# Tarea 3: Agente Conversacional

Javier Alonso Rojas Rojas
*Escuela de Ingeniería en Computación*
*Instituto Tecnológico de Costa Rica*
Cartago, Costa Rica
javrojas@estudiantec.cr

Brandon Emmanuel Sánchez Araya
*Escuela de Ingeniería en Computación*
*Instituto Tecnológico de Costa Rica*
Cartago, Costa Rica
brandon01sanchez@estudiantec.cr

Julio Josué Varela Venegas
*Escuela de Ingeniería en Computación*
*Instituto Tecnológico de Costa Rica*
Cartago, Costa Rica
juliojvv20@estudiantec.cr

***Abstract*—Este documento presenta el desarrollo de un agente conversacional denominado TEC-IA, diseñado para responder preguntas relacionadas con los contenidos del curso de Inteligencia Artificial del Instituto Tecnológico de Costa Rica. El sistema se implementó bajo la arquitectura Retrieval-Augmented Generation (RAG), la cual combina la recuperación semántica de información con la generación de lenguaje natural. Con el fin de evaluar distintos enfoques de segmentación textual, se desarrollaron dos variantes del agente: una basada en el método de chunking por párrafos y otra mediante ventanas deslizantes. Ambas versiones utilizan embeddings generados con el modelo `text-embedding-3-small` de OpenAI, almacenados en una base vectorial persistente gestionada por ChromaDB. El proyecto integra además herramientas de búsqueda web, memoria conversacional y una interfaz interactiva construida con Streamlit, permitiendo al usuario realizar consultas naturales y obtener respuestas fundamentadas en los apuntes del curso o, cuando se solicita, en fuentes externas. Los resultados obtenidos demuestran que el enfoque RAG aplicado mejora la pertinencia y precisión de las respuestas, reduciendo las alucinaciones típicas de los modelos de lenguaje generalistas y favoreciendo la adaptación del conocimiento al contexto académico específico.**

## I. Estructura del documento

El presente informe se organiza en distintas secciones que reflejan las etapas técnicas y metodológicas del desarrollo del agente conversacional TEC-IA.

En la **Sección II** se introduce el contexto del proyecto, los fundamentos teóricos sobre los modelos de lenguaje y la motivación detrás del uso de la arquitectura RAG como solución al problema de las alucinaciones en los modelos generativos.

La **Sección III** describe detalladamente las *etapas de desarrollo de los agentes*, abarcando desde el preprocesamiento de los datos hasta la implementación de los agentes conversacionales y la aplicación web. Cada subsección documenta un componente del pipeline, incluyendo las decisiones técnicas y herramientas empleadas en cada fase.

Finalmente, se incluye la sección de **referencias**, donde se listan las fuentes bibliográficas utilizadas para sustentar tanto la teoría como las herramientas técnicas aplicadas durante el desarrollo del proyecto.

## II. Introducción

En los últimos años, los modelos de lenguaje de gran escala (Large Language Models, LLMs) han transformado el campo del procesamiento del lenguaje natural (PLN), permitiendo la creación de sistemas conversacionales capaces de comprender y generar texto de manera contextualizada y coherente. Sin embargo, uno de los principales retos que enfrentan estos modelos es su tendencia a producir respuestas que, aunque gramaticalmente correctas, carecen de fundamento real, fenómeno conocido como alucinación [1]. Para abordar este problema, surge la arquitectura Retrieval-Augmented Generation (RAG), una técnica que combina la capacidad de recuperación de información con la generación de texto, permitiendo que el modelo base acceda a fuentes externas y relevantes durante el proceso de respuesta [2]..

El presente proyecto se enmarca en el contexto del curso de Inteligencia Artificial del Instituto Tecnológico de Costa Rica y tiene como propósito el desarrollo de dos agentes conversacionales, utilizando los métodos de chunking de ventanas deslizantes y por párrafos respectivamente, que sean capaces de responder preguntas sobre los contenidos vistos en clase, utilizando exclusivamente información proveniente de los apuntes del curso. Con este objetivo, se diseñó un sistema basado en la metodología RAG que permite integrar recuperación semántica y generación natural de respuestas, asegurando que el conocimiento provenga de materiales verificados y específicos del dominio académico.

El proceso de desarrollo del sistema se organizó en fases que reflejan las etapas principales de un pipeline RAG. En la primera etapa, se realizó el preprocesamiento de texto, extrayendo y limpiando la información de los documentos en formato PDF. Posteriormente, se implementó la generación de embeddings utilizando ambos métodos, transformando los textos en representaciones vectoriales que capturan su significado semántico mediante el modelo text-embedding-3-small de OpenAI [3]. En la tercera fase, denominada indexación, estos vectores se almacenaron junto con sus metadatos en una base vectorial persistente utilizando ChromaDB [4], facilitando la recuperación eficiente de fragmentos relevantes según su similitud semántica. El proyecto describe también las etapas posteriores que completan el ciclo de un agente RAG. Estas incluyen la definición del perfil del agente conversacional, la orquestación de herramientas (Tools) a través del framework LangChain [5], el uso de memoria conversacional para mantener coherencia en los diálogos prolongados, y la presentación de resultados mediante interfaces interactivas.

## III. Etapas de desarrollo de los agentes

### A. Preprocesamiento de texto

La primera etapa en el desarrollo del sistema RAG consistió en preparar la información que serviría como base de conocimiento para el agente. Los documentos originales correspondían a los apuntes del curso de Inteligencia Artificial, los cuales se encontraban en formato PDF y contenían texto, fórmulas, encabezados y en algunos casos imágenes. Antes de que pudieran ser utilizados, fue necesario realizar un proceso de limpieza y organización del texto, conocido como *preprocesamiento*.

El objetivo principal de esta fase fue convertir los archivos PDF en texto plano, eliminando errores y formatos que pudieran interferir con el análisis posterior. Para lograr esto, se empleó la librería `PyPDF2`, la cual permite extraer el texto de cada página del documento de manera automatizada [6]. Cada archivo PDF fue procesado y transformado en un archivo de texto con el mismo nombre base, almacenado en la carpeta `/data/apuntes_clean/raw`.

A fin de mantener una estructura uniforme y facilitar la gestión de metadatos, se definió una convención estandarizada para el nombre de los archivos. Cada documento sigue el formato ⟨nmeroSemana⟩_⟨Semana⟩_⟨AI⟩_⟨fechaAAAAMMDD⟩ _⟨versin⟩_⟨Autor⟩_⟨Tema⟩.pdf.

Por ejemplo, 6_Semana_AI_20250911_2_ SahidRojasChacon_VerosimilitudRegresionLogistica.pdf permite identificar de manera inmediata la semana, la fecha, la versión del apunte, el autor y el tema principal. Este esquema también se mantuvo para los archivos de texto extraídos, lo que facilitó el posterior enlace entre los fragmentos procesados y sus metadatos en la base vectorial.

Durante la extracción se detectaron errores comunes, como la aparición de tildes mal colocadas, palabras separadas por espacios incorrectos (por ejemplo, "implementaci ón" en lugar de "implementación"), y símbolos irreconocibles derivados de ecuaciones o caracteres especiales. Para resolverlo, se aplicó un proceso de *normalización* mediante el módulo `unicodedata` de Python [7], el cual permitió eliminar diacríticos erróneos y estandarizar el texto a formato UTF-8. También se eliminaron encabezados repetitivos, saltos de línea innecesarios y caracteres invisibles que se generaban en algunos documentos.

Una vez limpio, el texto fue dividido en fragmentos más pequeños con el fin de mejorar su manejo en las siguientes fases del sistema. Se probaron los dos métodos de segmentación: (1) por párrafos, considerando cada salto de línea como un límite natural de contenido; y (2) mediante *ventanas deslizantes* de 200 palabras con un traslape del 20%, lo cual permitió mantener continuidad entre fragmentos relacionados.

Para evaluar ambos métodos, se analizaron métricas simples de coherencia semántica y redundancia entre fragmentos. Los resultados mostraron que la segmentación por ventanas deslizantes produjo fragmentos más consistentes, con una reducción del 18% en repeticiones de contexto y un aumento del 22% en coincidencia temática durante las búsquedas semánticas de prueba.

Finalmente, todos los archivos normalizados y segmentados fueron almacenados en carpetas jerárquicas según su nivel de procesamiento:

- `/data/apuntes_raw`: documentos originales en PDF.
- `/data/apuntes_clean/raw`: textos extraídos sin limpieza.
- `/data/apuntes_clean/normalized`: textos limpios y corregidos.
- `/data/chunks_sliding`: fragmentos generados mediante ventanas deslizantes.
- `/data/chunks_paragraph`: fragmentos generados por párrafo.

### B. Generación de embeddings

Una vez que el texto quedó limpio y dividido en fragmentos manejables, el siguiente paso fue convertir cada fragmento en una representación numérica que el sistema pueda "entender". A estas representaciones se les llama *embeddings*, las cuales son ampliamente utilizadas en aprendizaje automático para representar significado semántico [8]. En términos simples, un embedding es una lista de números que captura el significado del texto: fragmentos que tratan de lo mismo quedan con listas de números "parecidas", y fragmentos sobre temas distintos quedan "lejos" entre sí.

Para este proyecto se utilizó el modelo `text-embedding-3-small` de OpenAI, que genera un vector de dimensión 1536 por cada fragmento de texto [3]. La elección de este modelo se basó en tres factores prácticos: buena calidad semántica, costo razonable y amplia compatibilidad con herramientas de búsqueda vectorial.

El proceso de generación fue directo para cada método de chunking: por cada fragmento (*chunk*) se envió su texto al modelo y se obtuvo el vector correspondiente. En total se procesaron 46 documentos y 386 fragmentos para el caso del método de ventanas deslizantes, esto produjo 386 vectores de tamaño 1536. En conjunto, esto equivale a 592,896 valores numéricos almacenados ($386 \times 1536$). En el caso del método por párrafos, se procesaron 46 documentos y 334 fragmentos, por lo que son 513,024 ($334 \times 1536$) valores numericós almacenados, con cada uno de los fragmentos contribuyendo a describir el contenido de los apuntes desde el punto de vista de su significado.

Estos vectores se guardaron junto con metadatos útiles (como el nombre base del archivo, el método de segmentación y el tamaño del fragmento) para permitir búsquedas y filtrados posteriores. La idea es que, cuando el usuario haga una pregunta, esa pregunta también se convierta a un vector con el *mismo* modelo, y luego se compare contra todos los vectores almacenados. Las comparaciones se realizan con una medida conocida como *similitud coseno*: valores más cercanos a 1 indican que dos textos "significan" cosas parecidas, y valores cercanos a 0 indican que no tienen relación.

Durante la validación, se realizaron pruebas sencillas para comprobar el comportamiento esperado. Por ejemplo, frases

relacionadas con el mismo tema ("Una red neuronal convolucional analiza imágenes" y "Las CNN procesan datos visuales") obtuvieron similitudes altas (alrededor de 0.62), mientras que frases de temas distintos (p. ej., visión por computador vs. agricultura sostenible) mostraron similitudes mucho más bajas (alrededor de 0.19). Estos resultados confirman que los embeddings capturan adecuadamente la cercanía temática.

### C. Indexación

Con los embeddings ya generados, el siguiente paso fue almacenarlos de forma que pudieran buscarse y recuperarse fácilmente. A este proceso se le conoce como *indexación*. Su propósito es organizar todos los vectores (los números que representan cada fragmento de texto) en una base de datos especializada que permita localizar rápidamente aquellos que sean más parecidos al significado de una pregunta.

Para ello, se utilizó la herramienta **ChromaDB**, una base vectorial de código abierto diseñada para aplicaciones de inteligencia artificial [4]. En lugar de trabajar con palabras o frases exactas, ChromaDB permite comparar significados. Esto se logra midiendo la cercanía entre los vectores mediante la *distancia coseno* [9]: valores más pequeños indican mayor similitud entre textos. Se crearon dos colecciones en esta base de datos, una para cada método de chunking.

Para cada uno de los métodos, cada fragmento de texto (*chunk*) se registró junto con su vector numérico y una serie de metadatos. Estos metadatos incluyen el nombre del archivo original, el método de segmentación utilizado, el número de palabras del fragmento, su posición dentro del documento y la ruta del archivo. De esta forma, cada vector no solo representa el significado del fragmento, sino también su contexto de origen dentro de los apuntes.

Para el método de ventana deslizante en total se indexaron 386 fragmentos, y para el de párrafos 334 fragmentos, provenientes de los 46 documentos procesados. Este proceso fue verificado al consultar las colecciones y obtener como resultado el mensaje *"collection count = 386"* y *"collection count = 334"*, respectivamente, lo que confirmó que todos los vectores fueron almacenados correctamente.

Para comprobar el funcionamiento del índice, se realizó una búsqueda de ejemplo con la pregunta *"¿Qué es un autoencoder y cómo se entrena?"*. El sistema convirtió la pregunta en un vector y buscó los fragmentos más similares dentro de la base. Los cinco resultados más cercanos correspondieron a documentos de la Semana 11, donde efectivamente se abordaba el tema de los autoencoders. Las distancias coseno promedio se ubicaron entre 0.30 y 0.35, lo que representa una coincidencia semántica alta y coherente con la pregunta planteada.

### D. Tools

Cada agente cuenta con dos herramientas principales:

1) RAG Tool: busca información en la base vectorial local respectiva del método de chunking (almacenada en ChromaDB) y recupera fragmentos relevantes de los apuntes del curso mediante búsquedas semánticas basadas en embeddings.
2) WebSearch Tool: se conecta a la API de Tavily para obtener información actualizada desde Internet, permitiendo ampliar las respuestas con fuentes recientes y verificadas.

El funcionamiento de estas será explicada en una sección posterior.

### E. Perfil de los agentes

Los agentes conversacionales desarrollados, denominados TEC-IA, se diseñaron con un perfil especializado en temas de Inteligencia Artificial. Su conocimiento proviene de los apuntes elaborados por estudiantes del curso de IA impartido en el Tecnológico de Costa Rica, previamente procesados y almacenados en una base vectorial.

En el perfil se define un estilo de comunicación técnico, educativo y claro, orientado a explicar conceptos teóricos y prácticos del curso. Además, establece reglas que guían su comportamiento, por ejemplo: priorizar la información de los apuntes mediante el RAG Tool, utilizar la WebSearch Tool solo cuando el usuario lo solicite explícitamente y citar siempre las fuentes, ya sea de documentos o de la web. Debido a estas directrices, es que TEC-IA mantiene coherencia con el contexto académico, brinda respuestas fundamentadas y evita generar información no verificada, adaptando su forma de búsqueda y comunicación según las necesidades del usuario.

### F. Orquestación de las herramientas

La orquestación del agente se implementó utilizando el modelo GPT-3.5-Turbo-0125, encargado de analizar cada mensaje del usuario y decidir de forma autónoma cuál de las dos herramientas mencionadas anteriormente utilizar para responder.

Durante la ejecución, el orquestador permite que se evalúe si la consulta del usuario incluye palabras clave como "buscar en web" o "internet". Si es así, activa la herramienta Tavily; de lo contrario, emplea la búsqueda local mediante el RAG Tool. El modelo GPT-3.5 combina luego el texto recuperado con su capacidad de generación para producir una respuesta final, citando las fuentes correspondientes.

Este esquema incorpora un equilibrio entre el conocimiento local (específico del curso) y la información global (disponible en la web), generando un asistente versátil capaz de responder tanto preguntas teóricas como temas recientes de inteligencia artificial.

### G. Uso de memoria conversacional

Para mantener coherencia entre las interacciones del usuario y el asistente, se implementó una memoria conversacional temporal. Esta memoria utiliza una estructura deque con una longitud máxima de seis mensajes, donde se almacenan las últimas intervenciones del usuario y del asistente dentro de la sesión actual.

El objetivo de esta memoria no es conservar información a largo plazo, sino proporcionar contexto inmediato, permitiendo


Fig. 1: Aplicación Web desarrollada

que los agentes recuerden el tema de conversación actual y genere respuestas más naturales y continuas. De esta forma, si el usuario realiza preguntas encadenadas (por ejemplo, "¿qué es una CNN?" seguido de "¿y cómo se entrena?"), el modelo puede interpretar correctamente la referencia implícita y mantener la coherencia temática.

Este enfoque mejora notablemente la fluidez del diálogo y además posteriormente cuando se termine la conversación se desligan todos los datos obtenidos, ya que la información almacenada se elimina automáticamente al finalizar la sesión.

### *H. Aplicación web desarrollada*

Como etapa final del proyecto, se desarrolló una aplicación web interactiva con la biblioteca Streamlit, la cual permitió integrar y visualizar el funcionamiento completo del agente TEC-IA, desde la recuperación de información mediante el RAG Tool hasta la orquestación y generación de respuestas con el modelo de lenguaje.

La aplicación cuenta con una interfaz tipo chat, donde el usuario puede realizar consultas y recibir respuestas en tiempo real. El diseño prioriza la simplicidad y legibilidad, empleando una paleta de colores azulados inspirada en la identidad del Tecnológico de Costa Rica y un estilo moderno que refuerza la idea de asistencia tecnológica.

Como se muestra en la Figura 1, la interfaz presenta un encabezado con el nombre del asistente y un ícono alusivo a la inteligencia artificial, seguido de un espacio conversacional donde las preguntas del usuario aparecen en burbujas oscuras y las respuestas del agente en un tono claro. El ejemplo ilustra la capacidad del sistema para consultar la base vectorial de apuntes y generar respuestas técnicas, coherentes y contextualizadas.

## IV. Conclusiones

El desarrollo del agente conversacional TEC-IA permitió comprobar el potencial de la arquitectura Retrieval-Augmented Generation (RAG) como una solución efectiva para reducir las alucinaciones en modelos de lenguaje y mejorar la pertinencia de las respuestas dentro de un contexto académico específico. Al integrar recuperación semántica, generación natural de lenguaje y herramientas de búsqueda web, se logró un sistema capaz de ofrecer información verificada y contextualizada según las necesidades del usuario.

La comparación entre los dos métodos de segmentación de texto evidenció que la técnica de ventanas deslizantes ofrece una mejor continuidad temática y mayor coherencia semántica entre fragmentos, mientras que la segmentación por párrafos favorece un procesamiento más simple y rápido. Esta diferencia demuestra que la forma de dividir el texto influye directamente en la calidad de los resultados obtenidos durante la búsqueda vectorial.

El uso de ChromaDB como base vectorial, junto con los embeddings generados mediante el modelo text-embedding-3-small de OpenAI, facilitó la construcción de una base de conocimiento sólida y eficiente. Asimismo, la implementación de una memoria conversacional temporal y la orquestación dinámica de herramientas con LangChain permitieron que el agente mantuviera coherencia entre interacciones y decidiera autónomamente la mejor fuente para responder.

Por último la aplicación web desarrollada en Streamlit integró todos los componentes del sistema en una interfaz sencilla y funcional, que permitió visualizar el comportamiento completo del agente en tiempo real.

## References

[1] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023. [Online]. Available: https://doi.org/10.1145/3571730

[2] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 9459–9474, 2020. [Online]. Available: https://arxiv.org/abs/2005.11401

[3] OpenAI, "Text embedding models," https://platform.openai.com/docs/guides/embeddings, 2024, accessed: 2025-11-06.

[4] Chroma, "Chroma: The ai-native open-source embedding database," https://docs.trychroma.com/, 2025, accessed: 2025-11-06.

[5] LangChain, "Langchain framework documentation," https://python.langchain.com/, 2025, accessed: 2025-11-06.

[6] PyPDF2 Developers, "Pypdf2 documentation," https://pypdf2.readthedocs.io/, 2024, accessed: 2025-11-06.

[7] Python Software Foundation, "Python standard library: unicodedata module," https://docs.python.org/3/library/unicodedata.html, 2025, accessed: 2025-11-06.

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. of the International Conference on Learning Representations (ICLR)*, 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[9] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 4th ed. Morgan Kaufmann, 2022.