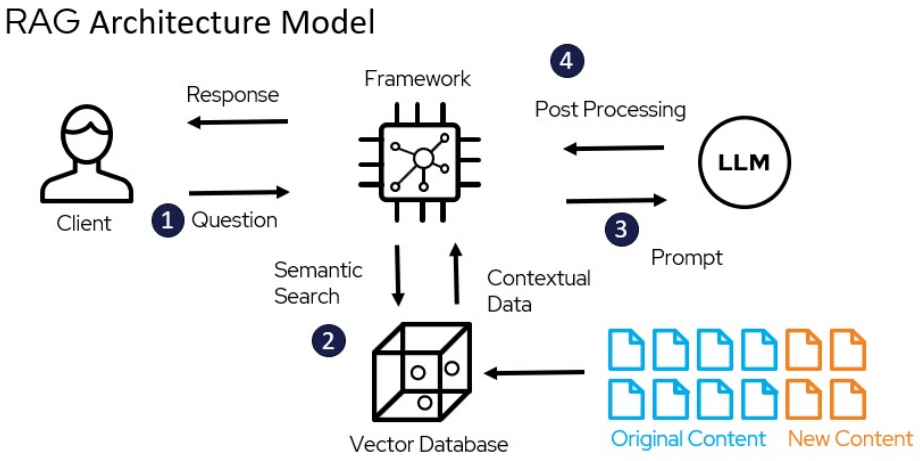

Figura 2.2: Esquema general de un sistema RAG: recuperación de documentos, fusión con el *prompt* y generación asistida por LLM [Gandhi, 2024]

- **Mejorar la veracidad**: incorpora información actualizada, esencial para consultas sensibles al tiempo.
- **Acceder a conocimientos especializados**: va más allá de los datos de entrenamiento originales, útil para especializados dominios.
- **Reducir alucinaciones**: disminuye la probabilidad de respuestas infundadas.
- **Versatilidad**: El mecanismo de RAG se puede implementar sin necesidad de un entrenamiento previo del modelo.

Inconvenientes

- **Calidad de recuperación**: la pertinencia de los documentos recuperados condiciona directamente la validez de la respuesta.
- **Latencia**: la fase de búsqueda puede introducir retrasos en el tiempo de respuesta.
- **Complejidad de infraestructura**: diseñar y escalar un sistema RAG eficiente requiere recursos y arquitecturas robustas.

2.3.3 Técnica *Fine-tuning*

El ajuste fino adapta modelos preentrenados a dominios o tareas concretas mediante entrenamiento supervisado sobre conjuntos de datos específicos. De este modo, el modelo interioriza vocabulario, estilo y requisitos particulares de la tarea, optimizando su desempeño en aplicaciones como análisis de sentimientos, generación de resúmenes o interacciones especializadas [Mahabadi et al., 2021]. Un ajuste fino eficaz equilibra el conocimiento general adquirido en el preentrenamiento con las demandas de la tarea objetivo.

El ajuste fino basado en instrucciones emplea ejemplos de pares *prompt –respuesta* que guían el comportamiento del modelo hacia formatos y contenidos deseados [Zhang et al.,

2024]. Existen bases de datos especializadas para entrenar modelos en tareas específicas. Entrenar un modelo exclusivamente en una tarea puede provocar un sobreentrenamiento [Kemker et al., 2018], donde el conocimiento previo se sobrescribe y disminuye el rendimiento en otras tareas. No obstante, resulta imprescindible evaluar la capacidad de generalización más allá del conjunto de entrenamiento proporcionado, ya que un sobreentrenamiento puede hacer que pierda su capacidad de generar texto coherente.

Para mitigar este efecto, el ajuste fino multitarea entrena simultáneamente en diversos objetivos, compartiendo representaciones y preservando la versatilidad del modelo. Métodos como FLAN–T5 [Longpre et al., 2023] y FLAN–PaLM [Chia et al., 2023] ejemplifican este enfoque, aunque requieren grandes volúmenes de datos y recursos computacionales.

Ajuste fino eficiente en parámetros (PEFT)

Las técnicas de ajuste fino eficiente en parámetros (PEFT) reducen el coste y el impacto sobre los pesos preentrenados al modificar solo un subconjunto reducido de parámetros o añadir componentes ligeros [Fu et al., 2023]. De este modo, se conserva la mayoría de la estructura original y se evitan efectos adversos sobre las capacidades generales del LLM.

Métodos selectivos Actualizan únicamente ciertos parámetros, como capas o sesgos concretos, lo que aporta eficiencia computacional, si bien su adaptabilidad a tareas muy diferentes puede resultar limitada.

Una técnica selectiva es *Bias-only Fine-Tuning* (BitFit) que solo actualiza los términos del sesgo de las capas del modelo, dejando los pesos principales congelados en el entrenamiento [Zaken et al., 2021].

Métodos aditivos Los métodos aditivos PEFT incorporan parámetros o capas entrenables adicionales a un modelo preentrenado, sin modificar la estructura central ni los parámetros originales del modelo. Esta categoría incluye dos enfoques principales:

- *Adaptadores*: capas insertadas en la arquitectura que ajustan el modelo a tareas específicas de forma modular.
- *Prompts suaves*: vectores de embedding entrenables añadidos al inicio del *prompt*, permitiendo una adaptación rápida mediante *prompt tuning*.

Aunque ofrecen gran flexibilidad, en escenarios complejos pueden requerir numerosos parámetros adicionales, lo que impacta en almacenamiento y despliegue. La técnica más destacada es Low-Rank Adaptation (LoRA) que se basa en añadir una matriz extra a los pesos del modelo y entrenar solo la nueva matriz [Hu et al., 2021, Dettmers et al., 2023].

2.3.4 Generación de tareas educativas con LLM

Los LLM se han propuesto recientemente como herramientas para generar automáticamente tareas educativas adaptadas al contexto del aprendiz (por ejemplo, [Cui y Sachan,

2023]). Estos modelos, entrenados con enormes cantidades de datos textuales, poseen un conocimiento amplio y la capacidad de seguir instrucciones para producir contenidos complejos. En el ámbito educativo, estas capacidades implican que un LLM con un *prompt* adecuado puede generar enunciados de tareas, preguntas o actividades didácticas relevantes para distintos contenidos y niveles, e incluso personalizarlas según las necesidades del alumno o su entorno.

Ahora bien, para aprovechar plenamente los LLM en la generación de tareas educativas es necesario asegurar la relevancia y corrección de las actividades propuestas. Diversos trabajos recientes exploran cómo mejorar estos modelos en dicho cometido. Por una parte, se ha propuesto integrar mecanismos de recuperación de información externa durante la generación: [Li et al., 2025] plantean un modelo de RAG que consulta fuentes de conocimiento abiertas para obtener datos actualizados o específicos del dominio educativo antes de crear la tarea. Este enfoque ha mostrado mejoras en la precisión factual y la contextualización de las tareas generadas, ya que el modelo puede basar sus enunciados en contenidos verídicos y pertinentes al currículo en lugar de solo en su conocimiento entrenado.

Por otra parte, se investigan técnicas de ajuste fino especializadas para dotar a los LLM de mayores habilidades en la planificación y elaboración de tareas. En particular, [Zeng et al., 2023] introducen **AgentTuning**, un método de ajuste fino orientado a que los LLM actúen como agentes capaces de descomponer y resolver tareas complejas mediante interacciones o herramientas externas. Al entrenar al modelo con trayectorias de solución de tareas paso a paso, este adquiere la capacidad de generar actividades educativas más estructuradas y multietapa sin perder sus habilidades generales de lenguaje.

Para evaluar que los LLM generen instrucciones y tareas adecuadas, se han desarrollado marcos de evaluación exhaustivos de su atención prestada al *prompt*. Trabajos como **InstructEval** [Chia et al., 2023] y **EvalVerse** [Kim et al., 2024] proponen evaluar a los modelos desde múltiples perspectivas – resolución de problemas, calidad de la redacción, alineamiento con valores humanos, etc. – verificando que el modelo siga fielmente las instrucciones educativas y produzca enunciados conformes a los objetivos pedagógicos.

Las razones por las cuales se ha optado por desarrollar un sistema propio de generación de tareas educativas, en lugar de utilizar una solución existente, son principalmente dos. Primero, buscamos una herramienta que se adapte específicamente a las necesidades pedagógicas identificadas. Segundo, queremos asegurar la privacidad y confidencialidad de los datos, evitando depender de servidores de terceros para la obtención y almacenamiento de información.

2.4 Conclusiones

La integración del aprendizaje ubicuo con tecnologías avanzadas de IA generativa, especialmente mediante el uso de modelos de lenguaje y técnicas como RAG, representa una oportunidad significativa para personalizar y enriquecer las experiencias educativas relacionadas con el patrimonio cultural. Los LLM destacan por su capacidad de generar

automáticamente tareas educativas adaptadas al contexto y las necesidades específicas del estudiante, apoyándose en fuentes externas para garantizar precisión factual y pertinencia curricular.

El enfoque basado en Datos Abiertos Enlazados (LOD), probado en sistemas como Casual Learn, ya ha demostrado su eficacia generando tareas geolocalizadas y contextualizadas de forma masiva. Sin embargo, aunque este enfoque proporciona contenidos educativos relevantes, su escalabilidad está limitada por la dependencia de plantillas predefinidas y la necesidad de actualizaciones manuales o semiautomáticas cuando cambian los datos subyacentes.

En contraste, los LLM permiten una generación dinámica y personalizada de tareas, capaz de integrar nuevas fuentes de información y adaptarse en tiempo real a diversos contextos educativos. La combinación de ambas tecnologías, utilizando técnicas como RAG junto con bases de datos vectoriales y repositorios semánticos como Casual Learn, permite solventar las limitaciones individuales de cada método. Sin embargo, esta integración plantea desafíos técnicos significativos en términos de latencia, calidad de recuperación de información y la complejidad de la infraestructura necesaria.

Por ello, el desarrollo del sistema planteado en este trabajo deberá contemplar no solo la integración efectiva de ambas tecnologías, sino también la necesidad de simplificar el proceso subyacente de obtención y generación de tareas educativas. De este modo, se asegurará una experiencia fluida y efectiva que permita a estudiantes y docentes beneficiarse plenamente de las potencialidades de estas herramientas avanzadas sin enfrentarse a barreras tecnológicas innecesarias.

Capítulo 3

Análisis

En este capítulo se realiza el análisis de los requisitos del sistema propuesto, con el objetivo de determinar las funcionalidades y características que debe cumplir para satisfacer las necesidades de los usuarios finales. En el contexto de este TFG, los usuarios principales son estudiantes y docentes que utilizarán la herramienta para generar y evaluar tareas de aprendizaje ubicuo. A partir de un ejemplo de uso, se identifican los requisitos funcionales y no funcionales del sistema. Además, se describen los casos de uso más representativos y se discute la arquitectura general del sistema, justificando la elección del modelo que finalmente se implementará.

3.1 Introducción

Siguiendo la metodología ágil **Scrum** presentada en la Sección 1.2, que permite un desarrollo iterativo y adaptativo del producto, en este capítulo se aborda la fase de análisis del proyecto. Esta fase tiene como propósito identificar los requisitos que el sistema debe cumplir para alcanzar los objetivos planteados. Dado que no existe un cliente tradicional, los requisitos se han definido en colaboración con los tutores del TFG y expertos del grupo de investigación GSIC/EMIC, basándose en su experiencia en tecnologías educativas y en las necesidades de los usuarios finales: estudiantes de los últimos cursos de Enseñanza Secundaria Obligatoria y Bachillerato, así como docentes interesados en integrar el aprendizaje ubicuo en sus clases.

El sistema propuesto busca aprovechar herramientas de inteligencia artificial generativa (como modelos de lenguaje de gran tamaño) y datos de Wikipedia y información proporcionada por los docentes para generar tareas contextualizadas y ofrecer retroalimentación automática. Además, en esta sección se reflexiona sobre la arquitectura general del sistema, influenciada por los requisitos identificados y las limitaciones técnicas del entorno de desarrollo, como el uso de frameworks.

3.2 Ejemplo de Uso del Sistema

Para contextualizar los requisitos, se describe un escenario representativo de cómo un usuario interactuaría con el sistema. Supongamos que un docente de Historia del Arte desea generar tareas personalizadas para sus estudiantes, aprovechando el patrimonio cultural de Valladolid. El docente accede al sistema a través de una interfaz web o móvil, selecciona el tema «Arquitectura Herreriana» y especifica que las tareas deben estar vinculadas a monumentos locales, como la Catedral de Valladolid. El sistema, utilizando Inteligencia Artificial Generativa y datos extraídos de Wikipedia o de la información proporcionada, genera automáticamente un conjunto de tareas que indica el docente:

- 5 preguntas de opción múltiple sobre las características herrerianas de la catedral.
- Una actividad que requiere tomar una fotografía de un elemento arquitectónico específico.

Posteriormente, un estudiante accede a una de estas tareas desde su dispositivo móvil mientras visita la catedral. Tras responder a una pregunta de texto libre, el sistema evalúa automáticamente su respuesta utilizando un modelo de lenguaje y le proporciona retroalimentación inmediata, indicando si su análisis es correcto y sugiriendo aspectos mejorables. Este ejemplo ilustra la necesidad de un sistema que integre generación de tareas contextualizadas y evaluación automatizada, alineándose con los objetivos del proyecto descritos en la Sección 1.1.

3.3 Identificación de Requisitos

Estos requisitos se heredan de los descritos en trabajos anteriores de Casual Learn [García Zarza, 2021; Ruiz-Calleja et al., 2022], de donde se han identificado los requisitos funcionales y no funcionales que el sistema debe cumplir. Estos requisitos buscan asegurar que la herramienta sea útil tanto para docentes como para estudiantes en un entorno de aprendizaje ubicuo.

3.3.1 Requisitos Funcionales

RF01: El sistema debe permitir a los docentes generar tareas de aprendizaje personalizadas, especificando parámetros como tema, nivel de dificultad, tipo de tarea y contexto geográfico (p. ej., puntos de interés locales).

RF02: El sistema debe integrar datos de Wikipedia y proporcionados por el docente para enriquecer las tareas con información contextual relevante.

RF03: El sistema debe soportar la generación de diversos tipos de tareas, incluyendo preguntas de desarrollo, de opción múltiple y actividades basadas en la ubicación (p. ej., tomar fotografías).

RF04: El sistema debe evaluar automáticamente las respuestas de los estudiantes y proporcionar retroalimentación inmediata.

3.3.2 Requisitos No Funcionales

RNF01: El sistema debe generar tareas en el menor tiempo posible dado los recursos computacionales disponibles.

RNF02: El sistema debe garantizar la privacidad y seguridad de los datos de los usuarios, especialmente las respuestas de los estudiantes, cumpliendo con la Ley Orgánica 3/2018 de Protección de Datos y Garantía de los Derechos Digitales[Gobierno de España, 2018].

RNF03: El sistema debe ser escalable en función del *hardware* disponible, de modo que pueda gestionar simultáneamente tanto la generación de tareas como la corrección de las mismas sin comprometer el rendimiento. Con el *hardware* utilizado durante las pruebas, y dependiendo del modelo empleado, es posible atender entre 2 y 5 usuarios simultáneos.

3.4 Casos de Uso del Sistema

Los casos de uso describen las interacciones clave entre los usuarios y el sistema, basándose en los requisitos identificados. A continuación, se presentan los casos más representativos:

3.4.1 CU01: Generación de Tareas

Actor: Docente.

Precondiciones: El docente está autenticado en el sistema.

Postcondiciones: El sistema recibe los parámetros de la tarea y genera un conjunto de tareas personalizadas para el grupo de estudiantes.

Camino Básico:

1. El docente accede a la interfaz y selecciona parámetros para la generación de las tareas.
2. El sistema utiliza un modelo de IA generativa, datos externos (p. ej., Wikipedia) y los ficheros subidos por el docente para generar las tareas teniendo en cuenta los parámetros seleccionados anteriormente.
3. Las tareas generadas se presentan al docente para su revisión y su uso posterior.

3.4.2 CU02: Evaluación de Respuestas

Actor: Estudiante.

Precondiciones: El estudiante ha completado una tarea generada por el sistema.

Postcondiciones: El sistema evalúa la respuesta y proporciona retroalimentación detallada.

Camino Básico:

1. El estudiante accede a la tarea a través de la interfaz y envía su respuesta.
2. El sistema procesa la respuesta utilizando un modelo de lenguaje y teniendo en cuenta los parámetros del estudiante (p. ej., Nivel educativo).
3. El sistema devuelve retroalimentación al estudiante, indicando aciertos y áreas de mejora.

3.4.3 CU03: Subir Documento

Actor: Docente.

Precondiciones: El docente está autenticado en el sistema.

Postcondiciones: El sistema busca la base de datos vectorial asociada al docente.

Camino Básico:

1. El usuario selecciona la opción de subir documento.
2. El sistema toma los ficheros o las URLs proporcionadas, si son válidas las convierte a texto que vectoriza y almacena.
3. El nombre del fichero o URL aparece como almacenado en la base de datos vectorial.

3.4.4 CU04: Borrar Documento

Actor: Docente.

Precondiciones: El usuario está autenticado en el sistema.

Postcondiciones: El sistema elimina la información del documento o URL almacenada.

Camino Básico:

1. El usuario indica el nombre del fichero o URL que quiere eliminar.
2. El sistema borra la información seleccionada.
3. El nombre del fichero deja de aparecer como almacenado en la interfaz.

3.4.5 CU05: Consultar Documentos Almacenados

Actor: Docente.

Precondiciones: El usuario está autenticado en el sistema.

Postcondiciones: El sistema muestra la información almacenada.

Camino Básico:

1. El usuario indica que quiere consultar la información almacenada.
2. El sistema devuelve los nombres de los ficheros almacenados al usuario.

3.5 Discusión de la Arquitectura del Sistema

La selección de una arquitectura de sistema adecuada es un paso fundamental para garantizar el cumplimiento de los requisitos funcionales y no funcionales identificados. En el contexto de este proyecto, la necesidad de integrar componentes tecnológicamente avanzados, como modelos de Inteligencia Artificial generativa, bases de datos vectoriales y fuentes de datos externas, exige una evaluación metódica de las distintas alternativas arquitectónicas disponibles.

Para este sistema, se consideraron tres paradigmas principales:

- **Arquitectura Monolítica:** Este enfoque tradicional integra todas las capas funcionales (lógica de negocio, acceso a datos y, en ocasiones, la presentación) en un único componente de software. Su principal ventaja reside en la simplicidad del desarrollo y despliegue inicial, al no requerir la gestión de comunicación entre servicios distribuidos. Sin embargo, esta estructura presenta limitaciones significativas en términos de escalabilidad y flexibilidad, ya que cualquier modificación o fallo en una parte del sistema puede afectar al conjunto, y escalar componentes individuales de forma eficiente resulta complejo.

- **Arquitectura de Microservicios:** En contraposición, este modelo divide el sistema en un conjunto de servicios pequeños, autónomos e independientes (por ejemplo, un servicio para la generación de tareas, otro para la evaluación y un tercero para la gestión de datos). Estos se comunican a través de APIs bien definidas, habitualmente REST. Ofrece una escalabilidad y flexibilidad superiores, permitiendo desarrollar, desplegar y escalar cada servicio de forma independiente. Su principal desventaja es el notable incremento en la complejidad de gestión, que incluye la orquestación de servicios, la monitorización, la resiliencia y la consistencia de los datos en un entorno distribuido.

- **Arquitectura en Capas:** Este paradigma, que puede considerarse un punto intermedio, organiza el sistema en capas lógicas horizontales, pero generalmente dentro de una misma aplicación. Permite una clara separación de responsabilidades y una mayor modularidad que la arquitectura monolítica, facilitando su mantenimiento. No obstante, su flexibilidad para el escalado sigue siendo menor que la de los microservicios, ya que las capas permanecen acopladas dentro de una única unidad de despliegue.

Tras analizar las ventajas y desventajas de cada alternativa en el contexto de este Trabajo de Fin de Grado, se optó por implementar una arquitectura monolítica. Esta decisión se fundamenta principalmente en la necesidad de agilizar el desarrollo inicial y simplificar la gestión del sistema, priorizando la rápida construcción de un prototipo funcional. Una arquitectura de este tipo permite consolidar todas las funcionalidades (generación de tareas, evaluación, almacenamiento) en una sola aplicación ejecutada en una única máquina, lo que facilita considerablemente la gestión del ciclo de vida del software y el mantenimiento.

De esta manera, se evita la complejidad inherente a las infraestructuras distribuidas, como la configuración de redes, el descubrimiento de servicios y la gestión de la comunicación entre componentes. Además, la comunicación interna entre los distintos módulos del sistema se simplifica enormemente, ya que no es necesario implementar y mantener protocolos externos como APIs REST para su interacción, lo que agiliza el desarrollo inicial y facilita la realización de pruebas integradas, rápidas y eficientes.

Si bien se reconoce que esta arquitectura no es la más óptima en términos de escalabilidad a largo plazo, se considera la más pragmática para los objetivos y el alcance de este proyecto. Se plantea como una base sólida sobre la cual, en trabajos futuros, se podría evolucionar hacia un modelo de microservicios a medida que las necesidades del sistema crezcan y se requiera una mayor escalabilidad y desacoplamiento.

3.6 Adaptación de plantillas LOD a la IA Generativa

Para satisfacer los nuevos requisitos, se reemplaza el sistema de plantillas LOD por un *prompt* parametrizado que actúe como instrucción dinámica para un LLM. A partir del análisis de los parámetros originales (tipo de entidad, propiedades como `dbo:style`, ubicación, `skos:related`) y de las necesidades de los educadores, se definen los siguientes grupos de parámetros:

Perfil del Estudiantado: Nivel educativo, dificultad y conocimientos previos.

Contexto del Contenido: Materia y tema, refinando la contextualización temática que antes se basaba únicamente en etiquetas `skos:related`.

Fuente de Datos: Posibilidad de subir ficheros o cargar desde URL, rompiendo la dependencia exclusiva del LOD mediante RAG.

Contexto de la Tarea: Espacio de la pregunta (físico/virtual) y puntos de interés A/B, heredando y mejorando la geolocalización y comparación de entidades.

Especificación Pedagógica: Taxonomía de Bloom para indicar el nivel cognitivo deseado (por ejemplo, “analizar”, “comparar”) y tipo de tarea (por ejemplo, “MCQ”, “toma una foto”).

El cambio de paradigma se resume en la siguiente comparativa:

Característica	**Plantillas LOD**	**Prompt IAGen**
Lógica	Extractiva: filtra entidades e inserta en texto fijo.	Generativa: comprende contexto y crea texto nuevo.
Fuente de Datos	Acoplada a ontologías LOD específicas.	Agnóstica: cualquier texto (fichero, URL) vía RAG.
Flexibilidad	Baja: requiere programar nuevas plantillas.	Alta: nuevos tipos de preguntas vía parámetros.
Personalización	Limitada: perfil genérico.	Profunda: nivel, dificultad y conocimientos detallados.
Control	Basado en propiedades de datos (`dbo:style`).	Basado en intenciones pedagógicas (Bloom, dificultad).

Tabla 3.1: Comparativa entre plantillas LOD y *prompt* parametrizado.

Para que las tareas generadas por el sistema de IA sean programáticamente útiles y puedan ser integradas en una aplicación de aprendizaje, es indispensable que su salida no sea texto libre, sino un formato estructurado y predecible. La inspiración para definir estos formatos proviene directamente del análisis de las tareas propuestas por profesores y expertos en la Tabla 2 del artículo *Supporting contextualized learning with linked open data*[Ruiz-Calleja et al., 2021].

Type	**TEL Experts**	**Teachers**
Outcome		
Text	36 (71 %)	15 (45 %)
Photo	18 (35 %)	15 (45 %)
Audio	0 (0 %)	4 (12 %)
MCQ selection	1 (2 %)	2 (6 %)
Map	1 (2 %)	0 (0 %)
Video	1 (2 %)	1 (3 %)

Tabla 3.2: Recreación de la sección de *Outcome* de la tabla 2 [Ruiz-Calleja et al., 2021].

Dicha tabla clasifica las tareas según diversas características, siendo la más relevante para nuestro propósito la columna *Outcome* (Resultado esperado). En ella se observa que las tareas manuales solicitaban resultados variados, como la redacción de *Text*, la toma de una *Photo*, o la selección en un *MCQ selection*. Estas categorías representan los tipos de interacción que los educadores consideran pedagógicamente valiosos en un contexto de aprendizaje ubicuo.

Nuestra aproximación adapta estos resultados esperados al paradigma de la IA Generativa, traduciéndolos en tipos de preguntas concretas que el LLM puede generar. La forma de materializar esta adaptación es instruir al modelo para que su respuesta se adhiera a un esquema de salida específico para cada tipo de tarea. Esto garantiza la consistencia, fiabilidad y facilidad de integración de las tareas generadas.

A partir de este análisis, hemos definido los siguientes tipos de tareas como objetivos iniciales, cada uno con su correspondiente formato de salida estructurada:

- **Desarrollo simple (Respuesta abierta):** Corresponde al *Outcome: Text* de la Tabla 2. Es ideal para preguntas que requieren reflexión, descripción o argumentación. La estructura incluirá un campo para el enunciado de la pregunta y la respuesta correcta a esta pregunta.
- **Opción Múltiple (Tipo test):** Es la implementación directa del *Outcome: MCQ selection*. Este formato es perfecto para evaluar el conocimiento factual de manera rápida y objetiva. La esctructura contendrá el enunciado, una lista de opciones y la indicación de la respuesta correcta.
- **Verdadero o Falso:** Aunque no figura explícitamente como una categoría mayoritaria en la tabla, es un tipo de pregunta fundamental para la verificación rápida de conceptos. Su estructura incluirá una afirmación y un valor booleano indicando la respuesta correcta.
- **Sacar Fotografía:** Refleja el *Outcome: Photo*. Esta tarea no evalúa conocimiento de forma tradicional, sino que promueve la observación activa en un entorno físico.

Al definir estos formatos de salidas estructuradas, no solo hacemos que las tareas sean técnicamente manejables, sino que también ofrecemos a los educadores la capacidad de solicitar tipos de interacción específicos, replicando la riqueza pedagógica observada en las tareas diseñadas manualmente, pero con la flexibilidad y escalabilidad de la IA Generativa. La especificación detallada de cada esquema JSON se abordará en el capítulo de Diseño e Implementación.

3.7 Conclusiones

La fase de análisis ha permitido definir los requisitos funcionales y no funcionales del sistema, así como los casos de uso que guiarán su desarrollo. Se ha establecido que el sistema debe generar tareas contextualizadas, evaluar respuestas automáticamente y ser accesible en entornos ubicuos, todo ello soportado por una arquitectura *monolítica* que asegura el desarrollo de un prototipo inicial. Estos resultados sientan las bases para la siguiente etapa del proyecto, el Diseño del Sistema (Capítulo 4), donde se detallarán los componentes específicos de la arquitectura, como la integración de *LangChain* para la gestión de contexto, el uso de Ollama para la generación de contenido y la estructura de la API REST que conectará los servicios. El diseño partirá de las necesidades identificadas aquí, especificando cómo se implementarán técnicamente las funcionalidades propuestas.

Capítulo 4

Diseño

En este capítulo se detalla el proceso de diseño del sistema propuesto, que combina el aprendizaje ubicuo con la generación de tareas asistida por un modelo de lenguaje siguiendo la técnica de Generación Aumentada por Recuperación (RAG). Se describen la arquitectura general del sistema, el diseño del servidor y de la API REST desarrollada, la construcción de los *prompts* optimizados mediante la metodología C.R.A.F.T., así como los componentes encargados de la generación de tareas y la corrección automática. También se define el formato de las tareas generadas (en formato JSON estructurado mediante modelos Pydantic) y se presenta una visión de la arquitectura completa integrando todos los elementos.

4.1 Introducción

El presente capítulo tiene como finalidad detallar el diseño de la aplicación desarrollada en el marco de este TFG, sentando las bases necesarias para cumplir con los requisitos identificados y documentados en la fase de análisis previa, tal como se expone en el Capítulo 3. La exposición del diseño se inicia desde una perspectiva general, presentando la arquitectura lógica del sistema, para luego avanzar hacia una descripción minuciosa de los componentes principales que conforman la aplicación distribuida. En este contexto, se abordan las funciones específicas asignadas al servidor y al cliente, así como las interacciones entre estos elementos y, cuando sea pertinente, con servicios externos.

La arquitectura general del sistema se plantea desde un enfoque lógico, con el propósito de esclarecer las responsabilidades de cada componente de la aplicación. Este enfoque permite establecer una visión integral del sistema antes de profundizar en los detalles de su implementación. A continuación, se procede a diseñar de manera específica las partes desarrolladas en este TFG que se encargan del servidor del sistema. En el caso del servidor, se destacan sus características principales, con especial énfasis en la exposición de recursos que facilitan la obtención de información para la generación de tareas educativas. Asimismo, se subraya su capacidad para gestionar usuarios, almacenar datos de forma persistente y garantizar la privacidad mediante el procesamiento local de la información, evitando la dependencia de servicios externos. Este diseño asegura la integridad

y el aislamiento de los datos de cada usuario, al tiempo que incorpora un contexto ubicuo que enriquece las tareas generadas.

4.2 Arquitectura del sistema

El sistema se ha diseñado como una aplicación monolítica capaz de satisfacer los requisitos indicados en el Capítulo 3. A diferencia de arquitecturas distribuidas complejas, este enfoque consolida toda la lógica de negocio —incluida la gestión de datos, la interacción con los modelos de lenguaje y la exposición de la API REST [Fielding, 2000]— en un único servidor. Esta decisión de diseño, como se justificó en la sección 3.5, prioriza la simplicidad, la facilidad de despliegue y un control total sobre el flujo de datos, aspectos cruciales en un entorno académico y de investigación.

La arquitectura del sistema busca disminuir al máximo la probabilidad de alucinación del modelo y maximizar las posibilidades de generar tareas relevantes en el contexto educativo del alumno. Para lograrlo, se integra un potente pipeline de RAG directamente en el servidor.

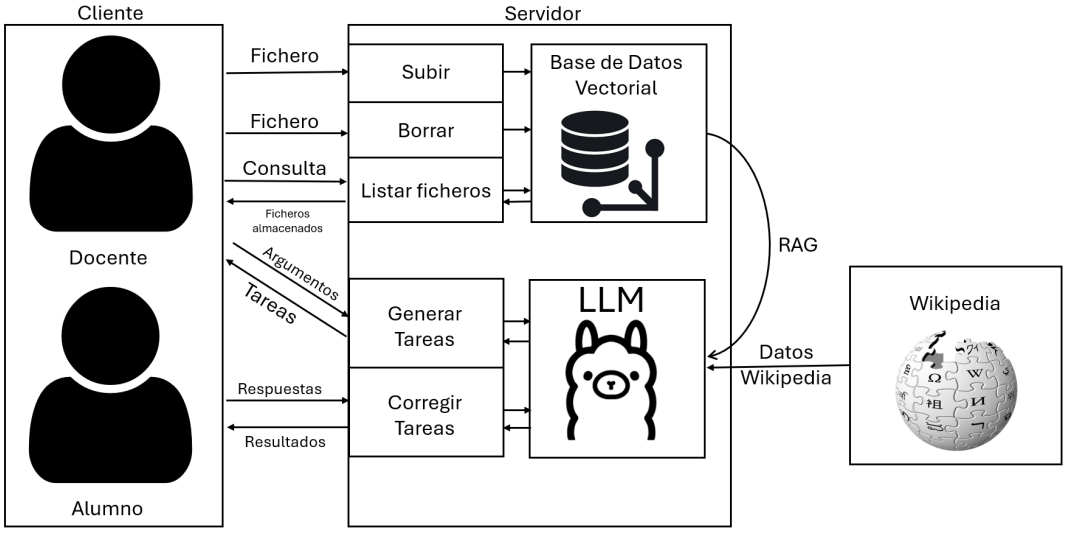

Figura 4.1: Esquema de la arquitectura del sistema.

Como se muestra en la Figura 4.1, la arquitectura se compone de los siguientes elementos clave:

- **Cliente:** Es la interfaz con la que interactúa el usuario (docente o alumno). Puede ser una aplicación web, una app móvil o cualquier otro software capaz de consumir una API REST. Su responsabilidad principal es presentar la información de manera intuitiva y enviar las peticiones del usuario al servidor. En este TFG no se desarrolla un cliente específico, pero la API REST se diseña para que cualquier cliente pueda implementarse fácilmente.
- **Servidor (Backend):** Es el corazón del sistema y el foco de este proyecto. Alberga toda la lógica y se encarga de:

  - *API REST:* Exponer los *endpoints* para que el cliente pueda interactuar con el sistema (gestionar ficheros, generar tareas, corregir respuestas).
  - *Gestión de Usuarios y Seguridad:* Autenticar las peticiones mediante claves de API, garantizando que cada usuario acceda únicamente a sus propios datos.
  - *RAG*:
    * *Gestión de la Base de Conocimiento:* Procesar los documentos y URLs subidos por el usuario (docente). Incluye la fragmentación del texto, la generación de *embeddings* y su almacenamiento en una base de datos vectorial persistente.
    * *Recuperación de Contexto:* Ante una solicitud de generación de tareas, buscar en la base de datos vectorial los fragmentos de texto más relevantes para el tema solicitado.
    * *Integración con Modelos de IA:* Orquestar las llamadas a los modelos de lenguaje locales. Construye los *prompts* inyectando el contexto RAG y la información de fuentes externas, y fuerza la salida en formato JSON estructurado.

- **Componentes en la generación de tareas:**
  - *Base de datos vectorial:* Es la que almacena los *embeddings* de los documentos del usuario de forma persistente. Cada usuario tiene su propia colección aislada del resto.
  - *Gestor de LLM:* Plataforma para ejecutar localmente modelos de lenguaje de gran tamaño. Al no depender de servicios externos, se garantiza la privacidad de los datos y se reducen los costes operativos.

- **Servicios Externos (Opcionales):**
  - *Fuentes de datos externos:* Utilizada para enriquecer el contexto de las tareas cuando se proporciona un Punto de Interés (POI), como un monumento o un lugar histórico. El servidor consulta esta API para obtener descripciones y datos relevantes (p.ej. Wikipedia).

Flujo de trabajo típico

Para ilustrar cómo interactúan estos componentes, consideremos el siguiente ejemplo de uso:

1. **Fase de Preparación (Docente):**
   a) El docente se autentica con su API Key mediante el cliente. Sube sus apuntes o otra fuente de información sobre el "Arte Románico en Palencia".
   b) El servidor procesa el fichero con el RAG, genera los embeddings y los almacena en la colección de vectores en la base de datos vectorial del docente.
2. **Fase de Generación de Tareas (Docente):**

a) El docente solicita 3 tareas de opción múltiple sobre "Arte Románico" en el POI "Catedral de Palencia". Que con sus apuntes hara *RAG* para dar contexto al *LLM*

b) El servidor:

1) Consulta la Wikipedia API para obtener información sobre la "Catedral de Palencia".
2) Realiza una búsqueda semántica en ChromaDB para recuperar los fragmentos más relevantes.
3) Construye un *prompt* que integra la información de Wikipedia y los fragmentos recuperados.
4) Llama al modelo de lenguaje local, exigiendo salida en formato JSON.

c) El cliente recibe el JSON con las 3 tareas con su solución correspondiente y las presenta al docente para su revisión.

3. **Fase de Realización y Evaluación (Estudiante):**

a) El estudiante responde a una de las tareas.

b) La respuesta y la solución correcta proporcionada en el momento de generar las tareas se envían al servidor.

c) El servidor utiliza un modelo de IA local para comparar la respuesta con la solución y devuelve retroalimentación automática.

Esta arquitectura monolítica y centrada en IA local ofrece una solución potente, segura y personalizable, alineada con las necesidades de un entorno educativo moderno."'

4.3 Diseño del servidor

El servidor del sistema tendrá que exponer una serie de recursos para la obtención de la información para generar las tareas. El servidor integra la gestión de usuarios, el almacenamiento de datos, la integración de contexto ubicuo y la potencia de cómputo necesaria para ejecutar los modelos de inteligencia artificial de forma local. Que la información se almacene de forma local es necesario para no tener que dar información a servicios de terceros. Así se garantiza la privacidad de los documentos del docente o de la información del alumnado.

4.3.1 Gestión de usuarios y seguridad

Dado que el sistema está diseñado para ser utilizado por múltiples usuarios (estudiantes o profesores), el servidor implementa una gestión de usuarios básica mediante un mecanismo de autenticación con claves de API. En lugar de manejar sesiones complejas o datos personales extensos, se asigna a cada cliente registrado una clave única y secreta. Esta clave de API debe incluirse en la cabecera de cada petición HTTP que el cliente realice al servidor.

El servidor verifica la validez de esta clave en cada solicitud, identificando así al usuario y autorizando el acceso a los recursos. Este esquema, aunque sencillo, cumple dos funciones clave:

- **Seguridad**: Impide accesos no autorizados a la API.
- **Multiusuario**: Permite asociar los datos a un usuario específico, garantizando que cada usuario solo pueda acceder a su propia información.

4.3.2 Almacenamiento de datos

El servidor gestiona el almacenamiento persistente de todos los datos del sistema.

- **Información para la generación de tareas**: Utilizando la base de datos vectorial en modo persistente, los *embeddings* y los fragmentos de texto se almacenan en el disco del servidor. Cada usuario tiene su propia colección en la base de datos, identificada de forma única, lo que garantiza el aislamiento de los datos. Esto significa que las búsquedas RAG de un usuario solo se realizarán sobre los documentos que ese usuario ha subido.
- **Ficheros de usuario**: Los ficheros originales que los usuarios suben se guardan en el sistema de ficheros del servidor, en una estructura de carpetas organizada por los identificadores únicos de cada usuario. Esto permite listar y eliminar de los ficheros fuente.
- **Tareas generadas**: Las tareas generadas por el LLM, junto con sus soluciones, no se almacenan de forma persistente en una base de datos. Se mantienen en la memoria del servidor durante la sesión activa del usuario, lo que simplifica el diseño.

4.3.3 Integración de contexto ubicuo

Un objetivo fundamental del sistema es soportar el aprendizaje ubicuo, lo que implica que las tareas puedan adaptarse al contexto físico o situacional del estudiante. El diseño del servidor incorpora este contexto dinámico de dos maneras:

1. **Información de Puntos de Interés (POI)**: El sistema puede recibir como entrada el nombre de un punto de interés (por ejemplo, Catedral de Valladolid). Utilizando la API de **Wikipedia**, el servidor extrae automáticamente el artículo correspondiente a ese POI. Esta información se inyecta directamente en el *prompt* del LLM, permitiéndole generar preguntas que hagan referencia a la historia, arquitectura o detalles específicos del lugar donde se encuentra el estudiante.
2. **Conocimiento del usuario (RAG)**: El sistema enriquece aún más el contexto utilizando la arquitectura RAG. Se buscan en la base de conocimiento del usuario los fragmentos más relevantes para el tema de la tarea y el POI. De este modo, las preguntas no solo se basan en la información genérica de Wikipedia, sino que también se conectan con los apuntes y materiales que el profesor o el propio alumno han cargado previamente.

Esta doble fuente de contexto permite crear tareas altamente personalizadas y situadas, aumentando la calidad y relevancia en un entorno de aprendizaje ubicuo.

4.3.4 Arquitectura RAG: indexación y recuperación de información

Una pieza central del sistema es el subsistema de RAG, encargado de incorporar información contextual relevante en la generación de las tareas. Este enfoque es fundamental para mitigar las alucinaciones de los modelos de lenguaje y asegurar que las tareas generadas se basen en conocimiento específico y fiable. Este subsistema opera en dos fases principales: la fase de indexación (preparación de la base de conocimiento) y la fase de recuperación semántica.

Fase de indexación

En primer lugar, el sistema debe preparar y almacenar la información de referencia de modo que pueda ser recuperada eficientemente más adelante. A tal efecto, el contenido relevante proporcionado por el usuario, el sistema acepta tanto `.pdf`, `.txt`, `.docx`, transcripciones de vídeos de YouTube y contenido de páginas web que se transforma en representaciones vectoriales mediante un modelo de *embeddings*.

El proceso de indexación se desglosa en los siguientes pasos:

1. **Subida de datos**: Cada fuente se convierte en un formato de texto y fuente del contenido.

2. **Fragmentación**: Cada documento o fuente de datos es dividido en fragmentos manejables llamados *chunks* de un tamaño fijo de caracteres con un cierto solapamiento entre fragmentos. Esta división es crucial, ya que fragmentos demasiado grandes pueden diluir la información relevante, mientras que fragmentos demasiado pequeños pueden carecer de contexto. Se utiliza una estrategia de división de texto recursiva que intenta mantener la cohesión semántica de los fragmentos.

3. **Vectorización**: A cada fragmento de texto se le calcula un vector en un espacio de alta dimensión que captura su contenido semántico. Este proceso de codificación semántica se realiza con un modelo de *embedding* entrenado para captar similitudes de significado entre textos. Dos fragmentos con significado similar tendrán vectores cercanos en este espacio vectorial.

4. **Almacenamiento**: Los vectores resultantes se almacenan en una base de datos vectorial. Junto a cada vector, se guardan el texto del fragmento original y metadatos que identifican la fuente y el identificador único del usuario propietario de la información.

De esta forma, la base de conocimiento queda indexada: es posible buscar información relevante mediante comparaciones de similitud entre vectores (generalmente usando la similitud del coseno), en lugar de por coincidencia de palabras clave exactas. La elección de una base de datos vectorial como ChromaDB permite almacenar eficientemente cientos o miles de embeddings y recuperarlos en tiempo real usando estructuras de índice optimizadas.

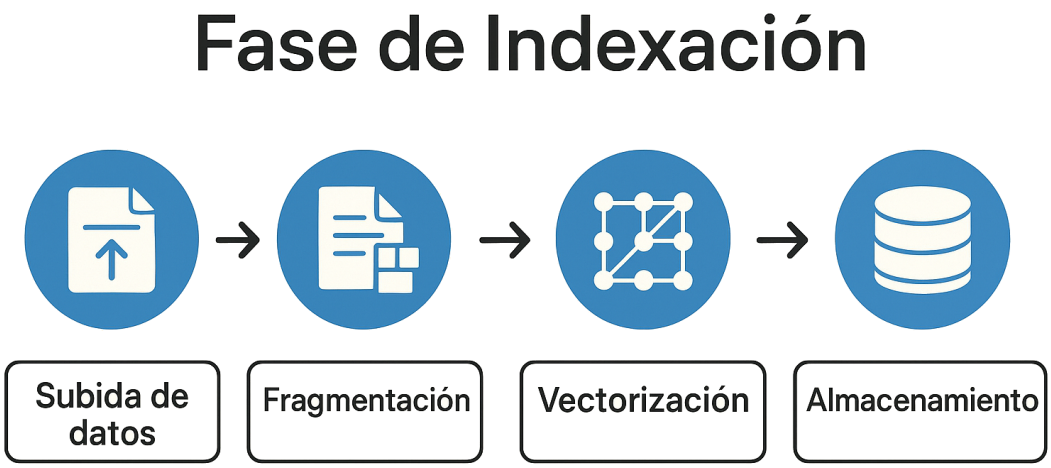

Figura 4.2: Esquema de la fase de indexación.

Fase de recuperación semántica

Una vez la base de conocimiento está indexada, la fase de recuperación se encarga de, dado un requerimiento o contexto actual, extraer los fragmentos de información más relevantes para la generación de las tareas. Cuando el sistema recibe una solicitud para generar nuevas tareas (por ejemplo, sobre un tema específico indicado por el usuario), el proceso es el siguiente:

1. **Creación de la consulta**: De la consulta del usuario se utiliza el **tema** de las tareas, el tema se transforma en un vector utilizando el mismo modelo de *embedding* empleado en la fase de indexación.

2. **Búsqueda por similitud**: Con este vector de consulta, la base de datos vectorial realiza una búsqueda por similitud. Devuelve los fragmentos de información almacenados cuyos vectores son más cercanos al vector de consulta en el espacio vectorial; es decir, los fragmentos más semánticamente relacionados con la solicitud.

3. **Construcción del contexto**: Esta búsqueda típicamente retorna un conjunto de los k fragmentos más similares, donde k es un parámetro configurable. Dichos fragmentos recuperados constituyen el **contexto** que luego se proporcionará al modelo generativo de lenguaje. Junto a estos fragmentos se pasan los metadatos para poder saber de donde proviene la información.

En esencia, el sistema está recuperando las piezas de conocimiento más pertinentes desde su base de datos para que el modelo las use al crear las tareas. Al limitar la generación de contenido a información respaldada por este contexto recuperado, se busca que las tareas producidas sean pertinentes, correctas y estén alineadas con el material de referencia disponible.

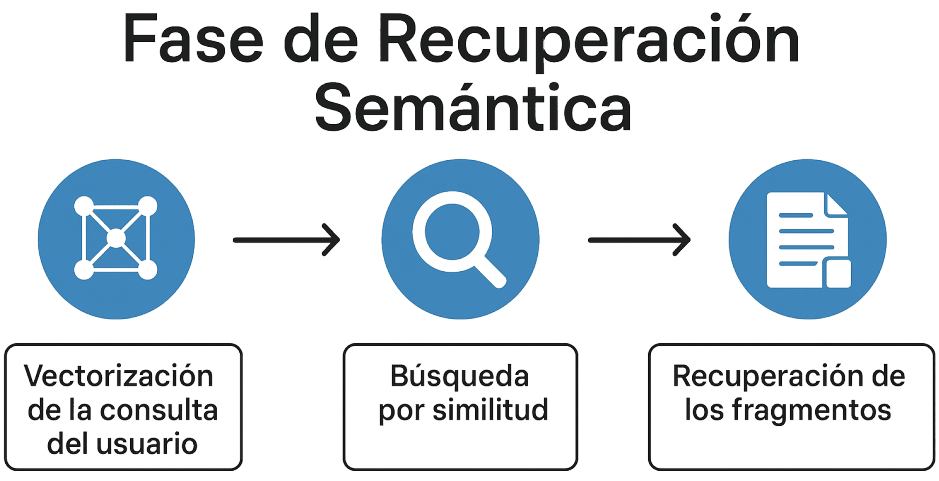

Figura 4.3: Esquema de la fase de recuperación semántica.

4.3.5 Diseño de la API REST

Diseño de la API REST

Para que una aplicación cliente (como una app móvil o una plataforma web) pueda interactuar con las funcionalidades de generación y evaluación de tareas, es necesario definir una interfaz de comunicación robusta y estandarizada. El servidor expone sus capacidades a través de un servicio web que sigue los principios de la arquitectura REST (*Representational State Transfer*) [Fielding, 2000]. Esta elección se fundamenta en varios principios clave que son ideales para este proyecto:

- **Arquitectura Cliente-Servidor:** Se mantiene una separación clara entre el cliente (la interfaz de usuario) y el servidor (la lógica de negocio y el acceso a la IA). Esta modularidad permite que el *backend* desarrollado en este TFG pueda ser consumido por diferentes tipos de clientes en el futuro sin necesidad de modificaciones.
- **Comunicación sin Estado (*Stateless*):** Cada petición del cliente al servidor contiene toda la información necesaria para su procesamiento. El servidor no almacena estado de sesión entre peticiones. Por ejemplo, la generación de tareas se inicia encolando una solicitud mediante `POST /query_task` (incluyendo tema, dificultad, parámetros, etc.) y su estado se consulta con `GET /query_task/{id_query_task}`, sin requerir contexto previo en el servidor.
- **Interfaz Uniforme:** Las interacciones se basan en la manipulación de recursos identificados por URIs únicas y operados mediante un conjunto limitado de métodos HTTP (GET, POST, DELETE, ...). En nuestro caso, los recursos principales son: los recursos de conocimiento (`/learning_recourses`), las peticiones de generación (`/query_task`) y las tareas de aprendizaje con sus subrecursos de respuestas y retroalimentación (`/learning_tasks/{id}/answers[/{id}][/feedback]`).

Para garantizar la seguridad y la personalización, la API implementa un sistema de autenticación basado en claves mediante la cabecera `X-API-KEY`. Cada usuario registrado posee una *API Key* única. Esto permite al servidor validar la petición y asociarla a los documentos y tareas específicos de ese usuario, asegurando el aislamiento y la privacidad de los datos. Asimismo, las autorizaciones se aplican por rol (profesor o estudiante) en cada petición.

Diseño de Endpoints La API se ha diseñado en torno a tres funcionalidades principales: la gestión del conocimiento del usuario (RAG), la generación de tareas y la corrección automática de respuestas.

A continuación se describen los endpoints, agrupados por rol de acceso.

1. **Endpoints exclusivos para Profesor**

   `POST /learning_recourses` Crea un nuevo miembro (recurso) en la colección de conocimiento (RAG) del profesor. El servidor procesa la entrada (fragmentación, vectorización) y la almacena en la colección asociada a su API Key.
   Entrada: `multipart/form-data` con un fichero (`.pdf`, `.txt`, `.docx`) o un campo de texto `URL`.

Salida: Objeto JSON con el identificador del recurso creado y un mensaje de éxito o error.

`GET /learning_recourses` o `GET /learning_recourses/{id_learning_recourse}` Lista todos los miembros de la colección del profesor o recupera la representación de un miembro concreto.
Entrada: Ninguna (solo la cabecera `X-API-KEY`); en la variante con identificador, este viaja en la ruta.
Salida: (Listado) JSON con un array de recursos; (Detalle) JSON con los metadatos del recurso solicitado.

`DELETE /learning_recourses` o `DELETE /learning_recourses/{id_learning_recourse}` Elimina la colección completa del profesor o, si se proporciona identificador, únicamente ese miembro. En ambos casos se borran también los *chunks* asociados en la base vectorial para mantener la consistencia del contexto RAG.
Entrada: Ninguna (solo la cabecera `X-API-KEY`); en la variante con identificador, este viaja en la ruta.
Salida: Objeto JSON con un mensaje de éxito.

`POST /query_task` Inicia una petición de generación de tareas con los parámetros proporcionados. La petición se encola y se devuelve un identificador para su seguimiento.
Entrada: Cuerpo JSON con parámetros como `tipo_pregunta`, `tema`, `cantidad`, `dificultad`, `nivel`, `poi`, etc.
Salida: Objeto JSON con `id_query_task` y estado inicial de la petición.

`GET /query_task/{id_query_task}` Consulta el estado de una petición de generación. Devuelve `false` si las tareas aún no están listas, o las URIs de las tareas cuando se han generado.
Entrada: Identificador de la petición en la ruta; cabecera `X-API-KEY`.
Salida: Objeto JSON, p. ej.:

```
{ "ready": false }
```

o bien

```
{ "ready": true, "tasks": ["uri1", "uri2", "..."] }
```

2. **Endpoints para Profesor y Estudiante**

`GET /learning_tasks` o `GET /learning_tasks/{id_learning_task}` Devuelve todas las tareas asociadas a un profesor (creadas por él) o asignadas a un estudiante (según la API Key/rol), o bien, con identificador, la tarea concreta.
Entrada: Ninguna (solo la cabecera `X-API-KEY`); en la variante con identificador, este viaja en la ruta.
Salida: (Listado) Array JSON de tareas; (Detalle) Objeto JSON con la tarea solicitada.

`POST /learning_tasks/{id_learning_task}/answers` Crea un *subrecurso* respuesta asociado a la tarea indicada (puede ser respuesta del profesor o del estudiante).
Entrada: Cuerpo JSON con la respuesta:

```
{
  "contenido_respuesta": "...",
}
```

Salida: Objeto JSON con el `id_answer` creado y un mensaje de éxito.

`GET /learning_tasks/{id_learning_task}/answers` Devuelve todas las respuestas asociadas a una tarea.
Entrada: Identificador de la tarea en la ruta; cabecera `X-API-KEY`.
Salida: Array JSON de respuestas.

`GET /learning_tasks/{id_learning_task}/answers/{id_answer}` Devuelve la respuesta identificada.
Entrada: Identificadores de tarea y de respuesta en la ruta; cabecera `X-API-KEY`.
Salida: Objeto JSON con el contenido y metadatos de la respuesta.

`GET /learning_tasks/{id_learning_task}/answers/{id_answer}/feedback` Devuelve la retroalimentación asociada a una respuesta concreta (p. ej., evaluación automática, comentarios, rúbrica).
Entrada: Identificadores de tarea y de respuesta en la ruta; cabecera `X-API-KEY`.
Salida: Objeto JSON con el *feedback*:

```
{
  "is_ok": true,
  "comentarios": "...",
}
```

El intercambio de datos se realiza exclusivamente en formato JSON, dada su ligereza, legibilidad y facilidad de procesamiento por parte de cualquier cliente web o móvil. La estricta adhesión a los esquemas Pydantic en las respuestas garantiza la fiabilidad y robustez de la comunicación entre cliente y servidor.

4.4 Diseño de los *prompts*

Para lograr que el modelo de lenguaje genere exactamente el contenido deseado, es crucial elaborar cuidadosamente los *prompts*. En este proyecto se ha adoptado la metodología **C.R.A.F.T.** como guía para la construcción de *prompts* efectivos. C.R.A.F.T. es un acrónimo que resume los elementos clave que debe incluir un buen *prompt*: **Context** (Contexto), **Role** (Rol), **Action** (Acción), **Format** (Formato) y **Tone** (Tono).

A continuación, se detalla cómo se ha aplicado esta metodología en los *prompts* del sistema:

- **Contexto (Context):** Es la parte más rica del *prompt*. Se le proporciona al modelo toda la información necesaria para generar tareas relevantes.
- **Rol (Role):** Se le asigna al modelo una personalidad específica. Por ejemplo:

  > "Eres un experto en materia. Estás desarrollando preguntas de aprendizaje ubicuo..."

Este rol guía al modelo para que su estilo y enfoque se ajusten a los de un tutor inteligente.

- **Acción (Action):** Se le indica de forma explícita la tarea a realizar. Por ejemplo:

  > "Debes crear preguntas abiertas y sencillas basadas en la taxonomía de Bloom (taxonomia) enfocadas en la observación directa del monumento".

- **Formato (Format):** Se le exige al modelo que devuelva la salida estrictamente en formato JSON, con estructura definida por un esquema Pydantic. Por ejemplo:

  > "Proporciona la respuesta en formato JSON [...] con la estructura: `{"question": "...", "solucion": "..."}`".

  Esto asegura que la salida del modelo pueda ser procesada automáticamente por la aplicación.

- **Tono (Tone):** Se define el estilo de comunicación deseado. Por ejemplo:

  > "Utiliza un tono educativo, claro y motivador. Escribe las preguntas como un profesor guía que invita a la exploración...".

Prompt del Generador de Tareas (Ejemplo para Preguntas Largas y Simples):

```
system_message = {
    "role": "system",
    "content": f""" Eres un experto en {materia}. Estás desarrollando preguntas de aprendizaje ubicuo sobre el tema {tema} para alumnos de {nivel}. Estos estudiantes están físicamente en un espacio {ubicuo}, es decir, en el lugar del monumento relacionado. Cuentan con el siguiente conocimiento previo de clase: {conocimiento_alumno}. {frase_poi} Además, el contexto de los apuntes de clase es: {contexto}.

    Debes crear preguntas abiertas y sencillas basadas en la taxonomía de Bloom ({taxonomia}) enfocadas en la observación directa del monumento. Cada pregunta debe formularse de tal manera que solo pueda ser respondida por un alumno que se encuentre presencialmente allí, interactuando con ese entorno.

    Las preguntas van dirigidas a tus alumnos de {nivel}, por lo que deben adecuarse a su nivel educativo y conectar con el conocimiento que ya poseen. Ten en cuenta su conocimiento previo proporcionado y el contexto del monumento para asegurarte de que las preguntas sean relevantes y comprensibles para ellos.

    Proporciona la respuesta en formato JSON, con cada pregunta y su solución correspondiente. Si la cantidad es mayor que 1, devuelve una lista de objetos JSON, donde cada objeto tenga la estructura:
    "question": "pregunta de aprendizaje ubicuo",
    "solucion": "respuesta esperada a la pregunta de aprendizaje ubicuo"
    Asegúrate de que el JSON sea válido y bien formado.

    Utiliza un tono educativo, claro y motivador. Escribe las preguntas como un profesor guía que invita a la exploración, asegurándote de que el lenguaje sea apropiado para el nivel de los alumnos y fomente la curiosidad sobre el monumento.
    """}

user_message = {
    "role": "user",
    "content": f"Genera {cantidad} preguntas abiertas de desarrollo sobre {tema} de la materia {materia} para tus alumnos, formuladas de manera sencilla. Asegúrate de que cada pregunta requiera que el alumno esté físicamente presente en el monumento para poder responderla."
}
```

Código 4.1: Prompt para la generación de tareas de desarrollo largo y simple. Donde se pueden observar los diferentes apartados de la metodología tomada para generar el *prompt*

Prompt del Corrector Automático:

```
system_message= {
    "role": "system",
    "content": """
    Eres un corrector educativo automatizado diseñado para proporcionar retroalimentación personalizada y constructiva a los estudiantes de manera directa, es decir, vas a estar hablando con el estudiante.
    Tu tarea es analizar la respuesta de un estudiante a una pregunta específica, compararla con la respuesta correcta y ofrecer una evaluación detallada que incluya tanto los aciertos como los errores.
    Debes explicar los conceptos relevantes de manera clara y relacionarlos con el objetivo de aprendizaje de la pregunta. Además, proporciona sugerencias prácticas para que el estudiante pueda mejorar su comprensión y recomienda recursos adicionales si es apropiado. Mantén un tono alentador y enfocado en el crecimiento del estudiante.

    Cuando recibas el nombre del estudiante, dirígete a él directamente para hacer la retroalimentación más personal. Si no se proporciona el nombre, omite cualquier referencia personal.
    Devuelve exclusivamente un JSON válido con la siguiente estructura:
      "is_ok": true,  // true si la respuesta es correcta, false si es incorrecta
      "diferencias": "Explicación breve de las diferencias si existen"
    Devuelve la salida en formato JSON válido.""" }

user_message = {
    "role": "user",
    "content": f"Vas a analizar la respuesta del alumno {usuario} que tiene un nivel de estudiante de {nivel} a la Pregunta: {pregunta}\nRespuesta correcta: {respuesta_correcta}\nRespuesta de {usuario}: {respuesta_alumno}"
}
```

Código 4.2: Prompt para la corrección automática de respuestas abiertas.

Gracias a este diseño meticuloso, el LLM recibe instrucciones claras y un marco de trabajo bien definido, lo que aumenta drásticamente la calidad, fiabilidad y consistencia de los resultados.

4.5 Diseño de las Tareas

El pilar de este diseño es el uso de salidas en formato JSON forzado. En lugar de permitir que el LLM genere texto libre, que requeriría un complejo y frágil post-procesamiento para su interpretación, se le instruye explícitamente para que devuelva una respuesta que se adhiere estrictamente a un esquema predefinido.

Este enfoque ofrece dos ventajas fundamentales:

- **Fiabilidad del sistema:** Se elimina la ambigüedad y se busca que la aplicación cliente pueda interpretar y renderizar las tareas correctamente.

- **Riqueza pedagógica:** Permite diseñar diversos tipos de tareas, cada una con un objetivo pedagógico distinto, que van desde la simple memorización hasta el análisis y la observación en el entorno físico.

A continuación, se detallan los tipos de tareas diseñadas y su estructura JSON.

4.5.1 Tarea de Desarrollo Simple (Long Simple)

Propósito pedagógico: Evaluar la comprensión de un tema, requiriendo que el estudiante elabore una respuesta abierta basada en el contexto proporcionado (ya sea de los documentos del usuario a través de RAG o de la información del punto de interés).

Estructura JSON:

```
{
  "question": "string",
  "solucion": "string"
}
```

Donde `question` es la pregunta formulada por el LLM y `solucion` es una respuesta ideal o esperada, generada también por el modelo. Esta solución es crucial para la posterior corrección automática.

4.5.2 Tarea de Opción Múltiple (MCQ)

Propósito pedagógico: Evaluar el conocimiento factual y la capacidad de discriminación entre opciones. Es una herramienta eficaz para la autoevaluación rápida.

Estructura JSON:

```
{
  "question": "string",
  "option0": "string",
  "option1": "string",
  "option2": "string",
  "option3": "string",
  "correct_option": integer
}
```

El campo `corect_option` es un entero (0–3) que indica el índice de la respuesta correcta, garantizando una evaluación inequívoca. La capacidad del LLM para generar "distractores" (opciones incorrectas pero plausibles) es una ventaja significativa sobre las plantillas.

4.5.3 Tarea Fotográfica

Propósito pedagógico: Fomentar la observación activa, la búsqueda de elementos específicos en el entorno y la aplicación práctica del conocimiento.

Estructura JSON:

```
{
  "question": "string"
}
```

La estructura es simple, ya que la "respuesta" no es un texto, sino un fichero de imagen que el usuario debe capturar y subir a través de la aplicación cliente.

4.5.4 Tarea de Sí o No (Yes or No)

Propósito pedagógico: Evaluar el conocimiento preciso sobre un hecho específico. Su simplicidad lo hace ideal para repasos rápidos o para iniciar una discusión más profunda.

Estructura JSON:

```
{
  "question": "string",
  "solucion": boolean
}
```

El uso de un tipo de dato booleano (`true`/`false`) para la solución asegura una evaluación automática robusta y sin ambigüedades.

4.6 Conclusiones

La fase de diseño ha sido fundamental para establecer una arquitectura robusta y coherente que dé soporte a los objetivos del proyecto. Siguiendo una metodología ágil, se ha definido un plan de trabajo claro que ahora transita hacia la etapa de implementación. Los resultados de esta fase no solo han delineado la estructura lógica del sistema, sino que también han sentado las bases para su funcionalidad, escalabilidad y usabilidad.

El diseño se centra en un servidor monolítico que encapsula la lógica de negocio, la gestión del conocimiento mediante la técnica RAG y la orquestación de modelos de lenguaje locales a través de Ollama. Una pieza clave de este diseño es la API REST, que actúa como interfaz estandarizada y desacoplada. Esta decisión de diseño, inspirada en la modularidad, asegura que el backend pueda ser consumido por una variedad de clientes futuros (aplicaciones web, móviles, etc.) sin necesidad de que estos conozcan la complejidad subyacente de los procesos de IA. Las interacciones mediante un formato estandarizado como JSON, validadas con modelos Pydantic, garantizan la fiabilidad y la correcta interpretación de los datos, eliminando la ambigüedad y fragilidad asociadas a las salidas de texto libre.

Asimismo, se ha puesto un especial énfasis en el diseño de las tareas educativas. A diferencia de los métodos tradicionales basados en plantillas rígidas, este diseño aprovecha la flexibilidad de la IA Generativa para crear actividades dinámicas y contextualizadas. La especificación de diferentes tipos de tareas (desarrollo, opción múltiple, fotográficas, etc.) con estructuras JSON bien definidas asegura que el sistema no solo sea potente, sino también pedagógicamente versátil y fiable para su uso en un entorno de aprendizaje.

Con estas bases de diseño sólidamente establecidas, se ha creado una hoja de ruta precisa para la siguiente etapa. El Capítulo 5 Implementación, se centrará en materializar estos conceptos en un sistema funcional, abordando la configuración del entorno de servidor, la integración de los componentes del pipeline RAG y la conexión final con los

modelos de lenguaje para dar vida a la generación de tareas de aprendizaje ubicuo.

Capítulo 5

Implementación

En este capítulo se describe cómo se llevó a cabo la implementación del sistema, detallando las herramientas, bibliotecas y decisiones prácticas tomadas para materializar el diseño planteado. Se abordan el entorno de desarrollo, la construcción de la base de conocimiento con ChromaDB y SentenceTransformers, la integración del modelo de lenguaje local con Ollama, la creación de la API REST con Flask y la implementación concreta de los módulos funcionales, ilustrando con fragmentos de código relevantes.

5.1 Introducción

Definido el diseño de la aplicación distribuida, se procede ahora a la fase de implementación, la cual se abordará utilizando la metodología ágil Scrum. Esta elección facilita un desarrollo iterativo e incremental del sistema, permitiendo ajustes rápidos y mejoras continuas basadas en la retroalimentación de mis tutores en las diferentes reuniones.

En esta fase, se establecerán las tecnologías específicas necesarias para la construcción de cada módulo del sistema, diseñado para generar tareas de aprendizaje ubicuo basado en Modelos de Lenguaje de Gran Tamaño (LLM). Además, se detallarán los servicios externos esenciales para el correcto funcionamiento de la aplicación, incluyendo aquellos destinados a la identificación y gestión de usuarios, persistencia de datos y personalización de las tareas según las preferencias individuales de cada usuario.

La sección describirá en profundidad cómo se ha llevado a cabo la implementación del servidor, explicando su organización estructural y justificando la elección de los servicios externos integrados. También se detallará qué pruebas o información se usaron para la decisión del modelo de lenguaje usado.

Finalmente, se destacarán claramente las interacciones entre los diferentes componentes del sistema, mostrando cómo dichas interacciones garantizan el cumplimiento efectivo de los requisitos establecidos en la etapa de análisis (Capítulo 3).

5.2 Implementación del servidor

Las dos funciones principales del servidor donde se ejecuta el sistema son: ser capaz de ejecutar modelos de lenguaje de forma local y obtener la información relevante para la generación de tareas. El lenguaje escogido para llevar a cabo el desarrollo del servidor ha sido **Python** en su versión 3.9.21, debido a la versatilidad y herramientas que ofrece para el trabajo con modelos de lenguaje. El código ha sido desarrollado en el entorno de VSCode (Visual Studio Code), un editor ligero, multiplataforma y altamente extensible. VSCode ofrece resaltado de sintaxis para Python, depuración integrada con puntos de interrupción y consola interactiva, terminal incorporada y soporte para ambientes virtuales. Además, cuenta con integración nativa con control de versiones Git, un marketplace de extensiones muy activo y gran capacidad de personalización. Para gestionar el entorno de paquetes se usó Conda, que es un gestor de entornos virtuales que permite aislar las dependencias de las librerías y así evitar conflictos entre los diferentes paquetes usados en el desarrollo del código.

La implementación de la API REST, aunque tendrá diferencias con la diseñada en el Capítulo 4 por falta de tiempo, se implementará usando el *framework* Flask en su versión más actualizada (3.1.1). Flask es un *framework* que proporciona una serie de funciones para el despliegue de una API REST de forma rápida y sencilla; también permite el uso de plantillas (*templates*) HTML para realizar pruebas más visuales del intercambio de datos entre el cliente y el servidor. El *framework* de Flask puede integrarse con otro *framework* llamado Flasgger, que ofrece funciones para documentar la API.

La estructura del código usada para el servidor web estará condicionada a los recursos que se van a exponer de la API REST. En el listado se puede observar la organización de los ficheros que conforman el código del servidor del sistema. Las dependencias necesarias para ejecutar el servidor están en un fichero llamado `requirements.txt`, donde en cada línea aparece el nombre de un paquete y, de forma opcional, se puede añadir la versión del paquete.

Para el control de versiones se empleó **Git**, con un repositorio alojado en GitHub. El uso de Git sirve para llevar un registro de los cambios, experimentar en ramas separadas sin afectar la versión estable y colaborar de manera organizada.

```
src/
    templates/
        index.html
    app.py
    corrector.py
    eleccion.py
    long_simple.py
    MCQ.py
    photograph.py
    yes_or_no.py
```

Código 5.1: Estructura del código fuente del sistema

5.3 Implementación de la Base de Conocimiento (RAG)

El núcleo de la funcionalidad RAG se implementó utilizando ChromaDB [Hong et al., 2025] para el almacenamiento vectorial y SentenceTransformers para la generación de *embeddings*.

5.3.1 Generación de *embeddings* con SentenceTransformers

La conversión de texto a vectores semánticos (*embeddings*) es un paso crucial. Para ello, se empleó el modelo `all-MiniLM-L6-v2` de la librería **SentenceTransformers**. Este modelo fue seleccionado por su excepcional equilibrio entre la calidad de los *embeddings* generados y su eficiencia computacional. Al ser un modelo ligero y rápido, resulta ideal para aplicaciones interactivas donde los tiempos de respuesta son críticos, sin sacrificar significativamente la precisión en tareas de similitud semántica, incluso con textos en español.

5.3.2 Fragmentación de Texto con LangChain

Para la etapa de fragmentación de texto (*chunking*), se utilizó el componente *RecursiveCharacterTextSplitter* de la librería **LangChain**. Aunque no se empleó la librería para orquestar todo el *pipeline*, esta herramienta es particularmente efectiva por su estrategia de división jerárquica. Intenta preservar la cohesión semántica del texto al priorizar la división por separadores naturales (párrafos, saltos de línea) antes de recurrir a la división por caracteres.

```python
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain.schema import Document

def dividir_en_fragmentos(document, tamano_fragmento=500, solapamiento=75):
    """
    Divide el contenido de un documento en fragmentos más pequeños
    utilizando una estrategia recursiva para mantener la cohesión semántica.
    """
    # Se inicializa el divisor con una lista de separadores priorizados.
    # Intenta dividir primero por párrafos, luego por líneas, etc.
    splitter = RecursiveCharacterTextSplitter(
        separators=["\n\n", "\n", " ", ""],
        keep_separator=False,
        is_separator_regex=False,
        chunk_size=tamano_fragmento,
        chunk_overlap=solapamiento # Solapamiento para no perder contexto entre fragmentos
    )

    # Se divide el contenido textual del documento
    fragmentos_texto = splitter.split_text(document.page_content)

    # Se crean nuevos objetos Document por cada fragmento, conservando
    # los metadatos del documento original (ej. la fuente).
    # Esto es vital para la posterior citación de fuentes.
```

```
    fragmentos_docs = [
        Document(page_content=frag, metadata=document.metadata)
        for frag in fragmentos_texto
    ]

    return fragmentos_docs
```

Código 5.2: Función para dividir un documento en fragmentos cohesivos.

5.3.3 Almacenamiento y Consulta con ChromaDB

Como base de datos vectorial, se eligió **ChromaDB** por su simplicidad, su capacidad para ejecutarse localmente y su modo de almacenamiento persistente. Esta elección permite indexar los *embeddings* de los documentos y recuperarlos eficientemente mediante búsquedas de similitud, garantizando que la información persista entre reinicios del servicio.

```
import chromadb

# Se inicializa un cliente persistente. Los datos se guardan en el directorio
# especificado, sobreviviendo a los reinicios del proceso.
chroma_client = chromadb.PersistentClient(path="/path/to/your/chroma_db")

def get_user_collection(api_key):
    """
    Crea o recupera una colección específica para cada usuario.
    Esto implementa un aislamiento de datos, asegurando
    que cada usuario solo acceda a sus propios documentos.
    """
    collection_name = f"user_{api_key}"
    return chroma_client.get_or_create_collection(name=collection_name)
```

Código 5.3: Inicialización del cliente persistente y gestión de colecciones en ChromaDB.

La función de búsqueda, pieza central del sistema RAG, se encarga de recibir la consulta del usuario, convertirla en un *embedding* y utilizarlo para encontrar los fragmentos de texto más relevantes en la colección del usuario dentro de ChromaDB. Los resultados se formatean como un contexto enriquecido para ser enviado al LLM.

```
def buscar_info_chromadb(api_key, query, embedding_model):
    """
    Busca en la colección de un usuario los fragmentos más relevantes para una consulta.
    """
    # 1. Recuperar la colección aislada del usuario
    user_collection = get_user_collection(api_key)

    # 2. Convertir la consulta del usuario en un vector (embedding)
    # Se debe usar el MISMO modelo que se usó para indexar los documentos.
    query_embedding = embedding_model.encode([query])

    # 3. Realizar la búsqueda por similitud en ChromaDB
    # n_results determina cuántos fragmentos relevantes se recuperarán.
```

```
    results = user_collection.query(
        query_embeddings=query_embedding,
        n_results=3
    )

    if not results or not results["documents"]:
        return "No se encontraron documentos relevantes."

    # 4. Formatear los resultados para inyectarlos en el prompt del LLM
    # Esta técnica de "prompt engineering" ayuda al LLM a entender el contexto.
    documentos = results["documents"][0]
    metadatos = results["metadatas"][0]
    contexto = []

    for i, (doc, meta) in enumerate(zip(documentos, metadatos)):
        # Se extrae la fuente de los metadatos guardados previamente
        fuente = meta.get("source", "Fuente desconocida")
        # Se envuelve cada fragmento en etiquetas para una mejor delimitación
        contexto.append(f"<INI_CONTEXT{i}>Fuente: {fuente} - Contenido: {doc}</END_CONTEXT{i}>\n")

    return " ".join(contexto)
```

Código 5.4: Función de búsqueda de información por similitud en ChromaDB.

5.4 Modelo de Lenguaje (LLM) y su Integración con Ollama

La selección de los modelos de lenguaje (LLM) constituye una decisión crítica que impacta directamente en la calidad y fiabilidad del sistema. Durante la fase de implementación, se llevó a cabo un proceso de evaluación empírica para determinar los modelos más adecuados para las dos tareas principales del sistema: la generación de preguntas y la corrección automática de respuestas. El análisis no solo consideró la capacidad de los modelos, sino también el equilibrio entre el rendimiento, la fiabilidad y los recursos computacionales requeridos. Además, la decisión estuvo respaldada por los resultados de los *benchmarks* de razonamiento y facticidad presentados en la Tabla 5.1.

- **Modelos Utilizados**:

  Inicialmente, las pruebas se centraron en modelos más ligeros, como `llama3.1:8b`. La hipótesis de partida era que un modelo de menor tamaño ofrecería una velocidad de inferencia superior y un menor consumo de recursos (RAM/VRAM), factores deseables para un sistema que podría enfrentarse a múltiples peticiones simultáneas. Sin embargo, aunque estos modelos demostraron ser rápidos, exhibieron dos limitaciones críticas que comprometían los objetivos del proyecto:

  – **Fiabilidad de la salida estructurada:** La adherencia al formato de salida JSON, forzado mediante esquemas Pydantic, no era consistente. Con frecuen-

cia, el modelo no lograba generar una estructura JSON válida, lo que provocaba errores e impedía el procesamiento automático de las tareas generadas.

– **Calidad del contenido y alucinaciones:** Se observó una mayor propensión a generar alucinaciones. En una herramienta con fines educativos, la precisión y la veracidad del contenido son requisitos no prescindibles.

Estas limitaciones evidenciaron que, para este caso de uso, la fiabilidad y la calidad del contenido debían priorizarse sobre la velocidad de inferencia. En consecuencia, la evaluación se orientó hacia modelos de mayor tamaño y especialización, lo que condujo a la adopción de la siguiente estrategia, confirmada también por los resultados comparativos obtenidos en los *benchmarks*:

– **Generación de tareas:** Se seleccionó el modelo `gemma3:27b`. Este modelo de Google demostró un rendimiento superior en la comprensión de instrucciones complejas, alcanzando una tasa de éxito cercana al 100 % en la generación de salidas JSON válidas y conformes al esquema. Además, redujo significativamente la propensión a las alucinaciones, produciendo preguntas coherentes y relevantes.

– **Corrección automática:** Se optó por el modelo `deepseek-r1:32b`, especializado en razonamiento. La corrección de respuestas abiertas requiere comparar semánticamente la respuesta del estudiante con una solución de referencia, identificar discrepancias y articular retroalimentación constructiva. En las pruebas, `deepseek-r1:32b` ofreció evaluaciones más precisas y justificaciones más detalladas que los modelos de propósito general.

Tabla 5.1: Benchmark de **razonamiento** (GSM8K) y **facticidad** (TruthfulQA).

Modelo	**GSM8K**	**TruthfulQA**
Llama 3.1 8B Instruct	82.0	54.5
Gemma 3 27B Instruct	92.1	62.2
DeepSeek-R1 Distill Qwen 32B	82.7	58.4

En conclusión, la implementación finaliza con una arquitectura de modelos especializada. A pesar del mayor coste computacional que suponen `gemma3:27b` y `deepseek-r1:32b`, su adopción se justifica plenamente por la drástica mejora en la fiabilidad, precisión y calidad general del sistema, tal como reflejan los resultados de los *benchmarks*, asegurando así el cumplimiento de los requisitos pedagógicos y funcionales del proyecto.

- **Integración**: La librería `ollama` para Python permite interactuar con los modelos como si se tratara de una llamada a una función. Una característica clave utilizada fue la capacidad de forzar la salida en formato JSON, pasando el esquema de un modelo **Pydantic** a la función de chat. Esto garantiza que la salida del LLM sea siempre estructurada y válida, eliminando la necesidad de un complejo post-procesamiento.

```python
from ollama import chat
from pydantic import BaseModel

# Se define la estructura de la respuesta esperada con Pydantic
class Quest(BaseModel):
    question: str
    solucion: str

class Test(BaseModel):
  quests: list[Quest]

# En la función generadora...
def generar_long_simple(...):
    # ... (construcción de system_message y user_message) ...

    response = chat(
        messages=[system_message, user_message],
        model="gemma3:27b",
        # Se pasa el esquema JSON del modelo Pydantic
        format=Test.model_json_schema(),
    )

    # Se valida y parsea la respuesta JSON directamente en un objeto Pydantic
    quests = Test.model_validate_json(response.message.content)
    return quests
```

Código 5.5: Llamada al LLM con formato JSON forzado mediante Pydantic.

Esta técnica de salida estructura en JSON es una de las implementaciones más elegantes del proyecto, asegurando robustez y fiabilidad en la comunicación con el LLM.

5.5 Análisis Detallado de la API y el Flujo de Datos

El fichero `app.py` constituye el cerebro de la aplicación, orquestando todas las interacciones entre el usuario, la base de conocimiento, y los modelos de lenguaje. En esta sección, se desglosa su funcionamiento interno, desde la configuración inicial hasta la lógica de cada uno de sus *endpoints* principales.

5.5.1 Configuración Inicial y Dependencias

El arranque de la aplicación define el entorno y carga los componentes esenciales.

- **Inicialización de Flask y Swagger:** Se crea la instancia de la aplicación Flask y se integra con Flasgger para la generación automática de documentación interactiva (Swagger UI), facilitando las pruebas y la comprensión de la API.
- **Configuración de Rutas:** Se establecen las rutas para el almacenamiento de ficheros subidos por los usuarios (`UPLOAD_FOLDER`) y para las imágenes generadas (`IMG_FOLDER`). Es crucial que cada usuario tenga su propio directorio, garantizando la privacidad y el aislamiento de los datos.

- **Carga de Modelos y Base de Datos:** Se inicializan los dos componentes clave del *pipeline* RAG:
  - `chromadb.PersistentClient`: Se conecta a la base de datos vectorial persistente. Al especificar una ruta en el disco, se asegura que todos los vectores y metadatos almacenados no se pierdan entre reinicios del servidor.
  - `SentenceTransformer(all-MiniLM-L6-v2)`: Se carga en memoria el modelo de *embeddings*. Aunque esto consume recursos de RAM/VRAM, permite una vectorización extremadamente rápida de los textos sin necesidad de llamadas a una API externa.
- **Gestión de API Keys:** Se utiliza un diccionario simple de Python (`API_KEYS`) para la autenticación. Cada petición a un *endpoint* protegido debe incluir una clave válida en la cabecera `X-API-KEY`.

5.5.2 El Proceso RAG: De la Ingesta a la Recuperación

El sistema implementa un ciclo RAG completo. La lógica para manejar este ciclo está distribuida en varias funciones de utilidad y un *endpoint* dedicado.

Ingesta y Procesamiento de Fuentes de Datos

El sistema es agnóstico a la fuente de datos, pudiendo procesar múltiples formatos. Para cada formato, existe una función de ingesta especializada:

- `ingest_text_file`, `ingest_pdf_file`, `ingest_docx_file`: Estas funciones se encargan de abrir y extraer el contenido textual de ficheros `.txt`, `.pdf` y `.docx`, respectivamente. Utilizan librerías como `PyPDF2` y `docx2txt` para la extracción.
- `ingest_web_page`: Recibe una URL, realiza una petición HTTP con `requests` para obtener el HTML y utiliza `BeautifulSoup` para limpiar el contenido, extrayendo únicamente el texto visible y eliminando etiquetas, scripts y estilos.
- `ingest_youtube_transcript`: Para URLs de YouTube, extrae el ID del vídeo y utiliza la librería `youtube-transcript-api` para obtener la transcripción automática (en español o inglés). Esto permite procesar como texto el contenido de un vídeo.

Todas estas funciones devuelven el texto extraído encapsulado en un objeto `Document` de LangChain, que convenientemente asocia el contenido con sus metadatos (como la ruta del fichero o la URL de origen).

Endpoint de Carga: `/api/cargar_info`

Este es el punto de entrada para añadir nuevo conocimiento a la base de datos de un usuario. Su flujo de trabajo es el siguiente:

1. **Autenticación**: Verifica la `X-API-KEY` del usuario.
2. **Recepción de Datos**: Acepta datos a través de un formulario `multipart/form-data`, que puede contener un fichero (`file`) o una URL (`URL`).

3. **Selección de Ingestor**: Basado en si se proporcionó un fichero o una URL (y el tipo de cada uno), llama a la función de ingesta apropiada (`ingest_pdf_file`, `ingest_web_page`, etc.).

4. **Fragmentación (Chunking)**: El texto completo obtenido es pasado a la función `dividir_en_fragmentos`, que utiliza el `RecursiveCharacterTextSplitter` de LangChain para dividirlo en trozos más pequeños y manejables (por defecto, de 500 caracteres con 75 de solapamiento). Esto es fundamental para que la búsqueda semántica sea precisa.

5. **Vectorización y Almacenamiento**: Itera sobre cada fragmento de texto:
   - Genera su *embedding* vectorial usando el modelo `embedding_model.encode()`.
   - Crea un ID único para el fragmento.
   - Prepara los metadatos, que incluyen la clave del usuario y la fuente original del documento.
   - Llama a `user_collection.add()` para guardar en lote los fragmentos, sus vectores, IDs y metadatos en la colección de ChromaDB específica del usuario.

6. **Respuesta**: Devuelve un mensaje de éxito o un error detallado si alguna de las fases falla.

Recuperación de Contexto: `buscar_info_chromadb`

Esta función es el corazón de la fase de recuperación semántica del RAG. Se invoca cada vez que se necesita contexto para responder una pregunta.

```
def buscar_info_chromadb(api_key, query):
    # 1. Obtener la coleccion especifica del usuario
    user_collection = get_user_collection(api_key)

    if not isinstance(query, str):
        return "Error: La consulta debe ser una cadena (de texto)."

    # Validacion para no buscar si el usuario no tiene documentos
    user_files_path = f"/home/lfsanchez/tfg-2024-luisfran/files/{api_key}"
    if not directory_has_files(user_files_path):
        return "No se encontraron documentos en ChromaDB."

    # 2. Convertir la consulta del usuario en un vector
    embedding_response = embedding_model.encode([query])

    # 3. Realizar la busqueda de similitud en ChromaDB
    # Se buscan los 3 fragmentos mas parecidos (top-k=3)
    results = user_collection.query(
        query_embeddings=embedding_response,
        n_results=3
    )

    if not results or not results["documents"] or not results["documents"][0]:
```

```
24          return "No se encontraron documentos relevantes en ChromaDB."
25
26      # 4. Formatear los resultados para el prompt del LLM
27      documentos = results["documents"][0]
28      metadatos = results["metadatas"][0]
29      contexto = []
30      for i, (doc, meta) in enumerate(zip(documentos, metadatos)):
31          fuente = meta.get("source", "Fuente desconocida")
32          # Se envuelven en etiquetas para que el LLM los identifique
    mejor
33          contexto.append(f"<\INI_CONTEXT{i}>Fuente: {fuente} - Contenido:
    {doc}<\END_CONTEXT{i}>\n")
34
35      return " ".join(contexto)
```

Código 5.6: Función para la búsqueda de contexto relevante en ChromaDB.

El proceso es claro: vectoriza la pregunta del usuario y la usa para encontrar en ChromaDB los fragmentos de texto previamente almacenados cuyo contenido semántico sea más similar. El resultado es una cadena de texto formateada que servirá como "memoria a corto plazo"para el LLM.

5.5.3 Endpoints de la API

Además de la carga de información, la API expone otros recursos para gestionar el ciclo de vida de los datos y generar las tareas.

Endpoint de Generación: `/api/generar_tarea`

Este es el *endpoint* más complejo y el que materializa el objetivo final de la aplicación.

1. **Validación y Parámetros**: Recibe una petición POST con un cuerpo JSON que contiene todos los parámetros del test: tipo de pregunta, tema, cantidad, dificultad, puntos de interés (POIs), etc.

2. **Recuperación de Contexto (RAG)**: Si se proporcionan `poi1` o `poi2`, llama a `buscar_info_chromadb` para cada uno. El contexto recuperado se concatena y se pasará al LLM. Si el usuario no ha subido ficheros, esta búsqueda no se realiza y el contexto estará basado únicamente en la información de Wikipedia que se obtiene dentro de los módulos de generación.

3. **Despacho Dinámico**: Llama a la función `eleccion()`, que actúa como un enrutador. Basándose en el parámetro `tipo_preguntas`, importa dinámicamente el módulo Python correspondiente (ej: `long_simple.py`) y ejecuta su función generadora, pasándole todos los parámetros necesarios.

4. **Generación de Material Visual**: Si se solicita material con imágenes (`material="texto y imagenes"`), invoca a la función `descargar_imagenes`. Esta función utiliza la API de Wikipedia y Wikimedia Commons para buscar y descargar imágenes relacionadas con el `poi1`, guardándolas en la carpeta del usuario.

5. **Composición de la Respuesta**: El resultado del módulo de generación (un objeto Pydantic) se convierte en un diccionario. Se le añade la lista de imágenes descargadas y se devuelve todo como una respuesta JSON.

Endpoints de Gestión de ficheros

Para que el usuario pueda administrar su base de conocimiento, se implementaron dos utilidades:

- `/api/subir_fichero`: Un *endpoint* POST que permite al usuario identificado subir diferentes tipos de fuentes de informacion ya sean ficheros de texto(.txt,.pdf,.doc,...) o una URL a una pagina web o video de youtube.
- `/api/listar_ficheros`: Un *endpoint* GET simple que devuelve una lista con los nombres de todos los ficheros que el usuario ha subido.
- `/api/eliminar_fichero`: Este *endpoint* GET es crucial para la consistencia de los datos. No solo elimina el fichero físico del servidor, sino que también realiza una operación de borrado en ChromaDB. Para ello, consulta en la colección del usuario todos los fragmentos cuyos metadatos apunten al fichero que se va a eliminar, recopila sus IDs y los pasa al método `user_collection.delete()`. Esto asegura que el contexto obsoleto no contamine futuras búsquedas.

Endpoint de Corrección: `/api/corregir`

Este recurso permite cerrar el ciclo de evaluación. Recibe la pregunta original, la respuesta correcta generada por el sistema, y la respuesta proporcionada por un alumno. Delega toda la lógica a la función `corregir_respuesta` del módulo `corrector.py`, que utiliza un LLM para comparar semánticamente ambas respuestas y determinar si la del alumno es correcta, ofreciendo además una justificación.

5.6 Consideraciones sobre la Implementación de la Aplicación Cliente

Si bien el alcance de este Trabajo de Fin de Grado se ha centrado en el diseño e implementación de la arquitectura del servidor y su API REST, resulta pertinente postular cómo debería ser una aplicación cliente que consuma dichos servicios. La robustez y flexibilidad de la API desarrollada permiten desacoplar completamente la lógica del backend de su presentación, haciendo que el sistema sea agnóstico a la tecnología del cliente. A continuación, se describen las funcionalidades clave y las tecnologías recomendadas para su desarrollo.

5.6.1 Funcionalidades Clave del Cliente

Una aplicación cliente funcional debería estructurarse en torno a dos perfiles de usuario principales: el docente y el estudiante, ofreciendo interfaces diferenciadas para cada uno.

Perfil de Docente

- **Autenticación**: La aplicación debe proporcionar una interfaz segura para que el docente introduzca y almacene su `X-API-KEY`.
- **Gestión de la Base de Conocimiento:**
  - Un formulario para subir ficheros (`.pdf`, `.txt`, `.docx`) o introducir URLs, que interactuaría con el endpoint `/api/cargar_info`.
  - Una vista para listar los ficheros ya procesados (obtenidos de `/api/listar_ficheros`) con la opción de eliminarlos (`/api/eliminar_fichero`).
- **Generación de Tareas:**
  - Un formulario de generación de tareas que permita al usuario configurar todos los parámetros soportados por la API: tipo de pregunta, tema, cantidad, dificultad, nivel educativo, puntos de interés (POI), etc.
  - Una vista para mostrar de forma clara y ordenada las tareas generadas por el sistema tras una llamada a `/api/generar_tarea`, permitiendo al docente revisarlas antes de su uso.

Perfil de Estudiante

- **Visualización de Tareas**: Una interfaz limpia para presentar el enunciado de la tarea, ya sea una pregunta de desarrollo, una de opción múltiple, una de verdadero/falso o una tarea fotográfica.
- **Interacción y Respuesta:**
  - Un campo de texto para respuestas abiertas.
  - Botones de opción (radio buttons) para las preguntas de opción múltiple.
  - Un interruptor (toggle) para las preguntas de verdadero/falso.
  - Integración con la cámara del dispositivo para capturar y subir imágenes en las tareas fotográficas.
- **Retroalimentación**: Una vez enviada la respuesta (mediante una llamada a `/api/corregir`), la interfaz debe ser capaz de mostrar la retroalimentación recibida del servidor, diferenciando visualmente si la respuesta fue correcta o no y mostrando el texto explicativo proporcionado por el LLM.

5.6.2 Pila Tecnológica Recomendada

Dada la naturaleza del sistema, se postulan dos posibles paradigmas de implementación para el cliente:

- **Aplicación Web Progresiva (PWA):** Sería la opción más versátil y de más rápida distribución.

  - *Frameworks:* Se recomienda el uso de un framework de JavaScript moderno como React, Vue o Angular. React, con su ecosistema de librerías como Axios para las llamadas a la API y React Router para la navegación, sería una elección especialmente sólida.
  - *Estilos:* Para agilizar el desarrollo de la interfaz, se podría utilizar una librería de componentes como Material-UI o un framework de CSS como Tailwind CSS.
  - *Ventajas:* Multiplataforma por defecto (accesible desde cualquier navegador en escritorio o móvil) y facilidad de despliegue.

- **Aplicación Móvil Nativa (iOS/Android):** Esta opción ofrecería una mejor integración con el *hardware* del dispositivo (GPS para una geolocalización más precisa, cámara nativa) y la posibilidad de un funcionamiento offline más avanzado.
  - *Frameworks Multiplataforma:* Para optimizar los recursos de desarrollo, se recomienda un framework como React Native o Flutter. Ambos permiten desarrollar para iOS y Android desde una única base de código, interactuando con la API REST del sistema de la misma manera que lo haría una aplicación web.
  - *Ventajas:* Mejor rendimiento y experiencia de usuario en dispositivos móviles, acceso completo a las capacidades del *hardware* y visibilidad en las tiendas de aplicaciones.

5.7 Conclusiones

La fase de implementación ha servido para materializar con éxito el diseño arquitectónico planteado, resultando en un sistema funcional y robusto. La elección de un ecosistema de herramientas de código abierto, centrado en Python, ha sido un factor clave para acelerar el desarrollo y garantizar la modularidad del proyecto.

El uso de frameworks y bibliotecas especializadas como Flask, LangChain, ChromaDB y Ollama permitió centrar los esfuerzos en la lógica principal de la aplicación —la orquestación del pipeline RAG y la generación de tareas— en lugar de tener que desarrollar desde cero soluciones para la gestión del servidor, el almacenamiento vectorial o la inferencia de modelos. De manera análoga a cómo servicios externos pueden acelerar el desarrollo, este enfoque basado en un ecosistema maduro permitió construir una solución potente y segura en un tiempo reducido, aprovechando la fiabilidad de componentes probados por la comunidad. La decisión de ejecutar los LLM localmente con Ollama y gestionar los embeddings con SentenceTransformers y ChromaDB fue una elección estratégica deliberada, priorizando la privacidad de los datos y el control total sobre el entorno, una ventaja fundamental frente a las soluciones propietarias basadas en la nube.

La elección de Python como lenguaje principal, junto con Flask para el servidor web, se ha demostrado especialmente acertada. Dado que el sistema se comunica constantemente mediante objetos JSON, la integración con bibliotecas como Pydantic para forzar y validar esquemas de salida directamente desde el LLM ha sido una de las decisiones de implementación más importantes. Esto no solo simplificó el procesamiento de las respuestas del modelo, sino que también garantizó una comunicación sin errores entre el servidor

y cualquier cliente potencial, eliminando la fragilidad inherente al parseo de texto libre.

Como contrapartida, esta arquitectura no está exenta de desafíos. La principal desventaja radica en la gestión de los recursos computacionales, ya que la ejecución local de grandes modelos de lenguaje es intensiva en memoria (RAM/VRAM) y capacidad de procesamiento. Además, la dependencia de múltiples bibliotecas de código abierto exige una cuidadosa gestión de versiones para evitar conflictos de compatibilidad a largo plazo.

En definitiva, aunque se han identificado áreas de mejora, la implementación actual valida de manera efectiva el diseño propuesto y constituye una base sólida y funcional. El sistema está ahora listo para ser evaluado en la siguiente fase, donde se analizará su rendimiento y la calidad de las tareas generadas en el Capítulo 6, "Análisis de Resultados".

Capítulo 6

Análisis de Resultados

En este capítulo se presentan los resultados obtenidos tras la implementación y las pruebas del sistema. A diferencia de un sistema en producción con usuarios reales, los datos presentados provienen de una serie de pruebas controladas y ejecutadas sobre el *hardware* específico del proyecto para garantizar la reproducibilidad de las mediciones. De los registros generados, se han extraído los tiempos de respuesta promedio y los picos de consumo energético para los escenarios de uso más representativos: la generación simple de tareas, la generación utilizando RAG y la generación compleja sin modificar el contexto. Primeramente, este capítulo presenta un análisis cualitativo que compara las tareas generadas con las de sistemas anteriores como Casual Learn, evaluando así la mejora en la calidad y complejidad pedagógica. Posteriormente se ejemplifican con las tareas usadas en la sección anterior como funcionaria el sistema de corrección con casos de respuestas verdaderas y falsas. Y por ultimo se presentan los resultados cuantitativos, que, aunque validan la funcionalidad del sistema, destacan la inferencia del LLM como el principal cuello de botella, con altas latencias en los casos más complejos, lo que lo hace más adecuado para un uso asíncrono.

6.1 Introducción

Una vez completada la fase de implementación descrita en el capítulo anterior, este capítulo se dedica a la presentación y análisis sistemático de los resultados obtenidos. El propósito es llevar a cabo una evaluación integral del sistema desarrollado desde una doble perspectiva: por un lado, validar su viabilidad técnica y su rendimiento computacional y, por otro, medir la calidad y el valor pedagógico de las tareas generadas en comparación con el estado del arte previo Además del valor añadido de la realimentacion automática.

Para ello, la evaluación se aborda desde tres ejes fundamentales. En primer lugar, se presenta una comparación cualitativa con metodologías anteriores, contrastando las tareas generadas por nuestro enfoque basado en IA Generativa con las producidas por sistemas basados en plantillas y LOD, como Casual Learn. A través de ejemplos concretos, este análisis busca mostar al lector las difrencias entre ambas metodologías.

En segundo lugar, se muestra un ejemplo de los resultados obtenidos en los ejemplos de tareas anteriores con casos de respuestas correctas y incorrectas poniendo a prueba la capacidad del modelo con los datos proporcionados de corregir errores o dar retroalimen-

tación adecuada.

En ultimo lugar, se realiza un análisis cuantitativo del rendimiento del sistema propuesto en el servidor utilizado para las pruebas, donde se examinan métricas clave como el tiempo de respuesta y el consumo de recursos bajo cargas de trabajo representativas. Este análisis es fundamental para identificar los cuellos de botella inherentes a la ejecución local de los modelos de lenguaje y para determinar los escenarios de uso más adecuados para el sistema.

Conjuntamente, estos análisis permitirán no solo validar el cumplimiento de los objetivos planteados en este Trabajo de Fin de Grado, sino también contextualizar sus contribuciones, identificar sus limitaciones y sentar una base empírica sólida para las conclusiones y las futuras líneas de trabajo que se expondrán posteriormente.

6.2 Comparación con las tareas de Casual Learn

Para evaluar la contribución de este proyecto, es fundamental compararlo no con un LLM de propósito general, sino con el estado del arte previo en la generación de tareas de aprendizaje ubicuo, representado por sistemas como Casual Learn [Ruiz-Calleja et al., 2022]. Como presentamos en el Capítulo 2 los sistemas se basan en la explotación de LOD mediante un paradigma de plantillas predefinidas.

Nuestro sistema, en cambio, adopta un paradigma generativo. Utiliza la información fáctica obtenida de RAG y Wikipedia no para rellenar huecos, sino como contexto para un LLM que genera la tarea completa desde cero. Esta diferencia fundamental en el enfoque da lugar a resultados cualitativamente distintos, como se ilustra en las Tablas 6.1, 6.2, 6.3, 6.4. Para los ejemplos, se utiliza como contexto la Catedral de Valladolid y la Catedral de Palencia

Además, este enfoque ofrece ventajas adicionales frente a las preguntas generadas con Casual Learn. Por un lado, la actualización de la información en la web de datos se refleja automáticamente en las preguntas gracias al uso de RAG, mientras que en Casual Learn sería necesaria la intervención manual de un desarrollador o administrador para regenerar los ítems. Por otro lado, las preguntas de Casual Learn presentan una menor riqueza expresiva, ya que se generan a partir de plantillas que reproducen siempre un mismo formato para cada tipo de cuestión; en cambio, en la propuesta aquí presentada las preguntas de un mismo tipo pueden adoptar formulaciones diversas, más cercanas al lenguaje natural. El precio a pagar por estas ventajas es la posible aparición de alucinaciones, así como ciertas limitaciones de rendimiento inherentes al uso de modelos generativos.

Tabla 6.1: Comparativa Identificación (Foto): Casual Learn (*Plantilla LOD*) vs. Sistema Propuesto (*IA Generativa*)

	Casual Learn	**Sistema Propuesto**
Contexto	Catedral de Valladolid.	Catedral de Valladolid.
Enunciado	Haz una foto de algún elemento de estilo Renacentista que encuentres en la Catedral de Valladolid.	Mirad con atención la fachada. ¿Podéis fotografiar algún detalle que os recuerde que esta Catedral empezó a construirse hace más de 400 años? ¡Buscad algo que os hable del pasado y de la historia de Valladolid!
Detalle	*Instrucción genérica sin foco pedagógico.*	*Ejercicio de "caza del tesoro" con clave interpretativa.*

Tabla 6.2: Comparativa Texto Abierto: Casual Learn (*Plantilla LOD*) vs. Sistema Propuesto (*IA Generativa*)

	Casual Learn	**Sistema Propuesto**
Contexto	Catedral de Valladolid.	Catedral de Valladolid.
Enunciado	¿Qué parte de Catedral de Valladolid crees que es la más antigua? ¿Por qué?	Fíjate en las diferentes partes de la fachada de la Catedral: ¿observas elementos decorativos? Describe al menos dos de ellos y piensa en qué estilo arquitectónico crees que encajan. ¿Qué te sugieren estos detalles sobre las ideas o creencias de la época en la que se construyó?
Detalle	*Pregunta genérica, respuesta factual*	*Observación dirigida y análisis estético.*

En Casual Learn no existen tareas previamente elaboradas con plantillas sobre la Catedral de Palencia ni sobre la Catedral de Valladolid, ni en formato verdadero o falso ni en formato de opción múltiple. Por ello, con el fin de ejemplificar cómo son las actividades que genera el nuevo sistema, a continuación se presentan dos ejemplos de cada tipo.

Tabla 6.3: Ejemplo de preguntas de Verdadero o falso (*IA Generativa*)

	Sistema Propuesto	**Sistema Propuesto**
Contexto	Catedral de Palencia.	Catedral de Valladolid.
Enunciado	Verdadero o Falso: Si caminas alrededor del edificio, ¿podrás identificar claramente un estilo arquitectónico uniforme en todas sus partes?	Verdadero o Falso: ¿Se puede observar, a simple vista, alguna construcción o vestigio de una iglesia anterior que haya sido parcialmente destruida para dar paso a la Catedral?
Solución	*Respuesta esperada: falso.* *(Razón: La catedral combina elementos de diferentes estilos (visigodo, románico, gótico, renacentista, barroco, neoclásico). Observando con atención, los alumnos podrán identificar diferentes estilos arquitectónicos en diferentes partes del edificio.)*	*Respuesta esperada: verdadero.* *(Razón: La información proporcionada menciona que la Catedral se construyó junto a una colegiata anterior, y que algunos de sus espacios fueron destruidos para continuar con las nuevas obras. Si observas atentamente, podrás identificar restos o vestigios de la antigua iglesia, especialmente en las cercanías de la Catedral.*

Tabla 6.4: Ejemplo de preguntas de opción múltiple (*IA Generativa*)

	Sistema Propuesto	**Sistema Propuesto**
Contexto	Catedral de Palencia.	Catedral de Valladolid.
Enunciado	Fíjate en la planta de la catedral (si es visible en el suelo o en algún plano informativo). ¿Cuántos cruceros diferentes puedes distinguir? ¿Qué forma particular le da esto a la planta de la catedral?	Presta atención a los elementos decorativos que hay en la portada principal (esculturas, relieves, etc.). ¿Qué temas o personajes parecen representarse en ellos?
Opciones	a) Un solo crucero, lo que da a la planta una forma de cruz simple. b) Dos cruceros, lo que hace que la planta sea más compleja y parezca una cruz patriarcal. c) Tres cruceros, formando una planta muy elaborada. d) No puedo ver la planta de la catedral. *Opción Correcta b.*	a) Escenas de la vida cotidiana de la época medieval. b) Animales fantásticos y criaturas mitológicas. c) Figuras religiosas, como santos, ángeles o escenas bíblicas. d) Escudos de armas y símbolos de la nobleza local. *Opción correcta c.*

La comparación pone de manifiesto que el sistema propuesto supera las limitaciones de los enfoques basados en plantillas predefinidas, como Casual Learn. Mientras que estos se restringen a la inserción de datos fácticos en estructuras rígidas, nuestro enfoque generativo permite elaborar enunciados más ricos, contextualizados e interpretativos.

Las actividades resultantes no se limitan a constatar hechos, sino que fomentan la observación, la reflexión y la conexión con el trasfondo histórico y cultural, lo que amplía su potencial pedagógico. Asimismo, la integración de técnicas de RAG con un LLM ofrece flexibilidad para generar distintos formatos de tarea (verdadero/falso, opción múltiple, desarrollo abierto, actividades de identificación), incluso en contextos para los que no existían plantillas previas.

En conjunto, el cambio de paradigma abre nuevas posibilidades para el diseño de experiencias de aprendizaje ubicuo, más cercanas a las necesidades del alumnado y a los objetivos educativos, sin embargo, las tareas de la nueva propuesta aún no han sido evaluadas por profesores a diferencia de las preguntas de plantillas de LOD.

6.3 Ejemplificación del uso del corrector

Tras comparar el desempeño del nuevo sistema con el sistema anterior, a continuación **ejemplificamos** cómo sería la corrección automática de distintos tipos de tareas por parte del modelo de lenguaje. Para cada tipo presentamos *dos casos*: una respuesta bien resuelta y otra con errores, con el fin de verificar si el modelo identifica adecuadamente cada situación y analizar la *retroalimentación* que proporciona.

6.3.1 Corrección de tareas de texto abierto

Caso 1 (respuesta correcta). *Usuario: Luis (Secundaria). Punto de interés: Catedral de Valladolid.*

Pregunta (desarrollo). *Fíjate en las diferentes partes de la fachada de la Catedral: ¿observas elementos decorativos? Describe al menos dos de ellos y piensa en qué estilo arquitectónico crees que encajan. ¿Qué te sugieren estos detalles sobre las ideas o creencias de la época en la que se construyó?*

Respuesta del alumno. Veo pilastras y frontones en la portada principal; encajan con un estilo clasicista/herreriano. Sugieren sobriedad y la autoridad de la Iglesia propia de la Contrarreforma.

Evaluación del modelo. ✓ Correcta.

Retroalimentación generada.

> ¡Hola Luis! Tu respuesta es excelente y demuestra una buena comprensión de los elementos arquitectónicos y su contexto histórico. Has identificado correctamente las pilastras y los frontones en la portada de la Catedral de Valladolid, y has acertado al relacionarlos con el estilo clasicista/herreriano. Además, has conectado estos elementos decorativos con ideas de autoridad y la sobriedad característica de la Contrarreforma. Tu capacidad para observar los detalles

> arquitectónicos y relacionarlos con el contexto histórico es impresionante. ¡Sigue así!

Comentario. El modelo *reconoce con precisión* la corrección de la respuesta y justifica por qué es adecuada, reforzando los aciertos con referencias al estilo y al contexto (Contrarreforma).

Caso 2 (respuesta incorrecta). *Usuario: Luis (Secundaria). Punto de interés: Catedral de Valladolid.*

Pregunta (desarrollo). *(Misma que el caso anterior).*

Respuesta del alumno. Veo gárgolas y tracerías flamígeras; diría que es gótico tardío. Me sugiere fantasía medieval y superstición.

Evaluación del modelo. ✗ Incorrecta.

Retroalimentación generada.

> Hola Luis, ¡muy bien por intentar identificar elementos decorativos en la fachada de la Catedral de Valladolid! Observaste correctamente las gárgolas y las tracerías flamígeras, que son características del estilo gótico. Sin embargo, la Catedral de Valladolid, aunque iniciada en el gótico, presenta una fachada principal de estilo clasicista/herreriano. Los elementos clave que debías haber identificado son las pilastras y los frontones clásicos en la portada, así como la sobriedad que caracteriza este estilo. La diferencia principal es que tu respuesta se centra en elementos góticos presentes en otras partes de la catedral, mientras que la pregunta se refiere específicamente a la fachada principal. El estilo clasicista/herreriano, presente en la portada, se caracteriza por la búsqueda de la armonía, la proporción y la sobriedad, lo que refleja las ideas de autoridad y racionalidad de la época de la Contrarreforma. Aunque la 'fantasía medieval y superstición' que mencionas pueden asociarse al gótico, la fachada clasicista/herreriana buscaba transmitir una imagen de solidez y poder, acorde con los valores de la Contrarreforma. Te recomiendo que revises imágenes de la fachada principal de la Catedral de Valladolid y compares los elementos decorativos con ejemplos del estilo clasicista y herreriano. Puedes buscar información sobre estos estilos en páginas web de historia del arte o en libros de texto.

Comentario. El modelo *detecta correctamente* el error de foco (confundir la fachada principal con otros elementos del conjunto) y ofrece una explicación contrastiva útil, además de sugerir recursos para revisar.

6.3.2 Corrección de verdadero/falso

Caso 3 (respuesta correcta). *Usuario: Luis (Secundaria). Punto de interés: Catedral de Valladolid.*

Pregunta (V/F). *¿Se puede observar, a simple vista, alguna construcción o vestigio de una iglesia anterior que haya sido parcialmente destruida para dar paso a la Catedral?*

Respuesta esperada. Verdadero.

Respuesta del alumno. Verdadero.

Evaluación del modelo. ✓ Correcta.
Retroalimentación generada.

> ¡Excelente, Luis! Tu respuesta es correcta. Efectivamente, al observar la Catedral de Valladolid, es posible identificar restos de la antigua iglesia de San Pedro el Real integrada en la estructura de la nueva catedral. Se pueden apreciar vestigios de muros y arcos románicos que formaban parte de la iglesia original, incorporados al edificio gótico. Esto es un ejemplo interesante de cómo las construcciones religiosas a menudo se levantan sobre estructuras preexistentes, reflejando la historia y evolución de la ciudad. ¡Sigue así!

Comentario. Además de validar el acierto, el modelo añade *contexto histórico* que enriquece el aprendizaje (integración de estructuras previas).

Caso 4 (respuesta incorrecta). *Usuario: Luis (Secundaria). Punto de interés: Catedral de Valladolid.*
Pregunta (V/F). *(Misma que el caso anterior).*
Respuesta esperada. Verdadero.
Respuesta del alumno. Falso.
Evaluación del modelo. ✗ Incorrecta.
Retroalimentación generada.

> Hola Luis, ¡buen intento! La respuesta correcta es verdadero. En la Catedral de Valladolid, si observas con atención, especialmente en el lado del Evangelio (el lado derecho si estás mirando hacia la fachada principal), puedes ver restos de la iglesia románica de San Pedro, que fue integrada parcialmente en la construcción de la catedral. No se demolió por completo, sino que se incorporaron partes de sus muros y elementos arquitectónicos. Esto es bastante común en la construcción de catedrales góticas, que a menudo se construían sobre cimientos de iglesias anteriores. Te sugiero que busques imágenes de la Catedral de Valladolid que muestren este detalle para que puedas visualizarlo. Puedes buscar en internet 'Catedral de Valladolid vestigios iglesia San Pedro' o consultar alguna guía turística o página web sobre la catedral. ¡Sigue practicando y explorando la arquitectura!

Comentario. La corrección es clara y específica (*localiza* el vestigio en el lado del Evangelio) y propone una estrategia de verificación visual, lo que favorece la autonomía del alumno.

6.3.3 Corrección de opción múltiple

Caso 5 (respuesta correcta). *Usuario: Luis (Secundaria). Punto de interés: Catedral de Valladolid.*
Pregunta (opción múltiple). *Presta atención a los elementos decorativos que hay en la portada principal (esculturas, relieves, etc.). ¿Qué temas o personajes parecen representarse en ellos?*
a) Escenas de la vida cotidiana de la época medieval.

b) Animales fantásticos y criaturas mitológicas.
c) Figuras religiosas, como santos, ángeles o escenas bíblicas.
d) Escudos de armas y símbolos de la nobleza local.

Respuesta esperada. c) Figuras religiosas, como santos, ángeles o escenas bíblicas.
Respuesta del alumno. c) Figuras religiosas, como santos, ángeles o escenas bíblicas.
Evaluación del modelo. ✓ Correcta.
Retroalimentación generada.

> ¡Hola Luis, excelente trabajo! Tu respuesta es completamente correcta. La portada principal de la Catedral de Valladolid está ricamente decorada con figuras religiosas, representando santos, ángeles y escenas bíblicas. Esto es típico en la arquitectura gótica, que buscaba transmitir enseñanzas religiosas a través del arte. Al identificar correctamente estos temas, demuestras una buena comprensión del propósito y simbolismo de la decoración en las catedrales medievales. ¡Sigue así!

Comentario. El modelo valida y explica el *porqué* de la opción correcta, conectando forma y función (didáctica religiosa del arte).

Caso 6 (respuesta incorrecta). *Usuario: Luis (Secundaria). Punto de interés: Catedral de Valladolid.*

Pregunta (opción múltiple). *(Misma que el caso anterior).*
Respuesta esperada. c) Figuras religiosas, como santos, ángeles o escenas bíblicas.
Respuesta del alumno. a) Escenas de la vida cotidiana de la época medieval.
Evaluación del modelo. ✗ Incorrecta).
Retroalimentación generada.

> Hola Luis, he revisado tu respuesta y veo que has elegido 'Escenas de la vida cotidiana de la época medieval'. Aunque la vida medieval es un tema interesante, la decoración de la portada principal de la Catedral de Valladolid se centra principalmente en figuras religiosas. Los relieves y esculturas de la portada no representan escenas del día a día, sino personajes bíblicos, santos y ángeles. Esto es muy común en las catedrales góticas, que buscaban enseñar la fe cristiana a través del arte. Piensa que en la Edad Media, la mayoría de la gente no sabía leer, así que las imágenes eran una forma importante de comunicar las historias de la Biblia. Te recomiendo que revises imágenes de la portada de la Catedral de Valladolid y prestes atención a las figuras representadas. Busca imágenes de santos, ángeles o escenas de la Biblia. Esto te ayudará a comprender mejor el objetivo de la decoración de la portada. Puedes buscar en sitios web de historia del arte o en guías turísticas de Valladolid. ¡Sigue practicando y no te desanimes! El arte y la historia pueden ser complejos, pero con esfuerzo y dedicación, podrás comprenderlo mejor.

Comentario. El modelo corrige de forma *explicativa* (no sólo dicotómica), y añade una breve justificación pedagógica (función narrativa de la iconografía).

6.4 Evaluación del rendimiento del servidor

Para validar la viabilidad técnica del sistema, se llevó a cabo una evaluación cuantitativa del rendimiento. El objetivo fue medir la eficiencia y capacidad de respuesta de la arquitectura bajo cargas de trabajo representativas, así como identificar los posibles cuellos de botella inherentes a la ejecución local de los modelos de lenguaje.

Las pruebas se ejecutaron en un servidor del grupo de investigacion GSIC/EMIC con capacidad para este proyecto, cuyas especificaciones técnicas son las mostradas en la tabla 6.5:

Componente	**Detalle**
Servidor:	HPE ProLiant DL380 Gen10
CPU:	Intel Xeon-S 4210 (10 núcleos @ 2.20 GHz)
RAM:	64 GB DDR4 RDIMM
GPU:	2 × NVIDIA Quadro P5000 (32 GB VRAM en total)
Almacenamiento:	HPE 1.8 TB SAS 10K RPM HDD
Software:	Python 3.9, Flask, Ollama 0.1.32, ChromaDB 0.4.24

Tabla 6.5: Especificaciones del servidor

Se definieron dos métricas clave:

Tiempo de Respuesta (Latencia): Tiempo total en segundos desde que el servidor recibe una petición HTTP hasta que envía la respuesta completa. Esta métrica es necesaria para determinar si el sistema puede actuar de forma síncrona con los estudiantes.

Consumo de Recursos: Se monitorizó el pico de uso de las GPUs en Watios que nos servirá para calcular el coste económico en kWh del servidor en una petición al sistema.

Se diseñaron tres escenarios de prueba para simular los casos de uso principales. Cada prueba se ejecutó 10 veces, registrando el valor promedio para asegurar la consistencia de los resultados.

- **Escenario A (Generación Simple):** Solicitud de generación de 5 tareas de opción múltiple (MCQ) sobre el "Arte Herreriano" sin contexto de RAG de la Catedral de Valladolid.
- **Escenario B (Generación con RAG):** Solicitud de generación de 5 tareas de opción múltiple (MCQ) sobre el "Arte Herreriano" de la Catedral de Valladolid , utilizando el sistema RAG para recuperar contexto relevante de un documento de 5 páginas previamente indexado.
- **Escenario C (Generación Compleja):** Solicitud de generación de 5 tareas de opción múltiple (MCQ) sobre el "Arte Herreriano" de la Catedral de Valladolid recuperando y añadiendo todo el contexto recuperado en el *prompt*.

Escenario de Prueba	Tiempo de Respuesta (s)	Consumo (kWh)
A: Generación Simple	43	0.00175
B: Generación con RAG	42	0.0017
C: Generación Compleja	365	0.00577

Tabla 6.6: Resultados promedio de tiempo de respuesta y consumo de recursos por escenario de prueba.

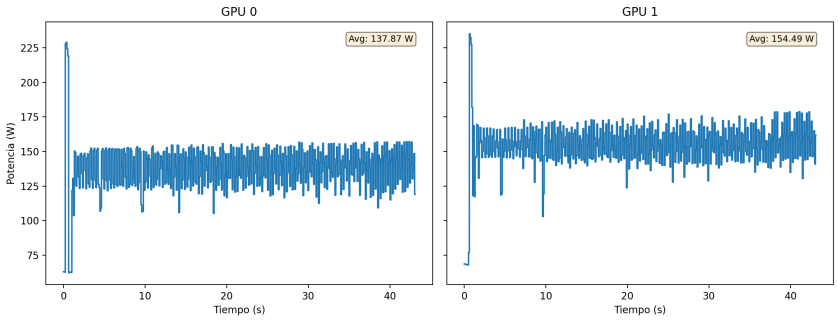

Figura 6.1: Gráfica de consumo frente al tiempo de respuesta del caso A

Cuello de Botella en la Inferencia del LLM. Los resultados mostrados en la tabla 6.6 confirman que el principal factor limitante del sistema es el tiempo de inferencia de los modelos de lenguaje. Con tiempos de respuesta que van desde los 42 s hasta los 365 s, el sistema no es apto para interacciones en tiempo real, pero sí es perfectamente válido para casos de uso asíncronos, como la preparación de material educativo por parte de un docente o de forma autodidacta por parte de un alumno.

Impacto del contexto RAG y la Complejidad. El Escenario C es, con diferencia, el más lento. Este incremento se debe a la combinación de dos factores: la sobrecarga del sistema RAG (vectorización de la consulta y búsqueda en ChromaDB) y, sobre todo, el mayor tiempo que necesita el modelo `gemma3:27b` para procesar un *prompt* enriquecido con más contexto y generar tres respuestas extensas en lugar de una sola.

Consumo de recursos y requisitos de *hardware*. El consumo de VRAM es el factor más crítico. El sistema utiliza casi la totalidad de los 32 GB disponibles, lo que demuestra que la elección de modelos de 27B y 32B parámetros lleva el *hardware* a su límite. Este alto consumo es un "coste fijo", ya que los modelos se cargan en memoria una sola vez y permanecen allí para atender peticiones sucesivas. El consumo de RAM, aunque significativo, se mantiene en niveles manejables gracias a la eficiencia de ChromaDB y Flask.

Implicaciones para la escalabilidad. Las pruebas, realizadas de forma secuencial, indican que el servidor puede gestionar las peticiones de un único usuario de manera funcional. Sin embargo, como establece el requisito no funcional RNF03, la concurrencia es una limitación. Dado el alto consumo de recursos y los tiempos de latencia, el sistema en

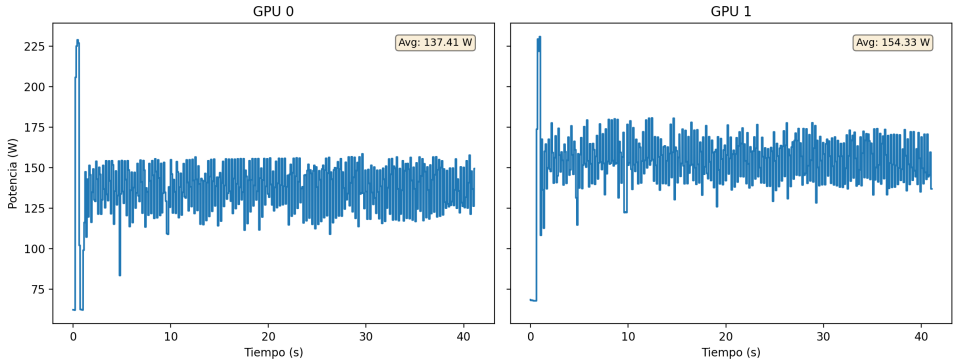

Figura 6.2: Gráfica de consumo frente al tiempo de respuesta del caso B

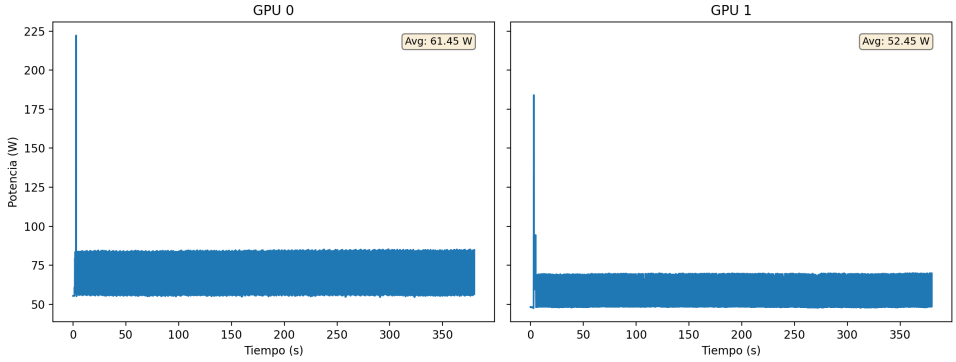

Figura 6.3: Gráfica de consumo frente al tiempo de respuesta del caso C

su estado actual podría atender a un número muy limitado de usuarios simultáneos (2–5) antes de que las colas de peticiones degradasen la experiencia de forma inaceptable. Una escalabilidad real requeriría una infraestructura de despliegue más robusta (p. ej., con balanceo de carga) y, potencialmente, la optimización de los modelos mediante técnicas de cuantización más agresivas o destilación.

La evaluación de rendimiento demuestra que el sistema es funcional sobre el *hardware* especificado y valida las decisiones de diseño. Sin embargo, también subraya que la latencia y el consumo de VRAM son los principales desafíos técnicos, definiendo el sistema como una potente herramienta de autor para uso individual o en grupos pequeños, y sentando las bases para futuras optimizaciones. Como posibles soluciones, se podría recurrir a APIs de terceros que disponen de una infraestructura optimizada para LLMs de mayor capacidad, aunque con un coste por token, o bien invertir en GPUs más potentes que permitan reducir tanto la latencia como los problemas asociados al consumo de memoria.

6.5 Conclusiones

Del análisis presentado en este capítulo se pueden extraer varias conclusiones fundamentales sobre la viabilidad, el rendimiento y el valor pedagógico del sistema desarrollado. La evaluación de rendimiento demuestra que el sistema es funcional sobre el *hardware* especificado y valida la viabilidad técnica de la arquitectura monolítica escogida. Sin embargo, también subraya que la latencia de la inferencia y el alto consumo de VRAM son los principales desafíos técnicos, confirmando que el cuello de botella del sistema reside en la ejecución de los modelos de lenguaje.

Desde una perspectiva cualitativa, la comparativa con sistemas basados en plantillas LOD como Casual Learn revela la contribución más significativa de este proyecto. Mientras que los sistemas de plantillas se limitan a la recuperación y presentación de datos fácticos, nuestro enfoque generativo transforma esos datos en un contexto para crear tareas pedagógicamente más ricas y complejas. Esto se manifiesta en la capacidad de generar distractores plausibles en preguntas de opción múltiple, formular preguntas abiertas que fomentan la observación y el análisis crítico, y, en general, promover habilidades cognitivas de orden superior en lugar de la mera memorización.

Además el sistema presenta un valor añadido de poder corregir o dar una retroalimentación de forma automática al estudiante de su respuesta y con los motivos por los cuales está bien o mal la respuesta para que le resulte más fácil el aprendizaje.

Estas conclusiones definen claramente el nicho de aplicación del sistema en su estado actual. No se trata de una herramienta de interacción en tiempo real para estudiantes, aunque pudieran existir casos donde si se pueda usar de forma síncrona, sino de una potente herramienta de autor para docentes o diseñadores instruccionales que pueden generar material de alta calidad de forma asíncrona. Las pruebas de rendimiento, realizadas de forma secuencial, indican que la escalabilidad para usuarios concurrentes es una limitación importante, en línea con el requisito no funcional RNF03. El sistema, por tanto, está optimizado para un uso individual o en grupos muy pequeños.

En definitiva, el conjunto de los análisis realizados permite afirmar que se ha cumplido el objetivo principal de este Trabajo de Fin de Grado: demostrar que es posible superar las limitaciones pedagógicas de los sistemas basados en plantillas y datos abiertos enlazados mediante el uso de Inteligencia Artificial Generativa, el siguiente paso sería hacer validación de la calidad de las tareas con docentes. Aunque la sensación que transmite el sistema es que a cambio de un mayor coste computacional y una latencia superior, se obtiene un salto cualitativo en la riqueza, variedad y eficacia pedagógica de las tareas de aprendizaje ubicuo generadas.

Capítulo 7

Conclusiones y líneas futuras de trabajo

En este capítulo final del Trabajo Fin de Grado (TFG) se relacionan los objetivos que motivaron su desarrollo con los resultados obtenidos, destacando el uso innovador de Modelos de Lenguaje de Gran Escala (LLM) para la generación y corrección de tareas de aprendizaje ubicuo. Este trabajo ha permitido explorar cómo estas tecnologías pueden generar contenido educativo contextualizado y personalizado, integrando el aprendizaje en el entorno cotidiano del usuario. Sin embargo, durante el desarrollo se han identificado limitaciones que no se habían considerado inicialmente. Estas limitaciones, junto con los logros alcanzados, sirven como base para proponer líneas de trabajo futuro que amplíen y optimicen el sistema.

7.1 Conclusiones del trabajo realizado

El objetivo principal de este TFG fue explorar el potencial de los LLM para generar tareas educativas en el marco del aprendizaje ubicuo, utilizando texto de alta calidad y Datos Abiertos Enlazados como fuentes de información. Los resultados de la generación de tareas y la retroalimentación de las mismas muestran que los LLM son altamente efectivos cuando se les proporciona texto rico y estructurado, produciendo tareas diversas y adaptadas al contexto del usuario. Este hallazgo es especialmente relevante en el aprendizaje ubicuo, donde la personalización y la relación con el entorno son fundamentales para una experiencia educativa inmersiva. Por ejemplo, las tareas generadas a partir de textos descriptivos sobre monumentos han demostrado ser más ricas y variadas que las producidas por métodos tradicionales basados en plantillas.

Otro aspecto crítico identificado es la escalabilidad del sistema. Aunque los experimentos realizados demostraron la viabilidad del enfoque en entornos controlados, el *hardware* actual no es suficiente para procesar grandes volúmenes de datos o responder a consultas complejas en tiempo real. Esta limitación sugiere la necesidad de optimizar tanto los recursos computacionales como los algoritmos utilizados para garantizar un rendimiento adecuado en escenarios de mayor escala.

En cuanto a las contribuciones, este TFG mejora significativamente los métodos ba-

sados en plantillas, que suelen generar tareas repetitivas y poco adaptadas. El enfoque propuesto introduce una mayor diversidad y contextualización, enriqueciendo la experiencia de aprendizaje, aunque sería necesaria la validación por parte de docentes y estudiantes para reforzar esta afirmación. Además, representa una novedad al ser, hasta donde sabemos, el primer intento de emplear LLM para generar contenido educativo ubicuo y el poder obtener retroalimentación de forma automática.

7.2 Limitaciones y líneas de trabajo futuro

A pesar de los avances logrados, el sistema presenta limitaciones que abren oportunidades para futuras mejoras. Una de las principales restricciones es su dependencia de texto de alta calidad. Para superar esto, una línea de trabajo futuro podría consistir en desarrollar técnicas de enriquecimiento de datos, integrando fuentes externas o generando descripciones textuales a partir de los datos semánticos disponibles.

Otra limitación es la especificidad del sistema, que actualmente se centra en temas relacionados con monumentos. Esto restringe su uso en otros contextos educativos. Una propuesta de mejora sería generalizar la aplicación para abarcar una gama más amplia de temas, transformándola en una herramienta educativa más versátil y adaptable a diferentes disciplinas.

En términos de infraestructura, la escalabilidad sigue siendo un desafío. Para abordar esta limitación, se podrían explorar mejoras en el *hardware*, como el uso de servidores más potentes, y optimizaciones algorítmicas que reduzcan los tiempos de procesamiento. Además, la integración con plataformas existentes, como CHEST, que también utiliza Datos Abiertos Enlazados, podría enriquecer el sistema al combinar múltiples fuentes de datos y mejorar la relevancia de las tareas generadas.

La implementación del diseño de la API REST del capítulo 4 sería otro trabajo para el futuro, que nos permita el almacenamiento persistente de las tareas completadas por los usuarios. Implementar esta funcionalidad permitiría a los estudiantes hacer un seguimiento de su progreso y retomar actividades previas, lo que contribuiría a una experiencia de aprendizaje más estructurada y continua.

Finalmente, la seguridad del sistema es un aspecto crítico que requiere atención. La versión actual no cuenta con medidas robustas para proteger los datos de los usuarios, lo que podría comprometer su privacidad. Una línea de trabajo futuro sería implementar protocolos de cifrado y autenticación más sólidos, garantizando la confidencialidad y la confianza en la plataforma, especialmente en entornos educativos.

En conclusión, este TFG ha establecido una base prometedora para el uso de LLM en el aprendizaje ubicuo, demostrando su capacidad para generar contenido educativo innovador y el *feedback* al estudiante sobre las respuestas al contenido. Las limitaciones identificadas no sólo destacan los retos pendientes, sino que también ofrecen una hoja de ruta clara para futuras investigaciones y desarrollos en este campo emergente.

Apéndice A

Documentación API

En las páginas siguientes se adjunta el código fuente de la API del sistema, desarrollado siguiendo los principios REST [Fielding, 2000]. La documentación interactiva de esta API fue generada automáticamente mediante la biblioteca Flasgger, que se integra con Flask para producir una especificación compatible con OpenAPI.

```
    openapi: 3.0.0

info:
  title: 'API de Generación y Corrección de Tests - L.F. System'
  version: '1.0.0'
  contact:
    name: 'Luis Francisco Sánchez'
    email: 'luisfrancisco.sanchez@estudiantes.uva.es'


tags:
  - name: 'Ficheros'
    description: 'Operaciones para gestionar los ficheros de
   conocimiento del usuario.'
  - name: 'Carga de Información'
    description: 'Endpoints para ingestar información desde ficheros o
   URLs en la base de datos vectorial.'
  - name: 'Generación de Tests'
    description: 'Endpoints para la creación dinámica de preguntas y
   tests.'
  - name: 'Corrección'
    description: 'Endpoints para evaluar las respuestas de los alumnos.'

components:
  securitySchemes:
    ApiKeyAuth:
      type: apiKey
      in: header
      name: X-API-KEY
      description: 'Clave API única para la autenticación del usuario.'

  schemas:

    TestGenerationRequest:
      type: object
```

```
33      description: 'Parámetros para generar un nuevo test.'
34      required:
35        - tipo_preguntas
36        - cantidad
37        - tema
38        - dificultad
39        - materia
40        - nivel
41      properties:
42        tipo_preguntas:
43          type: string
44          description: 'Tipo de pregunta a generar (ej: MCQ, long_simple
    , yes_or_no).'
45          example: 'MCQ'
46        cantidad:
47          type: integer
48          description: 'Número de preguntas a generar.'
49          example: 5
50        tema:
51          type: string
52          description: 'El tema principal sobre el que tratarán las
    preguntas.'
53          example: 'Catedrales'
54        dificultad:
55          type: string
56          description: 'Nivel de dificultad de las preguntas (ej: fácil,
     media, difícil).'
57          example: 'media'
58        materia:
59          type: string
60          description: 'La asignatura o campo de estudio.'
61          example: 'Historia del arte'
62        nivel:
63          type: string
64          description: 'El nivel educativo al que se dirige las tareas (
    ej: primaria (edad del estudiante), secundaria (edad del estudiante),
     universidad  (edad del estudiante), ...).'
65          example: 'secundaria'
66        poi1:
67          type: string
68          description: 'Punto de interés principal a buscar en la base
    de conocimiento para generar contexto.'
69          example: 'Catedral de Valladolid'
70        poi2:
71          type: string
72          description: 'Punto de interés secundario.'
73          example: 'Catedral de Palencia'
74        material:
75          type: string
76          description: 'Indica si se deben incluir imágenes en el test
    ("texto y imagenes").'
77          example: 'texto y imagenes'
78
79    TestGenerationResponse:
80      type: object
81      properties:
```

```
82          test:
83            type: array
84            description: 'Lista de preguntas generadas.'
85            items:
86              oneOf:
87                - $ref: '#/components/schemas/QuestionMCQ'
88                - $ref: '#/components/schemas/QuestionLongSimple'
89                - $ref: '#/components/schemas/QuestionYesNo'
90                - $ref: '#/components/schemas/QuestionPhotograph'
91          imagenes:
92            type: array
93            description: 'Lista de imágenes descargadas relacionadas con
    el test.'
94            items:
95              type: object
96              properties:
97                nombre:
98                  type: string
99                  example: 'catedral_de_valladolid_1.jpg'
100               ruta_local:
101                 type: string
102                 example: '/home/lfsanchez/tfg-2024-luisfran/files/user1/
    catedral_de_valladolid_1.jpg'
103               url_original:
104                 type: string
105                 example: 'https://commons.wikimedia.org/...'
106
107   QuestionMCQ:
108       type: object
109       required: [tipo, question, option0, option1, option2, option3,
    correct_option]
110       properties:
111         tipo: { type: string, example: 'MCQ' }
112         question: { type: string, example: '¿Qué estilo arquitectónico
    predomina en la Catedral de León, conocida como la "Pulchra Leonina
    "?' }
113         option0: { type: string, example: 'Románico' }
114         option1: { type: string, example: 'Barroco' }
115         option2: { type: string, example: 'Gótico' }
116         option3: { type: string, example: 'Herreriano' }
117         correct_option: { type: integer, example: 2 }
118
119     QuestionLongSimple:
120       type: object
121       required: [tipo, question, solucion]
122       properties:
123         tipo: { type: string, example: 'long_simple' }
124         question: { type: string, example: 'Describe las características
     principales de la fachada de Santa María de la Catedral de Burgos,
    mencionando sus elementos más significativos.' }
125         solucion: { type: string, example: 'La fachada de Santa María es
     un ejemplo del gótico francés. Está flanqueada por dos torres
    coronadas con agujas caladas del siglo XV. Sobre el arco de entrada
    se encuentra una galería con estatuas de los reyes de Castilla y un
    gran rosetón con una estrella de David en su tracería.' }
126
```

```
    QuestionYesNo:
      type: object
      required: [tipo, question, solucion]
      properties:
        tipo: { type: string, example: 'yes_or_no' }
        question: { type: string, example: '¿Se considera que la Catedral de Valladolid fue completada según los planos originales de Juan de Herrera?' }
        solucion: { type: boolean, example: false }

    QuestionPhotograph:
      type: object
      required: [tipo, question]
      properties:
        tipo: { type: string, example: 'photograph' }
        question: { type: string, example: 'Visita la Catedral Nueva de Salamanca y saca una foto del famoso astronauta esculpido en la Puerta de Ramos.' }

   \begin{lstlisting}[language=yaml]
CorrectionRequest:
  type: object
  required: [pregunta, respuesta_correcta, respuesta_alumno, nivel]
  properties:
    pregunta:
      type: string
      example: '¿En qué siglo se comenzó a construir la Catedral de Burgos?'
    respuesta_correcta:
      type: string
      example: 'La construcción de la Catedral de Burgos se inició en el siglo XIII, en el año 1221.'
    respuesta_alumno:
      type: string
      example: 'Empezó a construirse en el siglo XII.'
    nivel:
      type: string
      description: 'Nivel educativo para ajustar la rigurosidad de la corrección.'
      example: 'secundaria'

CorrectionResponse:
  type: object
  properties:
    is_ok:
      type: boolean
      description: 'True si la respuesta del alumno es correcta, False en caso contrario.'
    diferencias:
      type: string
      description: 'Explicación de por qué la respuesta es correcta o incorrecta, destacando las diferencias.'

    # ---- Modelos para Gestión de ficheros ----
    FileDeletionRequest:
      type: object
```

```
174       required: [fichero]
175       properties:
176         fichero:
177           type: string
178           description: 'Nombre del fichero a eliminar.'
179           example: 'historia_siglo_xx.pdf'
180
181     # ---- Modelo de Error Genérico ----
182     ErrorResponse:
183       type: object
184       properties:
185         error:
186           type: string
187           description: 'Mensaje de error descriptivo.'
188
189 paths:
190   /api/cargar_info:
191     post:
192       tags: [Carga de Información]
193       summary: 'Sube un fichero o carga desde una URL para procesar y
    almacenar.'
194       description: >
195         Permite ingestar información desde un fichero (txt, pdf, docx) o
     una URL (página web, vídeo de YouTube).
196         El contenido se divide en fragmentos, se vectoriza y se almacena
     en la base de datos ChromaDB del usuario.
197       security:
198         - ApiKeyAuth: []
199       requestBody:
200         required: true
201         content:
202           multipart/form-data:
203             schema:
204               type: object
205               properties:
206                 file:
207                   type: string
208                   format: binary
209                   description: 'El fichero a subir (txt, pdf, docx).'
210                 url:
211                   type: string
212                   description: 'URL de una página web o vídeo de YouTube
    .'
213                   example: 'https://es.wikipedia.org/wiki/
    Inteligencia_artificial'
214             encoding:
215               file:
216                 contentType: application/pdf, text/plain, application/
    vnd.openxmlformats-officedocument.wordprocessingml.document
217       responses:
218         '200':
219           description: 'Información procesada y almacenada exitosamente
    en ChromaDB.'
220           content:
221             application/json:
222               schema:
```

```
223                 type: object
224                 properties:
225                   message: { type: string, example: 'Información
    almacenada en ChromaDB' }
226         '400':
227           description: 'Error en la petición, como no proporcionar
    fichero ni URL, o un formato no soportado.'
228           content:
229             application/json: { schema: { $ref: '#/components/schemas/
    ErrorResponse' } }
230         '403':
231           description: 'La clave API proporcionada es inválida o no
    tiene permisos.'
232           content:
233             application/json: { schema: { $ref: '#/components/schemas/
    ErrorResponse' } }
234         '500':
235           description: 'Error interno del servidor durante el
    procesamiento o almacenamiento.'
236           content:
237             application/json: { schema: { $ref: '#/components/schemas/
    ErrorResponse' } }
238
239   /api/listar_ficheros:
240     get:
241       tags: [ficheros]
242       summary: 'Lista todos los ficheros de conocimiento de un usuario.'
243       security:
244         - ApiKeyAuth: []
245       responses:
246         '200':
247           description: 'Una lista con los nombres de los ficheros del
    usuario.'
248           content:
249             application/json:
250               schema:
251                 type: object
252                 properties:
253                   ficheros:
254                     type: array
255                     items: { type: string }
256                     example: ["documento1.pdf", "apuntes_clase.txt"]
257         '403':
258           description: 'Clave API inválida.'
259           content:
260             application/json: { schema: { $ref: '#/components/schemas/
    ErrorResponse' } }
261
262   /api/eliminar_fichero:
263     post:
264       tags: [ficheros]
265       summary: 'Elimina un fichero y sus fragmentos asociados de la base
     de datos.'
266       security:
267         - ApiKeyAuth: []
268       requestBody:
```

```
269         required: true
270         content:
271           application/json:
272             schema:
273               $ref: '#/components/schemas/FileDeletionRequest'
274       responses:
275         '200':
276           description: 'fichero y sus datos asociados eliminados
    correctamente.'
277           content:
278             application/json:
279               schema:
280                 type: object
281                 properties:
282                   message: { type: string, example: 'fichero documento.
    pdf eliminado correctamente' }
283         '403':
284           description: 'Clave API inválida.'
285         '404':
286           description: 'El fichero especificado no existe.'
287         '500':
288           description: 'Error interno al intentar eliminar el fichero o
    sus fragmentos.'
289
290   /api/generar_test:
291     post:
292       tags: [Generación de Tests]
293       summary: 'Genera un conjunto de preguntas de test basadas en pará
    metros específicos.'
294       description: >
295         Crea un test utilizando el modelo de lenguaje para generar
    preguntas. Puede usar el contexto
296         extraído de la base de conocimientos del usuario si se
    proporcionan "poi1" o "poi2".
297       security:
298         - ApiKeyAuth: []
299       requestBody:
300         required: true
301         content:
302           application/json:
303             schema:
304               $ref: '#/components/schemas/TestGenerationRequest'
305       responses:
306         '200':
307           description: 'Test generado exitosamente.'
308           content:
309             application/json:
310               schema:
311                 $ref: '#/components/schemas/TestGenerationResponse'
312         '400':
313           description: 'Parámetros inválidos, como una cantidad no numé
    rica.'
314         '403':
315           description: 'Clave API inválida.'
316
317   /corregir:
```

```
318     post:
319       tags: [Corrección]
320       summary: 'Corrige la respuesta de un alumno comparándola con la
    solución correcta.'
321       description: >
322         Utiliza un modelo de lenguaje para realizar una corrección semá
    ntica de la respuesta
323         de un alumno, proporcionando feedback sobre si es correcta y por
     qué.
324       security:
325         - ApiKeyAuth: []
326       requestBody:
327         required: true
328         content:
329           application/json:
330             schema:
331               $ref: '#/components/schemas/CorrectionRequest'
332       responses:
333         '200':
334           description: 'Resultado de la corrección.'
335           content:
336             application/json:
337               schema:
338                 $ref: '#/components/schemas/CorrectionResponse'
339         '400':
340           description: 'Faltan parámetros en la petición.'
341         '403':
342           description: 'Clave API inválida.'
```

Código A.1: Documentación de la API REST del servidor en formato YAML

Referencias

Banh, L., & Strobel, G. (2023). Generative artificial intelligence. *Electronic Markets*, *33*(1), 63. https://doi.org/10.1007/s12525-023-00680-1

Berners-Lee, T. (2006, 27 de julio). *Linked Data – Design Issues*. Consultado el 25 de junio de 2025, desde https://www.w3.org/DesignIssues/LinkedData

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language Models are Few-Shot Learners. En H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan & H.-T. Lin (Eds.), *Advances in Neural Information Processing Systems* (pp. 1877-1901, Vol. 33). Curran Associates, Inc.

Chellapilla, K., Puri, S., & Simard, P. (2006). High Performance Convolutional Neural Networks for Document Processing.

Chia, Y. K., Hong, P., Bing, L., & Poria, S. (2023). INSTRUCTEVAL: Towards Holistic Evaluation of Instruction-Tuned Large Language Models. https://arxiv.org/abs/2306.04757

Cui, P., & Sachan, M. (2023). Adaptive and Personalized Exercise Generation for Online Language Learning. https://arxiv.org/abs/2306.02457

Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, *36*, 10088-10115.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://arxiv.org/abs/1810.04805

Eager, B., & Brunton, R. (2023). Prompting higher education towards AI-augmented teaching and learning practice. *Journal of University Teaching and Learning Practice*, *20*(5), 1-19.

et al., D. (2024). DeepSeek-V3 Technical Report [Last revised 18 February 2025]. https://arxiv.org/abs/2412.19437

Fernández, M. d. (2023). La Inteligencia Artificial en Educación. Hacia un Futuro de Aprendizaje Inteligente. *Colección Estudios Culturales Serie Educación y Sociotecnociencia. https://dialnet. unirioja. es/descarga/libro/926431. pdf*.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* [Ph.D. Dissertation]. University of California, Irvine. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Fu, Z., Yang, H., So, A. M.-C., Lam, W., Bing, L., & Collier, N. (2023). On the effectiveness of parameter-efficient fine-tuning. *Proceedings of the AAAI conference on artificial intelligence*, *37*(11), 12799-12807.

Gallego-Lema, V. (2016, octubre). *El aprendizaje ubicuo en educación física en el medio natural: un estudio de caso* [Tesis doctoral].

Gandhi, M. (2024, 18 de abril). *Figure from: RAG 2.0: Finally Getting Retrieval-Augmented Generation Right?* [Figure on web page]. Cubed. Consultado el 5 de septiembre de 2025, desde https://cubed.run/blog/rag-2-0-finally-getting-retrieval-augmented-generation-right-05a067927589

García Zarza, P. (2021). *Diseño e implementación de una aplicación distribuida semántica para el apoyo del aprendizaje ubicuo relacionado con el patrimonio cultural* [Tesis de maestría, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad de Valladolid] [Trabajo Fin de Máster, Máster en Ingeniería de Telecomunicación].

García-Méndez, S., de Arriba-Pérez, F., & Somoza-López, M. d. C. (2025). A Review on the Use of Large Language Models as Virtual Tutors. *Science & Education*, *34*, 877-892. https://doi.org/10.1007/s11191-024-00530-2

Gemma Team. (2024). Gemma 2: Improving Open Language Models at a Practical Size [Last revised 2 October 2024]. https://arxiv.org/abs/2408.00118

Gholami, S., & Omar, M. (2023, septiembre). Do Generative Large Language Models need billions of parameters? [Version v1, 12 September 2023]. https://arxiv.org/abs/2309.06589v1

Giannakos, M., Azevedo, R., Brusilovsky, P., Cukurova, M., Dimitriadis, Y., Hernandez-Leo, D., Järvelä, S., Mavrikis, M., & Rienties, B. (2024). The promise and challenges of generative AI in education. *Behaviour & Information Technology*, *0*(0), 1-27. https://doi.org/10.1080/0144929X.2024.2394886

Gobierno de España. (2018). Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales [Boletín Oficial del Estado, núm. 294, 6 de diciembre de 2018].

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al. (2024). The LLaMA 3 Herd of Models [Last revised 23 November 2024]. https://arxiv.org/abs/2407.21783

Han, Y., Liu, C., & Wang, P. (2023). A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge. https://arxiv.org/abs/2310.11703

Herrera-Ortiz, J. J., Peña-Avilés, J. M., Herrera-Valdivieso, M. V., & Moreno-Morán, D. X. (2024). La inteligencia artificial y su impacto en la comunicación: recorrido y perspectivas. *Telos: Revista de Estudios Interdisciplinarios en Ciencias Sociales*, *26*(1), 278-296. https://doi.org/10.36390/telos261.18

Hong, K., Troynikov, A., Huber, J., & McGuire, M. (2025, abril). *Generative Benchmarking* (inf. téc.). Chroma. https://research.trychroma.com/generative-benchmarking

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). LoRA: Low-Rank Adaptation of Large Language Models. https://arxiv.org/abs/2106.09685

Hui, X., Reshef, O., & Zhou, L. (2024). The Short-Term Effects of Generative Artificial Intelligence on Employment: Evidence from an Online Labor Market. *Organization Science*, *35*(6), 1977-1989. https://doi.org/10.1287/orsc.2023.18441

Jiménez Cardona, N. (2023). Aplicación de la inteligencia artificial en la toma de decisiones jurisdiccionales (España). *REVISTA QUAESTIO IURIS*, *16*(3), 1612-1630. https://doi.org/10.12957/rqi.2023.74187

Kemker, R., McClure, M., Abitino, A., Hayes, T., & Kanan, C. (2018). Measuring Catastrophic Forgetting in Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1). https://doi.org/10.1609/aaai.v32i1.11651

Kim, J., Song, W., Kim, D., Kim, Y., Kim, Y., & Park, C. (2024). Evalverse: Unified and Accessible Library for Large Language Model Evaluation. https://arxiv.org/abs/2404.00943

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019, octubre). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension [Version v1]. https://arxiv.org/abs/1910.13461v1

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. En H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (pp. 9459-9474, Vol. 33). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

Li, Z., Yazdanpanah, V., Wang, J., Gu, W., Shi, L., Cristea, A. I., Kiden, S., & Stein, S. (2025). TutorLLM: Customizing Learning Recommendations with Knowledge Tracing and Retrieval-Augmented Generation. https://arxiv.org/abs/2502.15709

Longpre, S., Hou, L., Vu, T., Webson, A., Chung, H. W., Tay, Y., Zhou, D., Le, Q. V., Zoph, B., Wei, J., & Roberts, A. (2023). The Flan Collection: Designing Data and Methods for Effective Instruction Tuning. https://arxiv.org/abs/2301.13688

Mahabadi, R. K., Ruder, S., Dehghani, M., & Henderson, J. (2021). Parameter-efficient Multi-task Fine-tuning for Transformers via Shared Hypernetworks. https://arxiv.org/abs/2106.04489

Moradi, M., Yan, K., Colwell, D., Samwald, M., & Asgari, R. (2025). A critical review of methods and challenges in large language models. *Computers, Materials & Continua*, *82*(2), 1681-1698. https://doi.org/10.32604/cmc.2025.061263

Moreno-Izquierdo, L., Torres Penalva, A., et al. (2025). Inteligencia artificial y empleo: una reflexión aplicada al mercado laboral español.

Muñoz-Cristóbal, J. A., Jorrin-Abellan, I. M., Asensio-Perez, J. I., Martinez-Mones, A., Prieto, L. P., & Dimitriadis, Y. (2014). Supporting teacher orchestration in ubiquitous learning environments: A study in primary education. *IEEE Transactions on Learning Technologies*, *8*(1), 83-97.

Peñalver-Higuera, M. J., & Isea-Argüelles, J. J. (2024). Transformación hacia fábricas inteligentes: El papel de la IA en la industria 4.0. *Ingenium et Potentia*, *6*(10), 38-53. https://doi.org/10.35381/i.p.v6i10.3742

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, *21*, 1-67. https://doi.org/10.5555/3534678.3534816

Ruiz-Calleja, A., Bote-Lorenzo, M. L., Asensio-Pérez, J. I., Villagrá-Sobrino, S. L., Alonso-Prieto, V., Gómez-Sánchez, E., García-Zarza, P., Serrano-Iglesias, S., & Vega-Gorgojo, G. (2023). Orchestrating ubiquitous learning situations about Cultural

Heritage with Casual Learn mobile application. *International Journal of Human-Computer Studies*, *170*, 102959.

Ruiz-Calleja, A., García-Zarza, P., Vega-Gorgojo, G., Bote-Lorenzo, M. L., Gómez-Sánchez, E., Asensio-Pérez, J. I., Serrano-Iglesias, S., & Martínez-Monés, A. (2022). Casual learn: A linked data-based mobile application for learning about local cultural heritage. *Semantic Web*, *14*(2), 181-195.

Ruiz-Calleja, A., Vega-Gorgojo, G., Bote-Lorenzo, M. L., Asensio-Pérez, J. I., Dimitriadis, Y., & Gómez-Sánchez, E. (2021). Supporting contextualized learning with linked open data. *Journal of Web Semantics*, *70*, 100657.

Sierra, C. (2019). El impacto de la Inteligencia Artificial en nuestra sociedad. Retos y oportunidades [Consultado el 3 de abril de 2025]. https://www.csic.es/es/actualidad-del-csic/el-impacto-de-la-inteligencia-artificial-en-nuestra-sociedad-retos-y-oportunidades

UNESCO. (2023a). *El uso de la IA en la educación: decidir el futuro que queremos* [Consultado el 3 de abril de 2025]. Consultado el 3 de abril de 2025, desde https://www.unesco.org/es/articles/el-uso-de-la-ia-en-la-educacion-decidir-el-futuro-que-queremos

UNESCO. (2023b). Informe de seguimiento de la educación en el mundo, 2023 [Consultado el 24 de abril de 2025]. https://unesdoc.unesco.org/ark:/48223/pf0000388894

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J. G., Salakhutdinov, R., & Le, Q. V. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding. *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*, 5754-5764. https://papers.nips.cc/paper/8812-xlnet-generalized-autoregressive-pretraining-for-language-understanding

Zaken, E. B., Ravfogel, S., & Goldberg, Y. (2021). Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Zeng, A., Liu, M., Lu, R., Wang, B., Liu, X., Dong, Y., & Tang, J. (2023). AgentTuning: Enabling Generalized Agent Abilities for LLMs. https://arxiv.org/abs/2310.12823

Zhang, S., Dong, L., Li, X., Zhang, S., Sun, X., Wang, S., Li, J., Hu, R., Zhang, T., Wu, F., & Wang, G. (2024). Instruction Tuning for Large Language Models: A Survey. https://arxiv.org/abs/2308.10792