

Entidad: Cátedra ENIA-UPV (Universitat Politècnica de València), en colaboración con la ONG «Damos Nuestra Ilusión» (DNI), Valencia.

Periodo: 3 de enero ? 30 de junio de 2026.

Tutor en la entidad: Juan Miguel (Juanmi).

Proyecto: Diseño, desarrollo y despliegue de un asistente conversacional (chatbot RAG) para la ONG DNI.

1. Descripción concreta y detallada de las tareas, trabajos desarrollados y departamentos a los que he estado asignado

Durante las prácticas he estado integrado en la Cátedra ENIA-UPV, dentro del grupo de investigación en IA de la Universitat Politècnica de València, trabajando de forma coordinada con la ONG «Damos Nuestra Ilusión» (DNI) como entidad destinataria del sistema. El objetivo fue construir un asistente conversacional capaz de responder preguntas sobre los proyectos de voluntariado de la ONG (Desayunos Solidarios, charlas en residencias, refuerzo escolar, etc.) a partir de su documentación real.

El trabajo se organizó alrededor de un cambio de arquitectura de fondo. El sistema partía de una versión previa monolítica, en la que la lógica de recuperación de información, generación de respuestas y control de flujo estaba mezclada en un mismo bloque, difícil de probar y de mantener. Mi tarea principal fue rediseñar y reimplementar el sistema bajo una arquitectura hexagonal (puertos y adaptadores) combinada con un enfoque multiagente. Esto supuso:

- Definir la capa de dominio pura (entidades como Message, Conversation, Project, Chunk; y puertos abstractos: LLMPort, VectorStorePort, MemoryPort, EmbeddingPort, ConfigPort), sin dependencias de infraestructura.
- Implementar los adaptadores intercambiables detrás de esos puertos: ChromaDB para el almacén vectorial (búsqueda semántica, BM25 e híbrida), SentenceTransformers para los embeddings multilingües en español, y adaptadores de LLM para Ollama (modelos locales) y OpenAI como alternativa.
- Construir los agentes especializados con sus herramientas: un agente de recuperación (retrieval), uno de generación, un orquestador que coordina el flujo completo (cargar contexto ? clasificar proyecto ? recuperar ? generar ? validar ? guardar memoria) y un validador para detectar alucinaciones.
- Desarrollar las dos interfaces de usuario: una API REST con FastAPI y una web de chat propia, y un bot de Telegram conectado al mismo backend.
- Añadir persistencia real de conversaciones con PostgreSQL, de modo que el asistente recuerde el contexto entre turnos y entre reinicios del servicio.
- Desplegar el sistema en producción en un servidor de la UPV (grupo GTI), con servicios gestionados por systemd y un proxy inverso, y dejarlo accesible tanto por web como por Telegram.
- Diseñar e integrar un sistema de evaluación de usabilidad (cuestionario CUQ) con almacenamiento de resultados, para recoger feedback real de un piloto con usuarios.

Además de la implementación, elaboré la documentación técnica del proyecto (especificación, diseño, manual de usuario y manual de desarrollador) y una batería de tests unitarios y de integración que acompaña cada componente.

2. Valoración de las tareas desarrolladas con los conocimientos y competencias adquiridas en relación con los estudios universitarios

Las prácticas me han permitido aplicar de forma directa buena parte de lo estudiado en el grado y, sobre todo, integrarlo en un único proyecto real de principio a fin.

De las asignaturas de Ingeniería del Software apliqué los patrones de arquitectura limpia: la separación en capas, la inversión de dependencias y el patrón puertos/adaptadores fueron la columna vertebral del rediseño. Lo que en clase era un principio teórico («que el dominio no dependa de la infraestructura») se volvió una decisión con consecuencias muy concretas: gracias a ella pude cambiar el almacén vectorial o el modelo de lenguaje sin tocar la lógica de negocio, y pude testear el dominio con mocks en lugar de levantar servicios pesados.

De las asignaturas de Inteligencia Artificial y aprendizaje automático apliqué los conceptos de recuperación de información, embeddings, similitud semántica y modelos de lenguaje, que aquí se materializaron en un sistema RAG completo. De bases de datos, el diseño del esquema relacional para la persistencia de conversaciones en PostgreSQL. De programación y sistemas, el manejo de código asíncrono, la gestión de procesos con systemd, el despliegue tras un proxy y el trabajo por SSH sobre un servidor remoto.

También he reforzado competencias transversales: control de versiones con Git siguiendo convenciones de commits, escritura de tests como red de seguridad, y documentación técnica dirigida tanto a usuarios como a desarrolladores.

3. Relación de los problemas planteados y de los procedimientos seguidos para su resolución

A lo largo del proyecto surgieron varios problemas relevantes, que resolví de forma metódica (reproducir ? localizar la causa ? corregir ? verificar con un test):

- Pérdida aparente de memoria del asistente. El bot mezclaba proyectos y no recordaba turnos anteriores. Se atribuyó inicialmente a un fallo de persistencia, pero al investigar comprobé que PostgreSQL guardaba y cargaba correctamente. La causa real era que el historial de conversación se inyectaba dentro del mismo bloque que el contexto recuperado del RAG, y un endurecimiento del prompt («responde solo con información del contexto») hacía que el modelo tratara el historial como un dato no fiable y lo ignorara. Lo resolví separando el historial conversacional del contexto RAG en secciones distintas del prompt, con reglas propias. Fue un buen ejemplo de que el diagnóstico evidente no siempre es el correcto.

- Doble truncado de la ventana de contexto. El asistente olvidaba conversaciones de días anteriores. La causa era un recorte del historial aplicado dos veces (en el orquestador y de nuevo en el agente de generación), que dejaba fuera todo lo anterior al último turno. Lo corregí ampliando la ventana y eliminando el recorte redundante, con un test de regresión que reproduce el olvido.
- Timeout en la primera consulta (arranque en frío del modelo). La primera petición a un modelo grande en la GPU compartida tardaba mucho en cargar y provocaba errores de tiempo de espera. Lo resolví con un calentamiento del modelo al arrancar el servicio, un timeout específico para el arranque en frío y manteniendo el modelo en memoria.
- Mezcla de memoria entre usuarios. La memoria se compartía indebidamente entre conversaciones distintas. Lo corregí migrando la clave de la memoria a una clave compuesta por usuario y conversación.
- Seguridad frente a inyección de prompts. Detecté que las reglas escritas en el prompt no bastaban para evitar respuestas indebidas. Añadí sanitización de la salida en código (idioma, emojis, sufijos) además de endurecer el prompt, porque la lógica de seguridad debe ser determinista y auditable, no depender solo del modelo.
- Contexto perdido en referencias anafóricas y ordinales. El asistente perdía el hilo cuando el usuario decía «el tercero» o «ese proyecto». Lo resolví manteniendo el proyecto activo entre turnos y capturando correctamente el proyecto resuelto.

En todos los casos el procedimiento fue el mismo: reproducir el fallo, aislar la causa real (no la aparente), corregir con el mínimo cambio necesario, y dejar un test que evite la regresión.

4. Identificación de las aportaciones que, en materia de aprendizaje, han supuesto las prácticas

- De la teoría a las consecuencias reales de la arquitectura. Migrar de una arquitectura monolítica a una hexagonal me enseñó, con problemas concretos delante, por qué merece la pena la separación en capas: testabilidad, capacidad de sustituir infraestructura y facilidad para localizar dónde está cada responsabilidad.
- El ciclo completo («que el dominio no dependa de la infraestructura») se volvió una decisión con consecuencias muy concretas: gracias a ella pude cambiar el almacén vectorial o el modelo de lenguaje sin tocar la lógica de negocio, y pude testear el dominio con mocks en lugar de levantar servicios pesados.

De las asignaturas de Inteligencia Artificial y aprendizaje automático apliqué los conceptos de recuperación de información, embeddings, similitud semántica y modelos de lenguaje, que aquí se materializaron en un sistema RAG completo. De bases de datos, el diseño del esquema relacional para la persistencia de conversaciones en PostgreSQL. De programación y sistemas, el manejo de código asíncrono, la gestión de procesos con systemd, el despliegue tras un proxy y el trabajo por SSH sobre un servidor remoto.

También he reforzado competencias transversales: control de versiones con Git siguiendo convenciones de commits, escritura de tests como red de seguridad, y documentación técnica dirigida tanto a usuarios como a desarrolladores.

3. Relación de los problemas planteados y de los procedimientos seguidos para su resolución

A lo largo del proyecto surgieron varios problemas relevantes, que resolví de forma metódica (reproducir ? localizar la causa ? corregir ? verificar con un test):

- Pérdida aparente de memoria del asistente. El bot mezclaba proyectos y no recordaba turnos anteriores. Se atribuyó inicialmente a un fallo de persistencia, pero al investigar comprobé que PostgreSQL guardaba y cargaba correctamente. La causa real era que el historial de conversación se inyectaba dentro del mismo bloque que el contexto recuperado del RAG, y un endurecimiento del prompt («responde solo con información del contexto») hacía que el modelo tratara el historial como un dato no fiable y lo ignorara. Lo resolví separando el historial conversacional del contexto RAG en secciones distintas del prompt, con reglas propias. Fue un buen ejemplo de que el diagnóstico evidente no siempre es el correcto.
- Doble truncado de la ventana de contexto. El asistente olvidaba conversaciones de días anteriores. La causa era un recorte del historial aplicado dos veces (en el orquestador y de nuevo en el agente de generación), que dejaba fuera todo lo anterior al último turno. Lo corregí ampliando la ventana y eliminando el recorte redundante, con un test de regresión que reproduce el olvido.
- Timeout en la primera consulta (arranque en frío del modelo). La primera petición a un modelo grande en la GPU compartida tardaba mucho en cargar y provocaba errores de tiempo de espera. Lo resolví con un calentamiento del modelo al arrancar el servicio, un timeout específico para el arranque en frío y manteniendo el modelo en memoria.
- Mezcla de memoria entre usuarios. La memoria se compartía indebidamente entre conversaciones distintas. Lo corregí migrando la clave de la memoria a una clave compuesta por usuario y conversación.
- Seguridad frente a inyección de prompts. Detecté que las reglas escritas en el prompt no bastaban para evitar respuestas indebidas. Añadí sanitización de la salida en código (idioma, emojis, sufijos) además de endurecer el prompt, porque la lógica de seguridad debe ser determinista y auditable, no depender solo del modelo.
- Contexto perdido en referencias anafóricas y ordinales. El asistente perdía el hilo cuando el usuario decía «el tercero» o «ese proyecto». Lo resolví manteniendo el proyecto activo entre turnos y capturando correctamente el proyecto resuelto.

En todos los casos el procedimiento fue el mismo: reproducir el fallo, aislar la causa real (no la aparente), corregir con el mínimo cambio necesario, y dejar un test que evite la regresión.

4. Identificación de las aportaciones que, en materia de aprendizaje, han supuesto las prácticas

- De la teoría a las consecuencias reales de la arquitectura. Migrar de una arquitectura monolítica a una hexagonal me enseñó, con problemas concretos delante, por qué merece la pena la separación en capas: testabilidad, capacidad de sustituir infraestructura y facilidad para localizar dónde está cada responsabilidad.

- El ciclo completo de un producto software. He vivido todas las fases: diseño, implementación, pruebas, despliegue en producción, evaluación con usuarios reales y corrección de incidencias tras el despliegue. Ver mi código funcionando en un servidor de la UPV y siendo usado en un piloto real es distinto a entregar una práctica de clase.
- Depuración disciplinada. He aprendido a no quedarme con la primera explicación de un fallo. Varios de los bugs más costosos tenían una causa distinta de la aparente, y la diferencia estuvo en reproducir y verificar antes de tocar código.
- RAG y sistemas con LLM en la práctica. He interiorizado las limitaciones reales de estos sistemas (alucinaciones, arranque en frío, gestión de contexto, seguridad frente a inyección) y cómo mitigarlas con decisiones de ingeniería, no solo con prompts.
- Evaluación rigurosa. He aprendido a medir la calidad de un sistema de IA con métricas objetivas (RAGAS, con una calidad en torno a 0.91 y un 84,6% de acierto sobre el banco de preguntas) y con evaluación de usabilidad mediante usuarios reales, y a ser honesto con las cifras, corrigiendo métricas que no estaban respaldadas por los datos.

5. Evaluación de las prácticas y sugerencias de mejora

La valoración global es muy positiva. Han sido unas prácticas con un alcance amplio y responsabilidad real: no un ejercicio cerrado, sino un sistema completo que se ha desplegado y que la ONG puede usar. La combinación de un entorno de investigación (Cátedra ENIA-UPV) con un destinatario real (la ONG DNI) le dio sentido y utilidad al trabajo, y el acompañamiento del tutor fue cercano y útil para orientar las decisiones importantes.

Como aspecto a mejorar, señalaría:

- Infraestructura de cómputo. Buena parte de los problemas de rendimiento (arranque en frío, timeouts) venían de trabajar sobre una GPU compartida. Disponer de recursos dedicados, o de un entorno de staging separado del de producción, habría facilitado las pruebas.

En conjunto, las prácticas han cumplido con creces su objetivo formativo: he consolidado conocimientos del grado, he adquirido experiencia real de ingeniería de principio a fin y he entregado un sistema en producción con impacto para una entidad social.