DuocUC | ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES

# Asistente inteligente para una Universidad

**Fernanda Paredes**

**Giocrisrai Godoy**

**Ingeniería de soluciones con IA / 003D**

# Caso organizacional

La organización corresponde a una universidad ficticia, de tamaño medio, dedicada a la educación superior. Atiende a más de 10.000 estudiantes en distintas carreras, tanto diurnas como vespertinas. Su rubro principal es la formación académica profesional y técnica.

## Problemática o desafío

Actualmente, los estudiantes enfrentan demoras y dificultades para resolver consultas frecuentes relacionadas con notas, calendarios académicos y becas. La gestión manual mediante secretarías genera sobrecarga administrativa y baja satisfacción estudiantil. El desafío consiste en implementar un asistente inteligente capaz de responder estas consultas de manera rápida, confiable y transparente

### *Objetivos de la intervención*

- Reducir los tiempos de respuesta a consultas estudiantiles.
- Disminuir la carga administrativa de secretarías.
- Mejorar la satisfacción y confianza de los estudiantes.
- Garantizar transparencia en la entrega de información institucional.

## Motivación para el uso de IA y agentes inteligentes

En la primera etapa del proyecto se trabajó con técnicas básicas de prompting usando modelos LLM (GPT-4o), lo que permitió generar respuestas coherentes pero sin capacidad de mantener contexto entre turnos. Esta nueva versión avanza hacia un agente más completo, incorporando memoria conversacional, planificación y recuperación aumentada de información (RAG), con el objetivo de entregar respuestas más precisas, trazables y consistentes.

## Las técnicas empleadas incluyen:

- **Zero-Shot y Few-Shot Prompting:**
  Para generar respuestas claras sin necesidad de grandes ejemplos previos.
- **Razonamiento guiado por planificación (Planner Agent):**
  El agente identifica la intención del usuario y decide si debe recuperar información, razonar o redactar.

- **Retrieval-Augmented Generation (RAG):**
  Permite buscar fragmentos relevantes en documentos locales (becas, reglamentos) y combinarlos con la generación del modelo.

- **Memoria de corto plazo:**
  Uso de un sistema de resumen conversacional (SessionMemory) para mantener coherencia dentro de la sesión.

Estas técnicas permiten construir un agente más robusto, transparente y alineado a los requisitos institucionales, mejorando la experiencia del usuario y reduciendo la carga operativa de las áreas administrativas.

## Diseño e implementación del agente

El agente está diseñado bajo una arquitectura modular compuesta por tres niveles principales, lo que permite una ejecución ordenada, trazable y fácilmente mantenible:

1. Cerebro (Core Engine): Impulsado por el modelo GPT-4o, encargado del razonamiento, la generación de respuestas y la toma de decisiones. Este componente interpreta el prompt, integra el contexto del RAG y produce la respuesta final.
2. Memoria (Memory): Implementada mediante el módulo SessionMemory.
   - Mantiene el historial del diálogo durante la sesión.
   - Genera resúmenes para evitar acumulación excesiva de tokens.
   - Permite coherencia entre consultas consecutivas del usuario.
3. Herramientas y Subagentes (Tools & Subagents):

   Extienden las capacidades del agente principal mediante funciones especializadas:

   - **Planner Agent:** Define el flujo de acciones necesario para resolver la consulta (búsqueda, razonamiento, redacción).
   - **Subagente de búsqueda (RAG):** Recupera fragmentos relevantes desde documentos locales en la carpeta *data/*.
   - **Subagente de redacción:** Genera respuestas limpias, estructuradas y con tono académico.
   - **Observabilidad:** Registra logs, métricas y eventos mediante JSON estructurado.

## Orquestación y componentes

- Agente principal (controller): Recibe la consulta del usuario, ejecuta validación, activa el planner y envía la llamada final al modelo. También gestiona la memoria y registra los eventos mediante log_event()

- Subagente de búsqueda: Utiliza un método de chunking (800 caracteres por bloque, con 120 de solapamiento) y un sistema de puntuación por coincidencia de tokens para seleccionar los fragmentos más relevantes.

- Subagente de redacción: Ordena la información encontrada, genera una respuesta clara y cita la fuente correspondiente [archivo.txt].

- Memoria contextual: Mantiene coherencia entre preguntas. Ejemplo: si el usuario menciona "vespertina", la memoria registra este valor para los siguientes turnos.

## Diseño e implementación

Estructura del proyecto:

- **assistant_uni.py —** Controlador principal del agente y punto de entrada del programa.

- **.env —** Variables de entorno, API keys, base URL y configuración del modelo.

- **memory.py —** Módulo encargado de la memoria conversacional.

- **planner_agent.py —** Responsable de la planificación de tareas del agente.

- **observability.py —** Funciones de logging, métricas y trazabilidad (si usas archivo separado).

- **dashboard.py —** Dashboard de observabilidad implementado en Streamlit.

- **data/ —** Conjunto de archivos para el sistema RAG (ej.: becas_2026.txt, reglamento_academico.txt).

- **logs/agent.log —** Archivo de registro JSON estructurado donde se almacenan todos los eventos del sistema.

- **requiriments.txt** - Dependencias del proyecto.

### Dependencias:

openai, langchain, crewai, python-dotenv, python-json-logger, streamlit, pandas, numpy, matplotlib

El sistema fue probado en entorno virtual (.venv) y ejecutado desde Git Bash, confirmando respuestas adaptativas en tiempo real.

### Configuración de memoria y planificación

El agente utiliza un sistema de memoria diseñado para mantener coherencia durante la conversación:

- **Memoria de corto plazo:**
  Gestionada mediante el módulo SessionMemory, almacena turnos recientes de la conversación y genera un resumen contextual para mantener continuidad en las respuestas del agente.

No se implementa memoria persistente a largo plazo, ya que el enfoque del proyecto se centra en sesiones independientes con contexto limitado.

La planificación de tareas se realiza mediante el módulo planner_agent.py, que actúa como un **planificador secuencial**. Este componente analiza la intención del usuario, determina si debe activar la recuperación de información (RAG), generar una respuesta directa o combinar ambos procesos, y finalmente ejecuta las acciones necesarias de manera ordenada.

#### *Ejemplo de flujo*

1. El usuario ingresa una pregunta en assistant_uni.py.
2. El sistema ejecuta la sanitización **del input**, verificando longitud máxima y posibles instrucciones inseguras.
3. Se aplica **rate limiting**, permitiendo un número máximo de consultas por minuto.
4. El planner activa el módulo **RAG**, que selecciona los fragmentos más relevantes desde /data.
5. Se construye un prompt que contiene:
   - Instrucciones del SYSTEM_PROMPT
   - Fragmentos recuperados (Contexto RAG)
   - La pregunta del usuario.
6. Se envía la solicitud al modelo (client.chat.completions.create).
7. Se miden métricas como:
   - Latencia de respuesta (ms)
   - Tokens usados (si el proveedor lo admite)
8. Toda la interacción se registra en logs/agent.log con formato JSON.
9. Se actualiza la memoria y se entrega la respuesta al usuario.

## Documentación técnica

- Frameworks: OpenAI API (o Azure OpenAI), Streamlit, Python JSON Logger, pandas, matplotlib, dotenv.

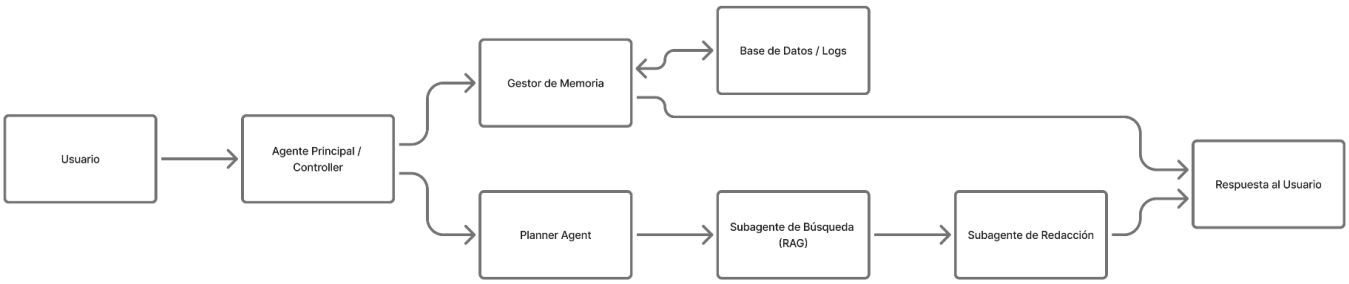
- Memoria: Implementada mediante un módulo propio (SessionMemory), con almacenamiento de contexto local y resumen conversacional.

- Planificación: Ejecución secuencial mediante un *Planner Agent* personalizado (planner_agent.py), que identifica subtareas (búsqueda, razonamiento y redacción).

- RAG (Recuperación aumentada): Sistema propio basado en búsqueda de texto local, chunking y puntaje por coincidencia de tokens.

- Observabilidad: Logging estructurado en formato JSON, métricas de latencia y eventos, y dashboard en Streamlit para visualización.

- Repositorio GitHub: Contiene código fuente, scripts, dashboard, logs, documentación y archivos de datos usados para RAG.



## Redacción técnica y resultados

Durante la validación, se realizaron consultas reales para evaluar la coherencia, planificación, trazabilidad y uso de memoria del agente.

Una de las preguntas utilizadas fue:

*"¿Cuándo inician las postulaciones a becas?"*

### *Respuesta generada:*

Las postulaciones se realizan entre marzo y abril, con fechas exactas que varían según la institución y el tipo de beneficio. Reúne tu formulario, concentración de notas y antecedentes socioeconómicos, y valida los plazos en la página oficial. (Utiliza esta información desde el archivo **[becas_2026.txt]**).

### *Evaluación:*

El agente respondió de manera coherente, mantuvo tono formal, citó la fuente local correspondiente y evitar repeticiones innecesarias gracias al uso de:

- Memoria contextual de sesión (SessionMemory)
- Planificación mediante Planner Agent
- RAG local para recuperar información desde archivos institucionales
- Temperatura baja (0.5) para mejorar estabilidad en las respuestas

Los resultados evidencian mejoras en consistencia, organización y claridad frente al prototipo inicial.

### Resultados de Observabilidad:

Además de evaluar la calidad de las respuestas, se midió el comportamiento interno del agente gracias al módulo de observabilidad, que incluye:

## 1. Logging estructurado (JSON)

Cada interacción genera eventos con:

- trace_id
- span_id
- role (system, user, assistant)
- latency_ms
- message
- tokens_used

### 2. Dashboard de métricas

Se implementó un dashboard en Streamlit para visualizar:

- Latencia promedio por interacción
- Distribución de latencia
- Eventos agrupados por rol
- Total de registros
- Tokens usados
- Exploración detallada por trace_id

El dashboard permite entender el desempeño interno, detectar errores y validar el flujo de orquestación del agente.

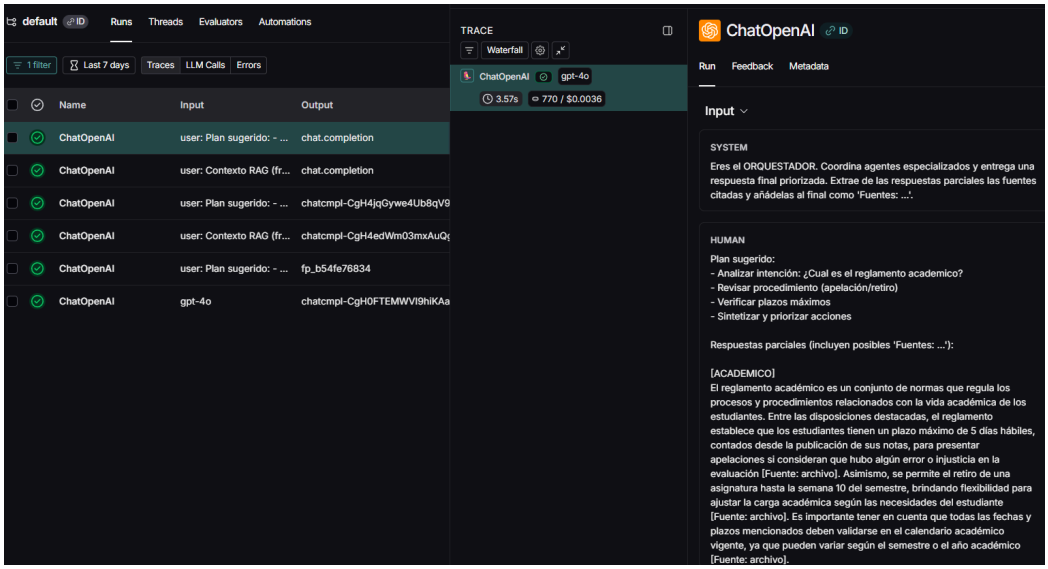
## Conclusión

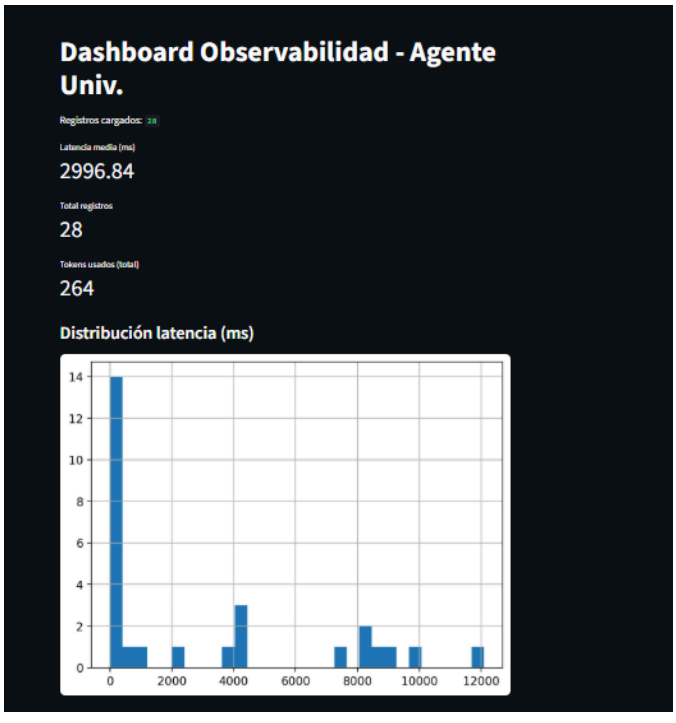
El Asistente Universitario Inteligente evolucionó desde un prototipo simple hacia un agente conversacional completo, incorporando memoria contextual, planificación y recuperación aumentada de información. Estas mejoras permitieron entregar respuestas más coherentes, precisas y consistentes frente a consultas estudiantiles reales.

La integración del módulo de observabilidad incluyendo logs estructurados y un dashboard en Streamlit aportó trazabilidad, métricas de rendimiento y mayor comprensión del funcionamiento interno del agente, elementos claves para un sistema escalable y mantenible.

En general, el proyecto cumplió sus objetivos, mostrando un agente capaz de apoyar procesos informativos y reducir carga administrativa. Como trabajo futuro, se considera integrar bases de datos persistentes, conectar APIs institucionales y ampliar las capacidades del dashboard para su eventual uso en entornos reales.

## Anexos




## Referencias


- OpenAI. (2025). *OpenAI API Documentation*. https://platform.openai.com/docs
- LangChain. (2025). *LangChain Framework Documentation*. https://python.langchain.com
- CrewAI. (2025). *Multi-Agent Coordination Library*. https://docs.crewai.io
- Saurabh Kumar. (s. f.). *python-dotenv Documentation*. https://saurabh-kumar.com/python-dotenv
- ChatGPT. (2025). *OpenAI Platform*. https://chat.openai.com