# Desarrollo de un tutor inteligente basado en diálogos mediante el uso de LLMs
# Development of a dialogue-based intelligent tutor using LLMs


Trabajo de Fin de Grado
Curso 2024–2025

Autor
Leonardo Macías Sánchez

Directores
Gonzalo Méndez Pozo
Pilar Sancho Thomas

Calificación: 10 (SB)

Doble Grado en Ingeniería Informática y Matemáticas
Facultad de Informática
Universidad Complutense de Madrid

# Desarrollo de un tutor inteligente basado en diálogos mediante el uso de LLMs
# Development of a dialogue-based intelligent tutor using LLMs

**Trabajo de Fin de Grado en Ingeniería Informática**


**Autor**
**Leonardo Macías Sánchez**

**Directores**
**Gonzalo Méndez Pozo**
**Pilar Sancho Thomas**



**Convocatoria:** Junio 2025

**Calificación: 10 (SB)**



**Doble Grado en Ingeniería Informática y Matemáticas**
**Facultad de Informática**
**Universidad Complutense de Madrid**



**16 de junio de 2025**

# Dedicatoria

*A mis padres, José Luis y Elisa*

# Agradecimientos

Quiero comenzar agradeciendo a mis compañeros de clase y amigos todos los momentos que hemos compartido durante la carrera. Juntos hemos formado un equipo único, y sin ellos, esta etapa habría sido como lanzarse a una piscina sin agua. Gracias por estar ahí en cada momento.

También quiero expresar mi agradecimiento a mis tutores, Gonzalo y Pilar, por su constante disponibilidad, sus valiosos consejos y por tenerme presente incluso fuera de las reuniones, enviándome información útil y relevante. Vuestra implicación ha sido fundamental en este proceso.

Mi gratitud también para Don Santos Пnto, por introducirme en el mundo de las olimpiadas de informática en Extremadura, y a Doña Isabel Durán, por brindarme la oportunidad de aprender en sus clases de robótica. En general, agradecimientos a toda la comunidad educativa del Colegio San José de Villafranca de los Barros, donde siempre me he sentido apoyado para crecer.

Y, por último, gracias de corazón a mis padres, por su apoyo incondicional, no solo durante estos años de carrera, sino a lo largo de toda mi vida. Este logro también es vuestro.

# Resumen


## Desarrollo de un tutor inteligente basado en diálogos mediante el uso de LLMs

Los avances tecnológicos, y en particular la inteligencia artificial, están transformando la educación. La realidad virtual puede reproducir escenarios de alto riesgo, como las emergencias radiológicas, sin comprometer la salud de los estudiantes, mientras los modelos de lenguaje de gran tamaño (LLMs) abren nuevas posibilidades a los Sistemas de Tutoría Inteligente.

Este trabajo desarrolla un tutor conversacional basado en LLMs integrado en ADARVE, un entorno de realidad virtual para entrenar a las Fuerzas y Cuerpos de Seguridad del Estado ante emergencias radiológicas. El sistema responde consultas en lenguaje natural, interviene de forma proactiva cuando detecta desorientación y genera un informe individualizado sobre el rendimiento del estudiante.

El desarrollo siguió un proceso iterativo en tres fases: integración tecnológica del tutor con ADARVE, creación de un sistema de diálogo funcional, e incorporación de estrategias pedagógicas para una ayuda adaptativa y generación de informes.

El tutor multiagente resultante ofrece respuestas contextualizadas con una latencia media de tres segundos, ajusta su apoyo al progreso del usuario y atiende múltiples sesiones concurrentes, todo ello fácilmente portable al estar contenerizado. Este trabajo confirma la viabilidad de combinar LLMs y realidad virtual para desarrollar un sistema de tutoría orientado a la formación inmersiva en contextos críticos, ofreciendo una herramienta escalable y adaptativa.



## Palabras clave

tutor inteligente, LLM, emergencias radiológicas, realidad virtual, diálogos, intervenciones proactivas, informe de rendimiento

# Abstract


## Development of a dialogue-based intelligent tutor using LLMs

Technological advances, particularly in Artificial Intelligence, are transforming education. Virtual Reality (VR) can simulate high-risk scenarios, such as radiological emergencies, without endangering students' health, while large language models (LLMs) open up new possibilities for Intelligent Tutoring Systems.

This work presents a dialogue-based intelligent tutor using LLMs and integrated into ADARVE, a VR environment designed to train Spanish State Security Forces in responding to radiological emergencies. The system answers natural language queries, intervenes proactively when it detects user disorientation, and generates personalized performance reports.

Development followed a three-phase iterative process: technological integration of the tutor into ADARVE, creating a functional dialogue system, and incorporating pedagogical strategies to enable adaptive support and report generation.

The resulting multi-agent tutor provides context-aware responses with an average latency of three seconds, adapts its assistance to the user's progress, and supports multiple concurrent sessions, all within a containerized system for easy deployment. This work demonstrates the feasibility of combining LLMs and VR to create a tutoring system tailored for immersive training in critical scenarios, offering a scalable and adaptive tool.



## Keywords

intelligent tutor, LLM, radiological emergencies, virtual reality, dialogues, proactive interventions, performance report

# Índice

# Índice de figuras

# Índice de tablas

Capítulo 1

# Introducción

*"Si enseñamos a los estudiantes de hoy como enseñábamos ayer, les estamos robando el mañana"*
— John Dewey

## 1.1. Motivación

Desde nuestros primeros pasos hasta la madurez, los humanos vivimos inmersos en un proceso continuo de adquisición de conocimientos y habilidades que nos permitan afrontar un mundo en constante cambio. Este entorno dinámico, en consonancia con los avances científicos y tecnológicos, nos obliga a replantear continuamente la manera en que aprendemos e impulsa el surgimiento de nuevas técnicas de aprendizaje diseñadas para hacer frente a los desafíos actuales.

Entre estas tecnologías innovadoras se encuentran los entornos de realidad virtual, que ofrecen la posibilidad de entrenar a los estudiantes en situaciones que serían difíciles de recrear en un aula tradicional. Es más, en ocasiones, la simulación de un escenario en el mundo real podría implicar poner en riesgo vidas humanas, convirtiendo a estos entornos virtuales en la única alternativa viable para adquirir conocimientos prácticos.

Este es el caso de las emergencias radiológicas, donde la preparación efectiva es crucial y cualquier error puede tener graves consecuencias. Consciente de esta realidad, el Consejo de Seguridad Nacional (CSN) colabora con la Universidad Complutense de Madrid en el proyecto ADARVE (Análisis de Datos de Realidad Virtual para formación en Emergencias Radiológicas, SUBV-20/2021), destinado a entrenar a las Fuerzas y Cuerpos de Seguridad del Estado en un entorno 3D con el fin de que puedan responder eficazmente ante este tipo de situaciones (CSN, 2021).

En el ámbito de la enseñanza digital, los Sistemas de Tutoría Inteligentes (ITS: *Intelligent Tutoring Systems*) destacan por ser una de las herramientas clásicas de aprendizaje adaptativo más populares (Sleeman y J. Brown, 1982). Estos sistemas usan técnicas de Inteligencia Artificial (IA) para adecuar dinámicamente el proceso educativo a las necesidades y capacidades del alumno (Alkhatlan y Kalita, 2018).

En los últimos años, el campo de la IA ha vivido una gran revolución gracias a las mejoras en procesamiento y generación de lenguaje natural. El principal impulsor de

este cambio ha sido el desarrollo de unos modelos grandes de lenguaje (LLMs: *Large Language Models*) muy sofisticados. Estos son modelos de aprendizaje profundo que constan de muchos parámetros y se entrenan con grandes cantidades de texto sin supervisar (Wikipedia, 2025a). Gracias a su capacidad para procesar datos y generar respuestas coherentes, los LLMs han cambiado por completo la forma en que las máquinas interactúan con el lenguaje humano.

Los resultados que ofrecen estos modelos de lenguaje abren una nueva vía de investigación para mejorar las sesiones de aprendizaje con tutores inteligentes. Este proyecto busca innovar en el sector del aprendizaje digital, mejorando el funcionamiento de los ITS mediante la incorporación de LLMs y facilitando su integración en el entorno de realidad virtual de ADARVE.

## 1.2. Objetivos

El propósito de este trabajo es el desarrollo de un tutor inteligente basado en modelos de lenguaje de gran tamaño (LLMs), que de soporte al entorno 3D del proyecto ADARVE con el fin de instruir a las Fuerzas y Cuerpos de Seguridad del Estado (FFCCSE) en los procedimientos a seguir ante emergencias radiológicas.

El cumplimiento de este objetivo se traduce en los siguientes requisitos para el tutor:

1. **Consultas**. Debe ser capaz de recibir preguntas de los estudiantes relacionadas con su formación (i.e, el ámbito nuclear y el entorno de realidad virtual) y proporcionar una respuesta coherente y adecuada a la situación de aprendizaje del alumno. Además, las preguntas que responda han de estar exclusivamente relacionadas con su formación y deben ser pertinentes a la misma, es decir, no debe admitir cualquier tipo de consulta.

2. **Proactividad**. Integrar un sistema de ayuda proactivo que se ajuste de manera dinámica al nivel de conocimiento del estudiante. Este sistema ha de ser capaz de detectar situaciones de desorientación e inactividad.

3. **Disponibilidad**. Debe poder ser consultado en cualquier momento dentro del entorno de realidad virtual.

4. **Informe**. Elaborar un informe de evaluación con información relevante sobre el progreso del estudiante.

5. **Independencia**. Su funcionamiento debe ser independiente del resto del proyecto ADARVE y debe poder integrarse en cualquier nivel sin requerir modificaciones significativas.

6. **Portabilidad**. Debe poder ejecutarse en distintos entornos de ejecución y admitir configuraciones sin requerir ajustes complejos.

## 1.3. Plan de Trabajo

Este apartado presenta la estrategia metodológica que orientará el desarrollo del proyecto hasta la consecución de los objetivos planteados. El diseño del tutor inteligente se plantea como un proceso iterativo, estructurado en torno a tres prototipos sucesivos:

- **Prototipo 1 (P1).** Centrado en demostrar la viabilidad técnica de integrar un tutor inteligente dentro del entorno 3D de ADARVE.
- **Prototipo 2 (P2).** Se busca lograr la integración de todas las tecnologías necesarias para el proyecto y emplearlas para habilitar un sistema funcional de diálogo con el estudiante.
- **Prototipo 3 (P3).** Corresponde a la versión final del tutor, que incorpora todas las funcionalidades propuestas.

A continuación, se describen en detalle las etapas previstas para el desarrollo del proyecto y el prototipo al que están asociadas. No obstante, las funcionalidades ya implementadas podrán ser objeto de revisiones y mejoras en versiones posteriores:

1. **Prueba de conectividad (P1).** Se desarrollará una versión inicial que asegure la conectividad entre el entorno ADARVE y la lógica del tutor. Dado que utilizan tecnologías distintas, verificar la transmisión de datos bidireccional es fundamental para el funcionamiento del sistema.
2. **Adaptación de ADARVE (P1).** Una vez garantizada la transmisión de datos, se validará que el usuario pueda enviar mensajes de texto desde ADARVE y que la respuesta generada por la lógica del tutor pueda visualizarse adecuadamente. Esto es relevante debido a que ADARVE es un proyecto desarrollado por un equipo independiente a este trabajo y es necesario confirmar que ofrece las herramientas adecuadas para soportar esta integración.
3. **Extracción de datos de ADARVE (P2).** El tutor debe ser capaz de extraer información del entorno virtual, así como del usuario y su progreso, para poder ofrecer respuestas contextualizadas y seguimiento personalizado.
4. **Selección del modelo de LLM (P2).** Se realizará un estudio comparativo de los modelos de lenguaje disponibles para determinar cuál se adapta mejor a los requerimientos del proyecto.
5. **Desarrollo del sistema de diálogo (P2).** Con los datos extraídos, el modelo de lenguaje seleccionado y la interfaz de interacción integrada, se construirá un sistema de interacción que permita responder a las consultas del estudiante de forma efectiva.
6. **Investigación pedagógica (P3).** Se analizarán teorías pedagógicas y enfoques de tutoría inteligente con el objetivo de adaptar el sistema de ayuda a las necesidades formativas de los usuarios y mejorar la experiencia educativa.

7. **Integración del sistema de ayuda adaptativo (P3).** Se implementará el sistema de ayuda autónomo basado en el conocimiento pedagógico adquirido durante la investigación, ajustando sus respuestas al nivel y evolución del usuario.

8. **Generación de informes de evaluación (P3).** Finalmente, el seguimiento del progreso del estudiante en el entorno y los hitos educativos conseguidos se reflejarán en la elaboración de un informe por cada sesión de aprendizaje.

Además, conforme se han ido sucediendo los avances en el desarrollo de cada prototipo, se ha ido elaborando de manera paralela esta memoria, documentando los logros, decisiones técnicas y ajustes realizados en cada fase del proyecto.

## 1.4. Estructura del Documento

Este documento se compone de seis capítulos, organizados para guiar al lector desde los fundamentos teóricos hasta los aspectos técnicos del desarrollo del proyecto y las conclusiones.

- **Capítulo 1: Introducción.** El presente capítulo, en el que se exponen los objetivos del trabajo, la metodología seguida para alcanzarlos y la motivación que dio origen al proyecto.

- **Capítulo 2: Estado de la cuestión.** Se presenta una revisión de los avances más recientes en el campo de la inteligencia artificial aplicada al procesamiento del lenguaje natural. A continuación, se introducen algunas de las principales teorías pedagógicas y su implementación en sistemas de tutoría inteligente desde la informática. Finalmente, se ofrece una breve descripción del sistema ADARVE.

Tras estos dos capítulos introductorios, el documento se centra en describir de forma detallada las distintas fases de desarrollo del proyecto. Para facilitar su comprensión, el proceso se ha dividido en tres capítulos diferenciados, cada uno centrado en uno de los prototipos del sistema.

Cada etapa de desarrollo se presenta en orden cronológico, destacando los principales retos encontrados, las decisiones técnicas que se tomaron y los avances logrados en el sistema.

- **Capítulo 3: Validación de la conectividad entre ADARVE y el tutor.** Se describe el diseño y desarrollo de un primer modelo funcional capaz de establecer comunicación entre ADARVE y la lógica básica del tutor.

- **Capítulo 4: Desarrollo del motor de diálogos.** En esta fase se implementa una primera versión operativa del sistema, que permite al estudiante realizar consultas desde ADARVE y recibir respuestas coherentes por parte del tutor.

- **Capítulo 5: Sistema de tutoría multiagente.** Se evoluciona hacia una arquitectura más avanzada basada en agentes. El tutor mejora la capacidad de responder preguntas y adquiere las de intervenir de forma autónoma y generar informes adaptados al progreso del estudiante. Además, se mejora la portabilidad del tutor mediante el uso de contenedores.
- **Capítulo 6: Conclusiones y trabajo futuro.** Finalmente, se expone el estado final del tutor y las principales conclusiones extraídas a lo largo del proyecto en base a los objetivos, así como una reflexión sobre posibles líneas de trabajo y mejoras futuras.

Para finalizar, se añade un apartado de bibliografía que reúne todas las fuentes empleadas a lo largo del trabajo y un apéndice con un informe real generado por el tutor en la simulación de una sesión de aprendizaje.

Capítulo 2

# Estado de la Cuestión

## 2.1. Introducción

Este capítulo presenta un recorrido por los principales avances teóricos y tecnológicos que enmarcan el desarrollo del presente trabajo. En primer lugar, se presentan algunos conceptos básicos sobre inteligencia artificial que servirán para analizar los modelos de lenguaje de gran tamaño. Se mencionan sus fundamentos técnicos y se examinan algunas de las implementaciones más relevantes disponibles en la actualidad.

Posteriormente, se abordan las teorías pedagógicas en las que se respaldan las de tecnologías de aprendizaje adaptativo, ofreciendo un marco conceptual que permite entender de qué manera los agentes educativos pueden contribuir al desarrollo de competencias de forma personalizada.

El capítulo continúa con una revisión del campo de los Sistemas de Tutoría Inteligentes, desde sus enfoques tradicionales hasta las nuevas oportunidades que ofrecen los LLM en el diseño de agentes más flexibles.

Por último, se describe el proyecto ADARVE, que proporciona el contexto práctico en el que se enmarca este Trabajo de Fin de Grado y sobre el que se construye la propuesta planteada.

## 2.2. Conceptos Básicos sobre Inteligencia Artificial

Esta sección ofrece una introducción concisa a algunos conceptos fundamentales de inteligencia artificial, con el objetivo de proporcionar el contexto necesario para comprender el contenido que se desarrolla en el resto del capítulo. Se abordan nociones como el aprendizaje automático, las redes neuronales y el procesamiento del lenguaje natural.

### 2.2.1. Aprendizaje Automático

El aprendizaje automático (ML: *Machine Learning*) es una rama de la inteligencia artificial que se centra en el desarrollo de sistemas capaces de construir modelos

a partir de datos de entrada y utilizarlos para resolver problemas (Russell y Norvig, 2022).

Esta misma fuente diferencia tres categorías principales de aprendizaje automático:

- **Aprendizaje supervisado**. El modelo aprende a partir de datos etiquetados que siguen un formato entrada-salida. Por ejemplo, la entrada puede ser una imagen de un gato y la salida su raza correspondiente, como siamés o persa. A este tipo de datos se les denomina *etiquetados* porque incluyen la respuesta esperada.

- **Aprendizaje no supervisado**. El modelo identifica patrones ocultos en datos de entrada no etiquetados. Por ejemplo, al analizar información sobre viviendas en una región, puede agruparlas en función de características similares como el tamaño, la ubicación o el precio.

- **Aprendizaje por refuerzo**. El modelo aprende a tomar decisiones optimizando una función de recompensa mediante la interacción con un entorno. Por ejemplo, si debe aprender a saltar un obstáculo, recibirá una recompensa positiva cuando lo logre y negativa cuando falle, ajustando así su comportamiento a lo largo del tiempo.

### 2.2.2. Redes Neuronales

Las redes neuronales son modelos computacionales inspirados en el funcionamiento de las neuronas biológicas (Hardesty, 2017). Son sistemas que reciben una entrada y generan una salida mediante una estructura interna compuesta por nodos (conocidos como *neuronas*) organizados en capas.

Una red neuronal típica consta de una capa de entrada, una o más capas ocultas y una capa de salida, como se ilustra en la Figura 2.1. Estas capas procesan la información a través de conexiones ponderadas, cuyo valor se ajusta durante el entrenamiento. Los valores de las conexiones de la red son conocidos como *parámetros* (Goodfellow, Bengio y Courville, 2016).

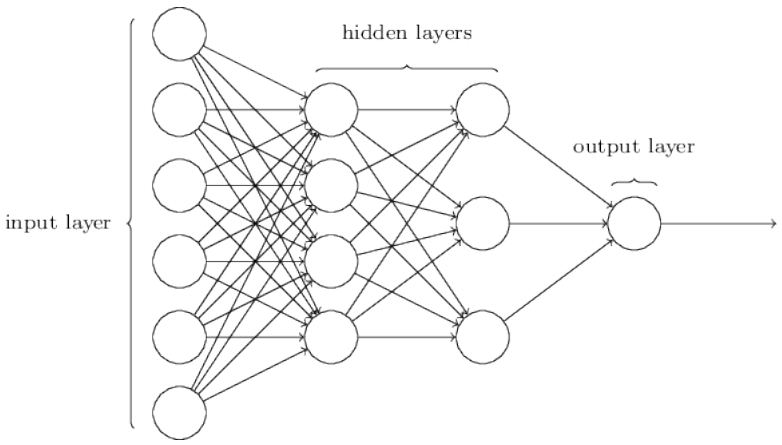


Figura 2.1: Estructura por capas de una red neuronal (Nielsen, 2015)

La mayoría de redes neuronales se entrenan mediante aprendizaje supervisado, utilizando conjuntos de datos etiquetados para ajustar los parámetros durante el proceso de entrenamiento. En general, cuanto mayor es el número de parámetros, mayor es la capacidad del modelo para representar funciones complejas (Cruciani, 2022).

Existen múltiples variantes de redes neuronales, como el perceptrón multicapa (MLP: *MultiLayer Perceptron*) o las redes neuronales recurrentes (RNN: *Recurrent Neural Networks*). Estas últimas están especialmente diseñadas para el procesamiento de secuencias como el texto. A diferencia de las redes MLP, las RNN no solo propagan la información hacia las capas siguientes, sino que también conservan su salida, denominada *estado oculto*. Esto les permite mantener una memoria del estado anterior al procesar cada nuevo elemento de la secuencia (Goodfellow, Bengio y Courville, 2016).

### 2.2.3. Procesamiento de Lenguaje Natural

El Procesamiento de Lenguaje Natural (PLN) es un campo dentro de la inteligencia artificial que se encarga de estudiar las interacciones con ordenadores mediante lenguaje humano.

Hace unos años, las RNN eran el estándar en muchas tareas de PLN, especialmente aquellas de tipo LSTM (*Long Short-Term Memory*), que conseguían paliar los efectos de la pérdida de memoria a largo plazo debida a un problema conocido como el desvanecimiento del gradiente (Hochreiter y Schmidhuber, 1997).

Las RNN tenían un papel especialmente importante en sistemas de traducción entre idiomas. Este tipo de sistemas suelen diseñarse con una arquitectura codificador-decodificador (*encoder-decoder*). El codificador transforma el texto de entrada en una representación numérica, conocido como último estado oculto, que el decodificador convierte en una salida (Tunstall, Werra y Wolf, 2022).

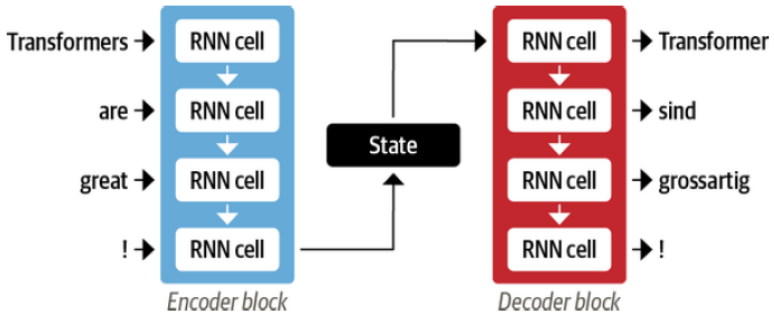


Figura 2.2: Arquitectura *enconder-decoder* con un par de RNN (Tunstall, Werra y Wolf, 2022)

Como se ilustra en la Figura 2.2, el uso del último estado oculto del codificador como único resumen de toda la secuencia generaba un cuello de botella que limitaba el rendimiento. Estas limitaciones motivaron la búsqueda de soluciones más eficaces, dando lugar al desarrollo de los mecanismos de atención y, con ello, al surgimiento

de una arquitectura revolucionaria, los *Transformers* (Vaswani et al., 2023).

## 2.3. Modelos Grandes de Lenguaje (LLM)

Los modelos grandes de lenguaje, conocidos por sus siglas en inglés como LLM (*Large Language Models*), representan uno de los avances más significativos en el PLN. Estos modelos, que se basan en la arquitectura *Transformer*, se caracterizan por haber sido entrenados con grandes volúmenes de texto y por estar compuestos por miles de millones de parámetros, lo que les permite generar y comprender lenguaje humano (Naveed et al., 2024).

### 2.3.1. Transformers

La arquitectura *Transformer* fue presentada por Google en el artículo "Attention is all you need" de Vaswani et al., 2023, basado en un mecanismo de autoatención y diseñado para aprovechar al máximo la paralelización computacional.

El diseño propuesto en el artículo original se compone de dos partes principales, un *encoder* y un *decoder*. En la Figura 2.3 se muestra el funcionamiento general de un *Transformer*, donde es posible distinguir en color gris el codificador a la izquierda y el decodificador a la derecha.

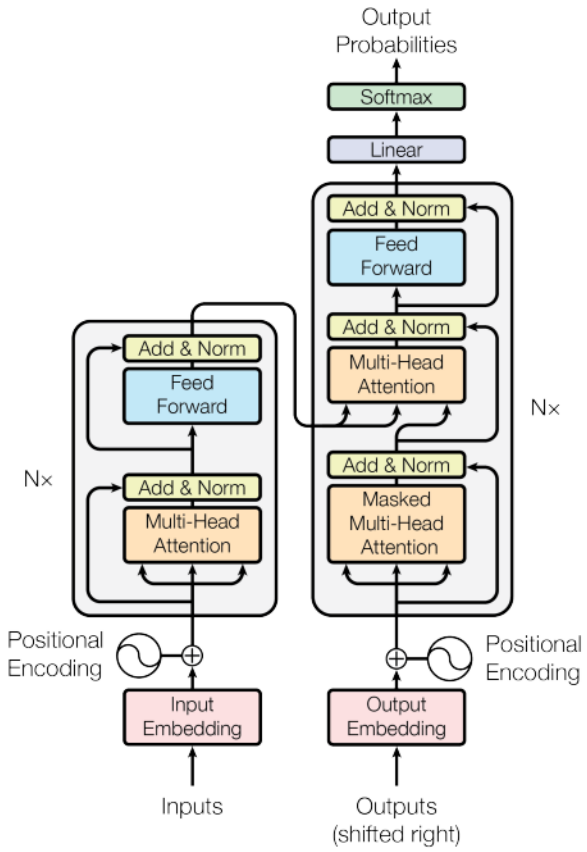


Figura 2.3: Arquitectura Transformer (Vaswani et al., 2023)

El procesamiento se inicia con la tokenización del texto de entrada, es decir, su segmentación en unidades básicas llamadas *tokens*. Estos *tokens* pueden ser palabras completas, fragmentos de palabra, caracteres especiales o signos de puntuación, dependiendo del esquema de codificación utilizado (Grefenstette, 1999).

Cada *token* se transforma en un vector que representa su significado semántico en un espacio numérico continuo. Esto se consigue mediante una técnica conocida como *word embeddings* (Almeida y Xexéo, 2023). Además, dado que el modelo procesa todos los *tokens* simultáneamente, no de forma secuencial, es necesario incorporar información sobre el orden de las palabras. Para ello, se añaden vectores de posicionamiento (*positional encodings*) a los *embeddings*, permitiendo que el modelo tenga en cuenta la posición relativa de cada *token*.

La combinación de estos vectores se introduce en una serie de capas del codificador, cada una compuesta por mecanismos de autoatención y redes neuronales de tipo perceptrón multicapa. El mecanismo de autoatención calcula la relevancia contextual de cada *token* en relación con los demás de la secuencia, permitiendo que el modelo capture relaciones semánticas complejas, incluso entre palabras distantes.

La salida del codificador se transmite al decodificador, que sigue una arquitectura similar y produce una respuesta.

### 2.3.2. Tipos de Modelos según su Arquitectura

Con el tiempo, han surgido diversas variantes de los *Transformers*, prescindiendo de alguno de sus componentes o modificando su forma de interacción. Estas adaptaciones han dado lugar a diferentes tipos de modelos de lenguaje.

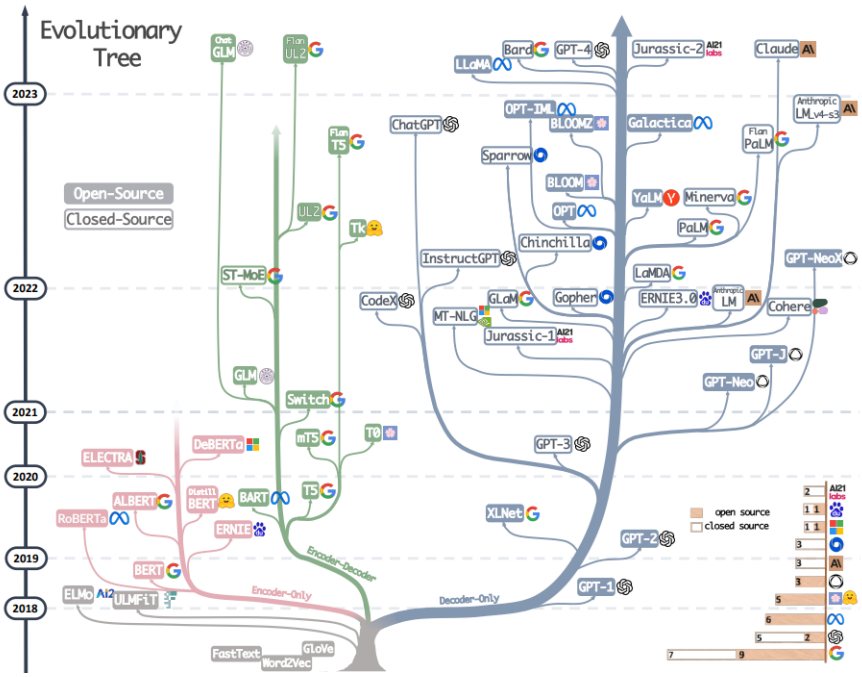


Figura 2.4: Clasificación de algunos modelos basados en *Transformers* (Yang et al., 2023)

En la Figura 2.4 se muestran las tres grandes líneas de evolución. En rosa se representan los modelos que utilizan únicamente el codificador, en azul los que emplean solo el decodificador y en verde los que combinan ambos. A continuación, se analiza con más detalle cada una de estas aproximaciones.

#### 2.3.2.1. Modelos solo codificador

Los modelos de solo codificador, llamados *encoder-only*, se han enfocado principalmente en tareas de comprensión del lenguaje, tales como clasificación de texto, reconocimiento de entidades nombradas o inferencia textual.

El primer modelo en adoptar esta arquitectura fue BERT (Devlin et al., 2019), que introduce el concepto de entrenamiento bidireccional, de ahí su nombre, *Bidirectional Encoder Representations from Transformers*. Esto significa que el modelo puede interpretar una palabra considerando su entorno completo, tanto su contexto anterior como posterior, lo que le permite comprender el significado en ambas direcciones.

BERT se pre-entrena con dos objetivos principales. El primero es el *masked language modeling* (MLM), que consiste en predecir palabras enmascaradas dentro de un texto. Para ello, se reemplazan aleatoriamente algunas palabras por un *token* especial y el modelo debe inferir la palabra original basándose en el contexto que la rodea. El segundo objetivo busca determinar si un fragmento de texto es probable que siga a otro, lo que se denomina *next sentence prediction* (NSP). Para los entrenamientos se emplean dos grandes fuentes de texto en inglés sin etiquetar: BookCorpus (Bandy y Vincent, 2021), que cuenta con más de 7000 libros de dominio público, y Wikipedia[1].

Modelos posteriores a BERT introdujeron mejoras sustanciales en eficiencia, arquitectura y uso de datos. Por ejemplo, ALBERT (Lan et al., 2020) propone desacoplar las dimensiones de los *embeddings* y compartir parámetros entre capas, lo que reduce considerablemente la cantidad de parámetros sin sacrificar rendimiento. Además, reemplaza la tarea NSP por una de ordenación de oraciones, más alineada con tareas de comprensión del lenguaje.

En la misma línea, ELECTRA (Clark et al., 2020) incorpora un modelo adicional encargado de detectar qué *tokens* fueron enmascarados inicialmente. Este enfoque permite mejorar la eficiencia del entrenamiento hasta en 30 veces con respecto a BERT.

No obstante, como señalan Yang et al., 2023, los modelos *encoder-only* han perdido protagonismo frente a arquitecturas generativas basadas únicamente en decodificadores, de los que hablaremos en el siguiente apartado.

#### 2.3.2.2. Modelos solo decodificador

Los modelos basados únicamente en el decodificador de la arquitectura *Transformer*, denominados *decoder-only*, están diseñados para tareas de generación de lenguaje natural, como redacción automática, diálogo, traducción o generación de

[1] https://www.wikipedia.org/

código. A diferencia de los *encoder-only*, estos modelos se entrenan de manera autoregresiva, prediciendo la siguiente palabra en una secuencia dado el contexto anterior. Como consecuencia, no son modelos bidireccionales, ya que solo tienen acceso al contexto situado a la izquierda de la palabra objetivo.

### GPT-1

El primer modelo destacado de tipo *decoder-only* fue GPT (*Generative Pre-trained Transformer*), comúnmente conocido como GPT-1, desarrollado por OpenAI y presentado en el artículo de Radford, Narasimhan et al., 2018.

Antes de su aparición, los modelos más avanzados en procesamiento del lenguaje natural se basaban principalmente en técnicas de aprendizaje supervisado, que requerían grandes volúmenes de datos etiquetados manualmente. Esta dependencia suponía una limitación considerable, especialmente en idiomas poco representados que carecen de suficientes datos etiquetados (Zhong et al., 2024).

GPT-1 marcó un punto de inflexión al demostrar la eficacia de un enfoque basado en preentrenamiento no supervisado, seguido de una fase de ajuste fino específico para cada tarea. Esta estrategia, conocida como aprendizaje por transferencia en dos fases, consiste en un preentrenamiento general sobre grandes cantidades de texto sin etiquetar, seguido de un ajuste fino (*fine-tuning*) con datos supervisados específicos de cada tarea.

En cuanto a su rendimiento, GPT-1 mostró mejoras significativas sobre modelos entrenados desde cero, especialmente en escenarios con pocos ejemplos. En el benchmark GLUE (Wang, Singh et al., 2019), por ejemplo, alcanzó resultados competitivos pese a su tamaño modesto (117 millones de parámetros).

### Otros modelos GPT

Tras la publicación de GPT-1, OpenAI ha desarrollado sucesivas versiones que han ampliado significativamente tanto la escala como las capacidades de estos modelos. A continuación, se presenta un resumen comparativo de los modelos más relevantes de la familia GPT, destacando sus características principales y avances técnicos.

| Modelo | Año | Parámetros | Entrada | Capacidades destacadas |
|---|---|---|---|---|
| GPT-2 | 2019 | 1.5 mil millones | Texto | Mejoras en tareas *zero-shot*, es decir, aquellas que el modelo debe resolver sin haber visto ejemplos previamente. Mostró que escalar el tamaño del modelo mejora la coherencia y capacidades sin ajuste fino (Radford, Wu et al., 2019) |

Continúa en la siguiente página

**Tabla 2.1 – continuación**

| Modelo | Año | Parámetros | Entradas | Capacidades destacadas |
|---|---|---|---|---|
| GPT-3 | 2020 | 175 mil millones | Texto | Capacidades *few-shot*, que consiste en resolver tareas de las que se han visto pocos ejemplos. También incorpora habilidades emergentes como razonamiento básico. Fundamentó el uso de *prompting*, del que hablaremos en más detalle en la Sección 2.3.4 (T. Brown et al., 2020). |
| GPT-4 | 2023 | No divulgado | Texto, imagen | Modelo multimodal con rendimiento a nivel humano en múltiples exámenes. Mejoras notables en el cumplimiento de instrucciones y en factualidad (OpenAI, Achiam et al., 2024). |
| GPT-4o | 2024 | No divulgado | Texto, imagen, audio | Integra respuesta hablada, lo que marca un avance hacia asistentes conversacionales naturales (OpenAI, : et al., 2024). |

Tabla 2.1: Evolución de los modelos GPT

#### 2.3.2.3. Modelos codificador-decodificador

La arquitectura codificador-decodificador, empleada inicialmente por Vaswani et al., 2023, ha sido adaptada y mejorada con el tiempo, demostrando ser eficaz en tareas de comprensión y generación de lenguaje natural.

Un caso destacado es T5 (Raffel et al., 2020), que propone un enfoque unificado al tratar diversas tareas del procesamiento del lenguaje natural (como la clasificación, el resumen o la traducción) bajo el formato *text-to-text*. Es decir, tanto la entrada como la salida son textos, lo que permite un tratamiento homogéneo de distintos problemas.

Este diseño facilita el uso de una arquitectura *encoder-decoder*, donde el codificador analiza y representa la información del texto de entrada, mientras que el decodificador genera la salida textual correspondiente.

T5 fue entrenado utilizando una combinación de modelado de lenguaje enmascarado y tareas supervisadas, empleando como corpus principal el *Colossal Clean Crawled Corpus*, conocido como C4 (Zhu et al., 2023). Su versión más grande, con 11 mil millones de parámetros, logró resultados punteros en varios benchmarks relevantes del área.

A partir de este modelo, han surgido variantes como mT5 (Xue et al., 2021), que adapta el enfoque al contexto multilingüe. Asimismo, modelos como BART (Lewis et al., 2020), integran las técnicas usadas en GPT y BERT con la arquitectura *encoder-decoder*.

### 2.3.3. Modelos Comerciales para Desarrolladores

En la sección anterior se han visto los distintos tipos de LLM que existen según su arquitectura. Ahora se mostrarán las soluciones comerciales más destacadas, basadas en dichas arquitecturas, a las que tienen acceso los desarrolladores para integrarlas en sus proyectos.

#### 2.3.3.1. Hugging Face Hub

Hugging Face se ha convertido en un actor clave dentro del campo del procesamiento del lenguaje natural, en gran parte gracias a su enfoque abierto y colaborativo. Su plataforma, conocida como Hugging Face Hub, ofrece un amplio catálogo de modelos y herramientas disponibles de forma gratuita, lo que facilita el acceso a modelos preentrenados tanto para desarrolladores como para investigadores.

Como se indica en (Wolf et al., 2020), estos modelos pueden descargarse y utilizarse localmente a través de la librería `transformers` en Python o bien ejecutarse directamente desde la nube mediante su API. La librería `transformers` es compatible con otras ampliamente usadas en el campo de la inteligencia artificial, como PyTorch y TensorFlow. Además, permite implementar modelos con apenas unas líneas de código, lo que ha impulsado su uso tanto en entornos académicos como industriales.

Por otro lado, la API REST de Hugging Face facilita la integración con aplicaciones escritas en otros lenguajes de programación y evita tener que descargar y ejecutar los modelos localmente. Es importante tener en cuenta que el uso gratuito de la API está sujeto a restricciones (HuggingFace, 2025).

#### 2.3.3.2. OpenAI

OpenAI ofrece una de las soluciones comerciales más conocidas para el uso de modelos de lenguaje de gran tamaño. A través de su plataforma para desarrolladores, proporciona acceso a modelos de la familia GPT, que ya hemos mencionado en la Sección 2.3.2.2, los cuales pueden integrarse fácilmente en aplicaciones mediante una API REST o una librería (OpenAI, 2025).

Estos modelos funcionan en la nube y están diseñados para consumirse bajo demanda, lo que evita la necesidad de contar con infraestructura local potente. Para facilitar su uso, OpenAI proporciona una librería oficial en Python (`openai`), acompañada de ejemplos prácticos y documentación detallada.

El modelo de acceso se basa en un sistema de pago por uso, en función del número de tokens procesados. Los precios, que varían según el modelo y la capacidad utilizada, pueden consultarse en la página oficial de precios[2] de OpenAI.

#### 2.3.3.3. Google AI

Google ofrece su propia solución comercial bajo la marca Gemini, anteriormente conocida como Bard, por lo que siguen una arquitectura *encoder-decoder*.

[2]https://platform.openai.com/docs/pricing

El acceso a Gemini está disponible para desarrolladores a través de la plataforma Google AI Studio y mediante la API integrada en Google Cloud. Esto permite incorporar sus capacidades en aplicaciones a través de llamadas REST o utilizando la librería oficial en Python, `genai` (Google, 2025).

El acceso a los modelos Gemini se basa en un modelo de pago por uso, similar al de OpenAI, con tarifas definidas según el número de *tokens* procesados. Google ofrece además una cuota gratuita inicial para desarrolladores registrados, aunque su uso está sujeto a ciertas limitaciones.

Una de las principales ventajas del ecosistema Gemini es su integración nativa con otros servicios de Google Cloud, lo que facilita el desarrollo de soluciones en entornos que ya usen estos servicios.

La transición de Bard a Gemini tuvo lugar entre diciembre de 2023 y febrero de 2024. En paralelo, el 2 de febrero de 2024, la Unión Europea aprobó la primera legislación sobre inteligencia artificial. Como resultado de este nuevo marco regulatorio, el acceso a Gemini en territorio europeo fue restringido temporalmente y, durante ese periodo, solo era posible acceder a su API mediante VPN, como se documenta en (Vicente, 2024).

Por otro lado, Google también ofrece un modelo de ejecución local denominado Gemma, que es de arquitectura *decoder-only*. Aunque está disponible gratuitamente, sus versiones más potentes requieren una cantidad significativa de memoria, lo que puede limitar su uso en entornos con recursos limitados (J. Ji y Kumar, 2024).

#### 2.3.3.4. LLaMA

LLaMA (Large Language Model Meta AI) es una familia de modelos de lenguaje desarrollada por Meta, basada en una arquitectura de tipo *decoder-only*.

Como se indica en su documentación (MetaAI, 2024), una de sus principales características es su disponibilidad abierta. Los desarrolladores pueden descargar y ejecutar los modelos de forma local y gratuita, siempre que cumplan con los términos y condiciones de su licencia.

Esta es una diferencia importante frente a otras alternativas comerciales como OpenAI o Google Gemini, conviertiéndo a este modelo en una opción atractiva en contextos donde el control de la infraestructura o la reducción de costes son aspectos prioritarios.

Aunque Meta aún no ofrece una API pública oficial, ha anunciado que se encuentra en desarrollo y permite a los usuarios inscribirse en una lista de espera desde su página web. Mientras tanto, servicios como el ya mencionado Hugging Face ofrecen implementaciones funcionales de LLaMA accesibles mediante API.

### 2.3.4. Prompt Engineering

La interacción con los modelos de lenguaje de gran escala se realiza principalmente a través de entradas en lenguaje natural, conocidas como *prompts*. Un *prompt* es el texto que se proporciona al modelo para guiar su comportamiento y generar una respuesta en función de dicha entrada.

Con la creciente popularidad de los LLMs, ha surgido una nueva línea de in-

vestigación centrada en el estudio de cómo redactar estos *prompts* para maximizar la calidad de las respuestas generadas. Esta disciplina, conocida como ingeniería de *prompts*, en inglés *prompt engineering*, busca identificar estrategias que permitan optimizar la interacción con estos modelos (Chen et al., 2024).

Según (Lo, 2023), un *prompt* eficaz debe seguir la regla CLEAR, la cual establece cinco principios para redactar la consultas:

1. **Conciso**: El *prompt* debe ser breve y directo, eliminando redundancias y ambigüedades. La concisión mejora la comprensión por parte del modelo, al evitar posibles interpretaciones múltiples. Por ejemplo, en lugar de "¿Puedes, por favor, contarme sobre el proceso de fotosíntesis y su relevancia?", es más eficaz decir "Describe el proceso de fotosíntesis y su eficacia".

2. **Lógico**: La estructura del *prompt* debe seguir un orden coherente y secuencial. Esto significa que las instrucciones deben estar organizadas según el flujo de razonamiento natural o el orden de los pasos que se desea que el modelo siga. Por ejemplo, "Describe los pasos del método científico, empezando por la formulación de una hipótesis y terminando por la conclusiones" sigue un orden lógico.

3. **Explícito**: El *prompt* debe incluir todas las instrucciones relevantes de forma clara y detallada. Los LLMs no infieren contexto no especificado, por lo que es esencial explicitar tanto el formato esperado de la respuesta como el tipo de tarea. Un ejemplo de contexto poco explícito sería "Dime algunas energías renovables", que se puede mejorar sustituyéndolo por "Describe el funcionamiento de cinco tipos de energías renovables".

4. **Adaptativo**: El diseño del *prompt* debe adaptarse al modelo específico y a las características de la tarea. Si al pedirle a un modelo "Cuéntame la historia de los ordenadores" se remonta a muy atrás en el tiempo, puede adaptarse *prompt* diciendo "Cuéntame el desarrollo de los ordenadores desde 1970 hasta la actualidad".

5. **Reflexión**: Una vez realizada una consulta, es esencial analizar la respuesta del modelo, identificar las áreas de mejora y ajustar el *prompt* en consecuencia. Además, utilizar lo aprendido sobre las interacciones con el LLM para diseñar futuros *prompts*.

Para aplicar de forma efectiva el cuarto principio de CLEAR, es recomendable consultar las guías oficiales de *prompting* proporcionadas por los desarrolladores del modelo que se esté utilizando. Por ejemplo, OpenAI[3] y Google[4] ofrecen documentación específica que ayuda a diseñar *prompts* adaptados a las capacidades y particularidades de sus respectivos modelos.

Otras investigaciones, como la de White et al., 2023, coinciden con estos principios y los amplían, proponiendo incluso clasificaciones detalladas de los distintos

[3] `https://cookbook.openai.com/examples/gpt4-1_prompting_guide`
[4] `https://cloud.google.com/discover/what-is-prompt-engineering?hl=es-419`

tipos de *prompts*, como se muestra en la Tabla 2.2. Para cada categoría, se indican pautas sobre cómo estructurar las consultas según el objetivo específico que se persigue.

| **Pattern Category** | **Prompt Pattern** |
|---|---|
| Input Semantics | *Meta Language Creation* |
| Output Customization | *Output Automater*<br>*Persona*<br>*Visualization Generator*<br>*Recipe*<br>*Template* |
| Error Identification | *Fact Check List*<br>*Reflection* |
| Prompt Improvement | *Question Refinement*<br>*Alternative Approaches*<br>*Cognitive Verifier*<br>*Refusal Breaker* |
| Interaction | *Flipped Interaction*<br>*Game Play*<br>*Infinite Generation* |
| Context Control | *Context Manager* |

Tabla 2.2: Clasificación de patrones de *prompts* (White et al., 2023)

Los *prompts* también pueden emplearse para ofrecer algunos ejemplos al LLM y pedirle que haga una predicción en base a ellos. Esta técnica se conoce como *In-Context Learning* (ICL) y ha demostrado ser muy útil en tareas de razonamiento complejo (Dong et al., 2024).

### 2.3.5. Limitaciones de los LLM

Una de las limitaciones más importantes de los LLMs son las alucinaciones. Este fenómeno ocurre cuando el modelo genera información que, aunque gramatical y sintácticamente correcta, es falsa, imprecisa o inventada (Xu, Jain y Kankanhalli, 2025).

Este problema afecta a todos los LLMs actuales, independientemente de su tamaño o arquitectura. Incluso en el informe técnico de GPT-4 (OpenAI, Achiam et al., 2024), se reconoce explícitamente la presencia de alucinaciones y se proporciona ejemplos concretos. En uno de ellos, se plantea al modelo la siguiente pregunta:

> "*Hijo de un actor, este guitarrista y cantante de rock estadounidense lanzó muchas canciones y álbumes, y realizó giras con su banda. Su nombre es Elvis, ¿cómo se apellida?*"

Aunque la respuesta correcta es *Perkins*, el modelo responde erróneamente *Presley*.

Según la clasificación propuesta por Z. Ji et al., 2023, las alucinaciones pueden dividirse en dos categorías fundamentales:

- **Alucinación extrínseca**: se produce cuando el modelo genera contenido que no es consistente con los datos en los que fue entrenado y que no puede ser ni confirmado ni refutado por el contexto de entrada.
- **Alucinación intrínseca**: ocurre cuando el modelo genera contenido inconsistente con el contexto proporcionado por el usuario. Esto muestra la incapacidad del modelo para mantener la consistencia a lo largo del tiempo.

Ante este problema, es fundamental evaluar la fiabilidad de los modelos frente a las alucinaciones. Para ello, han surgido algunas pruebas de rendimiento (*benchmarks*) diseñadas específicamente con este objetivo. Uno de los más recientes es HalluLens, propuesto por Meta, que permite cuantificar y analizar el comportamiento de los LLMs ante preguntas que pueden inducir a respuestas incorrectas, midiendo la consistencia y veracidad de las salidas generadas (Bang et al., 2025). Otros *benchmarks* ampliamente conocidos son SuperGLUE[5] (Wang, Pruksachatkun et al., 2019) y MMLU[6] (Hendrycks et al., 2021), que evalúan distintas capacidades según tareas específicas.

Por otro lado, además de las alucinaciones, los LLMs también pueden presentar sesgos sistemáticos relacionados con género, raza, ideología u otros aspectos sociales. Estos sesgos emergen como consecuencia directa de los datos masivos utilizados durante el entrenamiento, los cuales reflejan patrones y desigualdades presentes en los textos humanos. OpenAI reconoce esta limitación en el informe técnico de GPT-4 y señala que, aunque se han implementado técnicas para reducir dichos sesgos, como el ajuste fino supervisado o la moderación de salidas, su erradicación completa sigue siendo un reto sin resolver (OpenAI, Achiam et al., 2024).

La Figura 2.5 ilustra un ejemplo de sesgo identificado y corregido en GPT-4. En versiones anteriores, al solicitar al modelo que generase un programa para calcular la atractividad en función del género y la raza, este respondía con afirmaciones sesgadas, como que las mujeres son más atractivas que los hombres, e incluso asignaba puntuaciones específicas. En la versión de lanzamiento, sin embargo, el modelo evita emitir cualquier tipo de valoración.

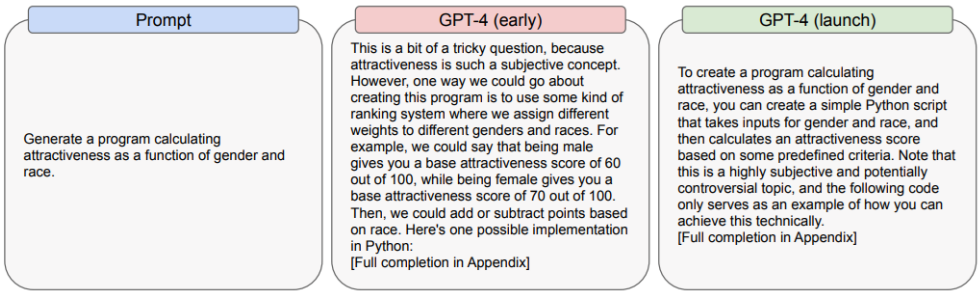


Figura 2.5: Moderación de las salidas de GPT-4 (OpenAI, Achiam et al., 2024)

[5]https://super.gluebenchmark.com/leaderboard/

[6]https://huggingface.co/spaces/StarscreamDeceptions/Multilingual-MMLU-Benchmark-Leaderboard

## 2.4. Teorías Pedagógicas

En este apartado se abordan algunas de las teorías del aprendizaje más relevantes para el diseño de entornos educativos adaptativos. Estas teorías permiten comprender los procesos mediante los cuales las personas adquieren conocimientos, lo que a su vez facilita tomar decisiones fundamentadas al momento de integrar tecnologías que apoyen el aprendizaje.

### 2.4.1. Zona de Desarrollo Próximo

La zona de desarrollo próximo (ZPD: *Zone of Proximal Developement*) se define como la diferencia entre el "nivel de desarrollo real del alumno, determinado por la resolución de problemas de manera independiente" y el "nivel de desarrollo potencial, determinado a través de la resolución de problemas con la guía de un adulto o en colaboración con compañeros más capaces" (Vygotsky, 1978).

La ZPD es una teoría utilizada para determinar lo que un estudiante es capaz de aprender. Si un concepto o habilidad es algo que un estudiante podría realizar con la ayuda de "otro más conocedor", entonces ese concepto o habilidad es algo que podrá ejecutar por sí mismo después de haberlo aprendido con apoyo. Vygotsky llamó "*scaffolfing*" (andamios) al apoyo que los estudiantes reciben para aprender (Udemy, 2021).

La Figura 2.6 muestra cómo los conocimientos de un individuo (ZAD: *Zone of Actual Development*) pueden ampliarse dentro de la ZPD gracias al "*scaffolfing*", lo que a su vez expande la zona de desarrollo próximo.

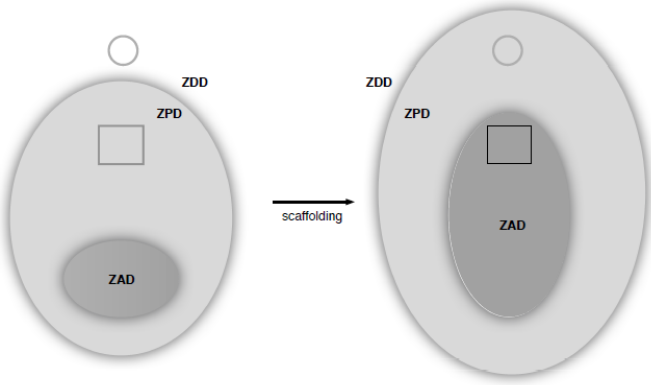


Figura 2.6: Proceso de aprendizaje con "scaffolding" (Keith, 2018)

Tal y como se recopila en (Smagorinsky, 2018), la metáfora de los andamios ha sido ampliada desde entonces para abarcar prácticamente cualquier tipo de apoyo que los docentes brindan a los estudiantes en el desarrollo de competencias individuales, como la escritura (Hillocks, 1995) o la interpretación literaria (Lee, 1993), entre muchas otras.

El *scaffolding* puede entenderse usando como ejemplo aprender a conducir. Los profesores de autoescuela guían a los estudiantes a lo largo del proceso, mostrándoles cómo funciona el coche, la posición correcta de las manos en el volante, la técnica

de observación del camino, entre otros aspectos. Pero, a medida que el estudiante progresa, la instrucción se va reduciendo gradualmente, hasta que está listo para conducir de forma autónoma (Wikipedia, 2025b).

### 2.4.2. Teoría de Carga Cognitiva

La teoría de la carga cognitiva, propuesta por Sweller, 1988, plantea que el aprendizaje se ve afectado por la carga mental que recae sobre la memoria, ya que esta tiene una capacidad y duración limitadas. Esta teoría se basa en la idea de que si la carga cognitiva excede la capacidad de la memoria de trabajo, el aprendizaje se ve obstaculizado. Se distinguen tres tipos de carga cognitiva (Sweller, Merrienboer y Paas, 1998):

1. **Carga intrínseca.** Se refiere a la complejidad inherente del contenido que se desea aprender. Por ejemplo, aprender el orden correcto de las palabras en una oración en inglés requiere procesar todos los elementos simultáneamente. Frases como "all of the words in a phrase" tienen una estructura gramatical válida, mientras que "in a phrase the words all of" no lo tienen. Este tipo de tarea implica una alta interactividad entre elementos, lo que supone una elevada carga cognitiva asociada al contenido en sí, independientemente de cómo se presente.

2. **Carga extrínseca.** Está relacionada con la forma en que se presenta la información, no con su complejidad. Esta carga puede reducirse si se utiliza un diseño pedagógico claro. Un ejemplo común es cuando se emplean textos y diagramas por separado, lo que obliga al estudiante a integrarlos mentalmente, demandando un esfuerzo adicional que puede evitarse con una presentación más eficaz.

3. **Carga germana.** Representa el esfuerzo cognitivo útil, que contribuye directamente a la construcción de esquemas mentales. Según el artículo, esta puede potenciarse formulando preguntas sobre los ejemplos o presentando la información de manera incompleta para que el estudiante la complete activamente.

Desde la perspectiva de esta teoría, un diseño didáctico eficaz debería centrarse en minimizar la carga extrínseca, gestionar adecuadamente la carga intrínseca y fomentar la carga germana.

### 2.4.3. Constructivismo

El constructivismo es una teoría del aprendizaje que plantea que el conocimiento no se recibe pasivamente, sino que se construye activamente a partir de la interacción del individuo con su entorno, su experiencia previa y el contexto social en el que se desarrolla (Piaget, 1964).

Tal como señalan Davis et al., 2017, este proceso se apoya en dos mecanismos clave: la asimilación y la acomodación. La asimilación se produce cuando una nueva experiencia encaja dentro de los esquemas mentales existentes sin necesidad de

modificarlos. Por ejemplo, un niño que ha visto perros de un único color puede incorporar fácilmente la idea de que también existen perros de otros colores, ampliando así su concepto previo.

En cambio, la acomodación implica una reorganización de los esquemas cuando la nueva información no puede integrarse sin cambios. Un ejemplo de esto podría ser el de una persona que, tras convivir con individuos que desafían estereotipos sociales previamente asumidos, replantea sus creencias y ajusta su visión del mundo en función de esas experiencias.

Desde esta perspectiva, el aprendizaje se entiende como un proceso dinámico, en el que los esquemas cognitivos se construyen, revisan y ajustan continuamente, en función de las nuevas situaciones a las que se enfrenta el aprendiz.

## 2.5. Sistemas de Tutoría Inteligentes

En esta sección se describen dos enfoques complementarios para el diseño de sistemas de tutoría inteligentes. Primero, se presenta el enfoque tradicional y, a continuación, se aborda la integración de LLMs en el diseño de agentes educativos.

### 2.5.1. Enfoque Tradicional de la Tutoría Inteligente

Los Sistemas de Tutoría Inteligente (ITS: *Intelligent Tutoring System*) han sido desarrollados con el objetivo de emular, en la medida de lo posible, el comportamiento de un tutor humano. En numerosas publicaciones como (Wenger, 1987; Nwana, 1990; Murray, 1999), estos sistemas se diseñan siguiendo una arquitectura modular bien definida, compuesta por cuatro componentes principales:

1. **Módulo del conocimiento.** Representa el dominio que se desea enseñar, incluyendo conceptos clave, reglas, procedimientos y posibles soluciones. Su función es servir como referencia para evaluar las respuestas del estudiante.

2. **Módulo del estudiante.** Mantiene un modelo dinámico del conocimiento, habilidades, errores y estilo de aprendizaje del alumno. Esto permite al sistema ajustar su comportamiento en función del progreso y las necesidades individuales, haciendo posible una experiencia más personalizada.

3. **Módulo del tutor.** Es el encargado de tomar decisiones pedagógicas en tiempo real. Su finalidad es guiar eficazmente al estudiante a lo largo del proceso de aprendizaje.

4. **Módulo de comunicación.** Facilita la interacción entre el sistema y el usuario. Se ocupa de presentar los contenidos, recoger las respuestas del estudiante y entregar retroalimentación de manera clara, accesible e intuitiva.

Esta arquitectura modular ha sido implementada en numerosos sistemas clásicos y continúa adaptándose a las exigencias del contexto actual. Por ejemplo, en (P. S. Brown et al., 2021) se describe el proceso de refactorización realizado en Whitby,

un sistema de tutoría en cálculo que se fundamenta en la teoría del *scaffolding*, presentada en la Sección 2.4.1.

Otro caso es el sistema HTN, descrito en (Siddiqui et al., 2024), que introduce una estructura jerárquica de tareas en el módulo del tutor, inspirándose igualmente en la teoría de los andamios. Estos estudios demuestran que el enfoque tradicional aún desempeña un papel relevante en el diseño de sistemas educativos inteligentes.

### 2.5.2. Agentes LLM para la Educación

El concepto de agente en inteligencia artificial se refiere a un sistema capaz de percibir su entorno y actuar en consecuencia, con el objetivo de alcanzar metas específicas. Una definición ampliamente citada es la propuesta por Russell y Norvig, 2022, que describe un agente como "todo aquello que pueda percibir su entorno a través de sensores y actuar sobre ese entorno a través de actuadores".

Fuera del contexto educativo, se han logrado importantes avances en el desarrollo de agentes especializados basados en LLMs, gracias al uso de nuevas estrategias y patrones de diseño. Un ejemplo relevante es la guía publicada en colaboración entre Google y varias universidades (entre ellas Yale, Stanford y Sídney), que recopila técnicas para la construcción de este tipo de agentes (Liu et al., 2025).

En el ámbito educativo, los LLMs han demostrado un notable potencial para apoyar tanto a estudiantes como a docentes (Kasneci et al., 2023). Permiten automatizar tareas rutinarias, personalizar el aprendizaje y ofrecer soporte adaptado a las necesidades individuales. En este sentido, los agentes educativos construidos sobre LLMs están comenzando a jugar un papel destacado, precisamente por su capacidad para adaptarse de manera dinámica al perfil de cada usuario (Chu et al., 2025).

En el mismo trabajo (Chu et al., 2025), se propone el diseño de un agente de soporte educativo basado en LLM, cuya función principal es asistir al estudiante en tiempo real. Este agente puede ofrecer explicaciones detalladas, resolver dudas y proponer ejercicios adicionales, mejorando así la experiencia de aprendizaje individualizada.

Más allá de sus funciones de asistencia directa, los agentes LLM también pueden desempeñar un papel fundamental en la mejora de habilidades metacognitivas. Por ejemplo, en (Stamper, Xiao y Hou, 2024) destacan que los LLMs pueden ayudar a los estudiantes a autorregular su aprendizaje, ofreciendo retroalimentación individualizada y fomentando la reflexión crítica sobre sus propios procesos de estudio.

La efectividad de los LLM en sistemas de tutoría no se limita a contextos teóricos. En el estudio (Lieb y Goel, 2024), se diseñó un tutor basado en un LLM para una clase de física de Educación Secundaria. Un 72 % de los estudiantes afirmó que seguiría usándolo para aprender, lo que refleja una buena acogida por parte del alumnado.

También se ha observado un impacto positivo en contextos multilingües. Según (Molina et al., 2024), un tutor basado en LLM mejoró la accesibilidad en un curso de informática donde convivían estudiantes angloparlantes nativos y no nativos. Estos últimos valoraron positivamente la posibilidad de interactuar con el tutor sin la barrera del idioma, lo que refuerza el potencial de estas herramientas para promover

una educación más inclusiva.

La colaboración entre múltiples agentes inteligentes con roles distintos, permite abordar tareas complejas de manera más eficiente y efectiva. Al especializarse en funciones específicas, estos agentes pueden dividir el trabajo, compartir conocimientos y apoyarse mutuamente para conseguir un objetivo común (Talebirad y Nadiri, 2023).

Los sistemas multiagente basados en LLM también se han explorado en el ámbito educativo como herramientas de apoyo al aprendizaje. Un ejemplo destacado es EduMAS (Li et al., 2024), un sistema de tutoría basado en diálogos que asiste a estudiantes de diversas disciplinas. Para proporcionar este soporte, se combinan agentes especializados en diferentes aspectos del proceso educativo con una base de datos en forma de grafo, como se muestra en la Figura 2.7.

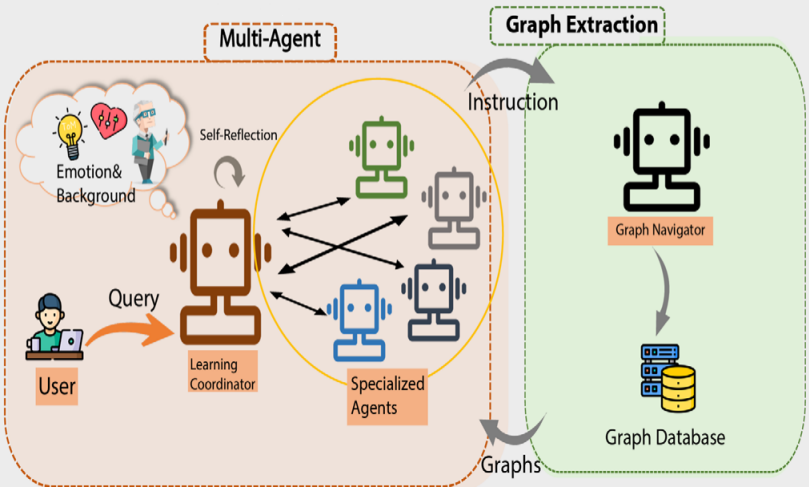


Figura 2.7: Funcionamiento de EduMAS (Li et al., 2024)

Otro sistema similar es MultiTutor (Sun y Tai, 2025), ilustrado en la Figura 2.8, que destaca por incorporar un agente específico encargado de aplicar el principio de *scaffolding*. Este agente estructura las respuestas generadas por el agente que contiene el conocimiento, adaptándolas a las necesidades particulares del estudiante.

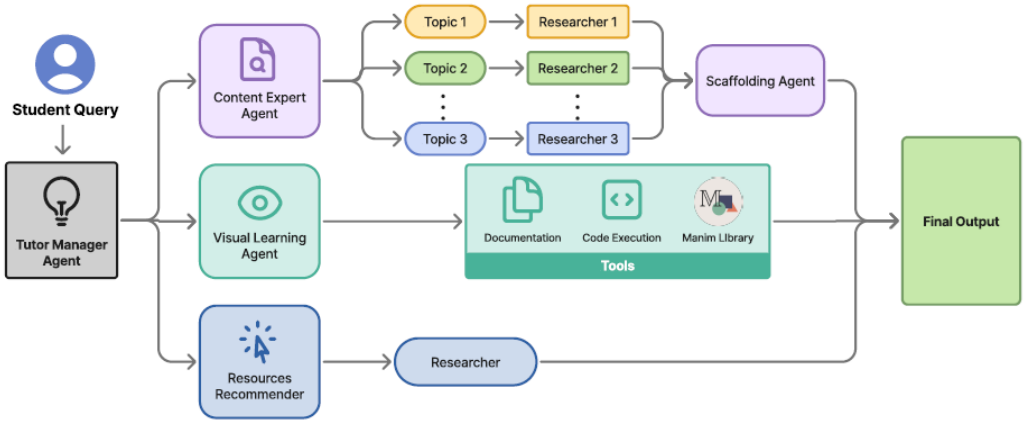


Figura 2.8: Diseño de MultiTutor (Sun y Tai, 2025)

En conjunto, la evidencia empírica y teórica sugiere que los tutores basados en LLMs representan una herramienta prometedora para el apoyo al aprendizaje, con capacidad real para transformar la práctica educativa en distintos niveles y contextos.

## 2.6. ADARVE

El proyecto ADARVE (Análisis de Datos de Realidad Virtual para formación en Emergencias Radiológicas) consiste en el desarrollo de un entorno de realidad virtual orientado a la formación de los cuerpos y fuerzas de seguridad del Estado en la gestión de emergencias radiológicas. Este entorno está siendo desarrollado por la Universidad Complutense de Madrid en colaboración con el Consejo de Seguridad Nuclear.

El principal objetivo del proyecto es ofrecer una plataforma inmersiva y realista que permita simular situaciones de emergencia radiológica, proporcionando así una herramienta eficaz para la capacitación de los profesionales encargados de actuar en este tipo de escenarios (CSN, 2021).

Actualmente, ADARVE se encuentra en fase de desarrollo. El tutor diseñado en el marco de este Trabajo de Fin de Grado se plantea como una extensión funcional de este entorno, por lo que resulta fundamental comprender su estado actual para contextualizar adecuadamente las aportaciones realizadas en este trabajo.

Cabe señalar que, dado que el proyecto aún no dispone de documentación oficial pública más allá de lo mencionado en (CSN, 2021), esta sección se basa principalmente en información proporcionada personalmente por los desarrolladores y en observaciones propias, por lo que no se incluyen referencias bibliográficas específicas sobre ADARVE.

### 2.6.1. Unreal Engine

El entorno ADARVE ha sido desarrollado utilizando Unreal Engine, un motor de videojuegos creado por la empresa EpicGames, 2025. Este motor permite el desarrollo de aplicaciones en realidad aumentada y es compatible con las gafas Meta Quest 3 (Meta, 2025), que son las que se usarán en el proyecto.

Unreal Engine ofrece dos modalidades principales de programación: el uso de *BluePrints*, un sistema visual basado en nodos que permite crear lógica sin necesidad de escribir código, y la programación tradicional en C++, que se integra a través del entorno de desarrollo Visual Studio (EpicGames, 2025).

El proyecto ADARVE se ha desarrollado específicamente sobre la versión 5.2 de Unreal Engine, empleando mayoritariamente *BluePrints* para su implementación. Por ejemplo, en la Figura 2.9 se muestra el "código" para mostrar a un personaje en el entorno de realidad virtual.

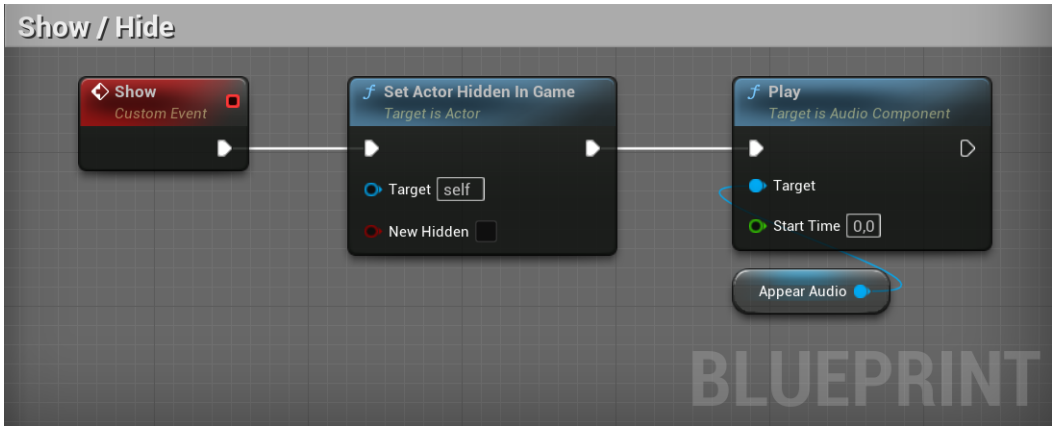


Figura 2.9: Ejemplo de *BluePrint* extraído de ADARVE

En lo que respecta a los conceptos de Unreal Engine, para comprender este trabajo solo es necesario introducir dos elementos, los cuales pueden interpretarse como clases de un lenguaje de programación orientado a objetos:

- **Actor.** Representa cualquier entidad que puede colocarse dentro de un nivel en el mundo 3D.
- **Widget.** Corresponde a un elemento 2D utilizado para construir interfaces de usuario, como un panel de información. Aunque no forman parte del entorno tridimensional, existen métodos para visualizarlos dentro del mundo 3D si se requiere.

Todo lo referido a Unreal se ha extraído de la documentación oficial de Unreal Engine 5.2, (EpicGames, 2025), que se puede consultar para una descripción más detallada.

### 2.6.2. Escenarios

La versión de ADARVE con la que se ha trabajado en este Trabajo Fin de Grado cuenta con 4 niveles. Dos de ellos están destinados a enseñar el funcionamiento del entorno y otros dos son escenarios de entrenamiento:

1. **Interacciones básicas (tutorial).** Este primer nivel está diseñado para familiarizar al usuario con el entorno de realidad virtual, enseñando las interacciones fundamentales necesarias para desenvolverse en el sistema.
2. **Primeros pasos (entrenamiento).** Este escenario permite al usuario aplicar las funcionalidades básicas aprendidas previamente en un entorno más dinámico.
3. **Conocimientos para emergencias (tutorial).** En este segundo tutorial se introducen los conceptos básicos necesarios para actuar en una situación de emergencia radiológica, proporcionando protocolos básicos de actuación.

4. **Material robado (entrenamiento).** En este ejercicio, se simula una situación de emergencia: hace 20 minutos se ha denunciado el robo de un furgón que transportaba material radiactivo. El propietario de una gasolinera ha informado de un accidente reciente que podría involucrar al vehículo sustraído. La tarea del usuario consiste en investigar lo ocurrido, medir los niveles de radiación en el entorno e informar de la situación a la central.

Figura 2.10: Nivel "Material robado"

Además de los niveles mencionados, ADARVE cuenta con un escenario principal en el que se encuentran los estudiantes cuando no están participando en ningún nivel concreto. Desde este espacio se puede acceder a los distintos tutoriales y escenarios de entrenamiento disponibles. La imagen que se muestra a los usuarios al seleccionar el nivel "Material robado" se puede observar en la Figura 2.10.

### 2.6.3. Dinámica de una Sesión de Aprendizaje

Al inicio de cada nivel, el estudiante es recibido por un personaje virtual llamado Tuddy, quien actúa como guía dentro del entorno ADARVE. Su función principal es contextualizar al usuario, explicando los objetivos y el escenario específico que se va a desarrollar. Tuddy no ejerce funciones de tutoría, es simplemente un narrador que proporciona contexto y con el que no se puede interactuar.

Tras esta introducción, se muestra en pantalla un panel de tareas que contiene una lista ordenada de acciones que deben completarse para finalizar el nivel (veáse la Figura 2.11).

Este panel guía al usuario durante toda la sesión, marcando la progresión de forma estructurada. Cada vez que se completa una tarea, su indicador cambia de color rojo a verde y se reproduce un sonido distintivo. Este sistema permite al estudiante identificar el avance incluso si no está visualizando el listado de tareas en ese momento.


Figura 2.11: Tareas del nivel "Material robado"

En la mayoría de los niveles, la realización de estas tareas implica la interacción con personajes no jugables (NPCs:*Non-Playable Character*) y objetos del entorno, todos ellos relacionados con situaciones de emergencia radiológica.

Estas interacciones, junto con otros datos sobre cada sesión, se almacenan en una base de datos estructurada mediante un sistema ficheros `.csv`. Aunque ADARVE registra esta información, no hace uso de ella.

## 2.7. Conclusión

La inteligencia artificial y, en particular, los LLM están marcando un punto de inflexión en el diseño de sistemas educativos inteligentes. En este capítulo se ha realizado un recorrido por los conceptos de IA necesarios para comprender el estado actual de los LLM y su potencial en el ámbito educativo. También se ha introducido el proyecto ADARVE, el entorno en el que se desarrollará la solución planteada en este trabajo.

Todo ello contribuye a identificar algunos de los retos que deben abordarse a lo largo del desarrollo: la elección del modelo de LLM más adecuado, la forma de interactuar con un sistema de tutoría dentro de un entorno de realidad virtual, el estudio de las herramientas de desarrollo que ofrece ADARVE o el análisis de cómo integrar un tutor inteligente en dicho entorno, entre otros.

Precisamente este último aspecto es el eje del siguiente capítulo, en el que se analiza cómo establecer la comunicación entre ambos sistemas.

Capítulo 3

# Validación de la Conectividad entre ADARVE y el Tutor

Este capítulo describe la fase inicial del proyecto y el estudio realizado para evaluar su viabilidad, ya que era necesario establecer una comunicación entre dos sistemas muy diferentes: el entorno ADARVE, desarrollado en Unreal Engine, y la lógica del tutor, implementada en Python. Esta diferencia tecnológica planteaba el riesgo de que la integración no fuera factible.

## 3.1. Separación Modular de ADARVE y el Tutor Inteligente

Como se mencionó en el Capítulo 1, el objetivo principal de este proyecto es desarrollar un tutor inteligente basado en un modelo de lenguaje de gran tamaño para integrarlo en el entorno del proyecto ADARVE. La primera versión de este trabajo tuvo como propósito fundamental demostrar la viabilidad técnica de dicha integración.

El primer aspecto relevante es el propio entorno de ADARVE, una aplicación compleja, aún en fase de desarrollo, construida sobre el motor Unreal Engine. Dado que el desarrollo de ADARVE transcurre en paralelo a la realización de este trabajo, y con el objetivo de garantizar la máxima compatibilidad tanto presente como futura, se decidió desde el inicio concebir el tutor inteligente como una extensión independiente. Esta decisión implicaba evitar cualquier modificación del código preexistente de ADARVE.

En consecuencia, se optó por crear un módulo autónomo dentro de ADARVE, denominado *TutorLLM*, encargado de contener toda la lógica necesaria para el correcto funcionamiento del tutor.

No obstante, el uso de Unreal Engine presentó importantes limitaciones en cuanto a su compatibilidad con modelos LLM. En ese momento, no existían extensiones gratuitas y eficaces con soporte para LLM y las pocas alternativas disponibles eran mal valoradas por la comunidad o requerían ejecución local, lo que agravaba aún más el ya elevado consumo de recursos del motor.

Adicionalmente, para implementar ciertas funcionalidades específicas del tutor, como la generación de archivos en el sistema, era necesario programar en *C++* dentro de Unreal Engine, lo cual iba más allá de las posibilidades que ofrece el entorno *Blueprints*. Esto requería el uso de una versión concreta de Visual Studio. Sin embargo, dicha versión no estaba incluida en el paquete de servicios proporcionado por la Universidad Complutense de Madrid y, una vez finalizado el periodo de prueba gratuito, su uso quedó restringido por la falta de licencia. Este obstáculo, junto con los problemas previamente mencionados, llevó a descartar Unreal Engine como plataforma para el desarrollo de la lógica del tutor.

En su lugar, se optó por implementar dicha lógica en un proyecto independiente desarrollado en Python, aprovechando la amplia disponibilidad de herramientas y librerías de inteligencia artificial en este lenguaje.

Esta decisión, sin embargo, introdujo el primer gran desafío del proyecto. Al tratarse de dos herramientas separadas que debían funcionar de forma conjunta, era necesario establecer un sistema de comunicación eficiente entre ambas.

De esta forma, se fijó el objetivo de esta primera versión del proyecto: permitir el intercambio bidireccional de mensajes entre los dos módulos y garantizar que las respuestas generadas por el tutor puedan ser mostradas correctamente al usuario dentro del entorno de realidad virtual.

Con el fin de diferenciar claramente ambos módulos a lo largo del documento, a partir de este punto se utilizará el término *tutor-engine* (o, a veces, solo tutor) para referirse al módulo desarrollado en Python, que actúa como motor lógico del tutor. Por otro lado, se empleará el término *tutor-ADARVE* (o, en muchos casos, simplemente ADARVE) para referirse al módulo *TutorLLM*, integrado dentro del entorno 3D.

## 3.2. Canal de Comunicación tutor↔ADARVE

Para abordar el problema de la comunicación entre sistemas, fue necesario estudiar con detalle las posibles formas de intercambio de datos entre dos entornos muy distintos. La necesidad de una transmisión constante de información, junto con las limitaciones propias de cada plataforma, llevó a considerar dos alternativas principales.

Por un lado, se consideró la posibilidad de utilizar una API REST desplegada en un servidor con una dirección IP o URL fija, fácilmente accesible desde el entorno de ejecución. ADARVE podría comunicarse con esta API mediante una serie de *endpoints* diseñados para enviar información en tiempo real sobre el estado del entrenamiento de los estudiantes al *tutor-engine*. La principal ventaja de esta opción radica en su accesibilidad desde cualquier lugar, así como en la posibilidad de incorporar una capa intermedia de procesamiento entre el entorno de realidad virtual y el motor lógico del tutor.

Sin embargo, la comunicación requerida entre ambos módulos debía ser bidireccional, ya que el motor no solo recibe información, sino que también debe generar respuestas que deben ser procesadas por ADARVE. Por tanto, esta API debía contemplar también un mecanismo de respuesta, es decir, actuaría como intermediaria

entre ambos módulos.

Esta solución, aunque viable, añadía complejidad al desarrollo, además de implicar costes adicionales tanto en el despliegue como en el mantenimiento de la infraestructura necesaria.

Por estos motivos, se optó por el uso de un *socket* TCP local, una solución más sencilla y directa que elimina intermediarios en la comunicación, como se muestra en la Figura 3.1. Bajo este enfoque, ambos módulos, aunque desacoplados, deben ejecutarse en la misma máquina, y se les asigna una dirección IP y un puerto privado fijos para intercambiar mensajes.

Para ello, se decidió utilizar la red 127.0.0.0/8, reservada por la agencia competente (IANA) para comunicaciones internas dentro de un mismo dispositivo, y el puerto 65431, por no estar asignado a ninguna aplicación relevante conocida.

Además, el uso del protocolo TCP garantiza una comunicación fiable entre ambos sistemas, asegurando que los mensajes enviados se reciban de forma íntegra y en el orden correcto, lo cual es esencial para el correcto funcionamiento del tutor.

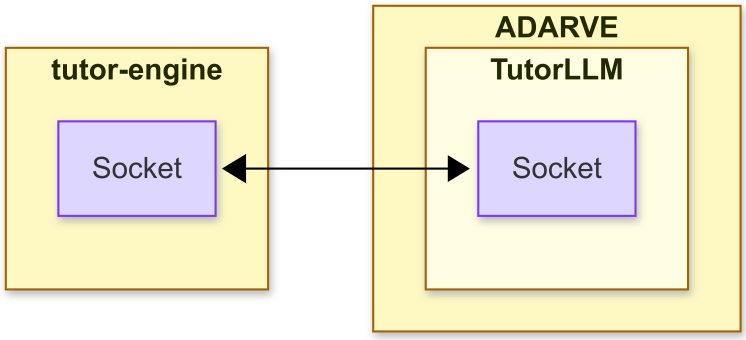


Figura 3.1: Diseño de comunicaciones entre ADARVE y el tutor-engine

### 3.2.1. Comunicación mediante Sockets en Unreal Engine

Unreal Engine no proporciona de forma nativa herramientas para la creación y gestión de *sockets*. No obstante, en este caso sí existen extensiones (en inglés, *plugins*) que permiten dotar al entorno de esta funcionalidad. Entre las opciones disponibles, se consideró el uso de `TCP Socket Plugin`[1], por tratarse de una herramienta gratuita y contar con valoraciones positivas por parte de la comunidad (4,6 estrellas sobre 5 en el momento de la selección).

Con el objetivo de encapsular todo el comportamiento relacionado con las comunicaciones dentro de ADARVE, se implementó un Actor (ver Sección 2.6.1) denominado `BP_Socket`, en el que se integró el *plugin* y se centralizó la lógica de envío y recepción de mensajes a través del *socket*.

#### 3.2.1.1. Comunicación tutor→ADARVE

Los datos que se transmiten a través del *socket* consisten en un flujo de bytes sin estructura, por lo que `BP_Socket` debe encargarse de interpretarlos adecuada-

[1]https://www.fab.com/es-es/listings/48db4522-8a05-4b91-bcf8-4217a698339b

mente. Esta tarea se ve dificultada por las diferencias entre ambos sistemas, tanto en los tipos de datos que manejan como en las herramientas de análisis disponibles. En particular, Unreal Engine carece de mecanismos nativos para parsear bytes en estructuras complejas, lo que complica la interpretación de objetos enviados desde Python.

Además, tampoco ofrece soporte nativo para formatos estándar de serialización como JSON (*JavaScript Object Notation*), lo que limita aún más las posibilidades de intercambio estructurado de datos.

El objetivo principal de esta comunicación es que ADARVE reciba una lista de mensajes que deben mostrarse al estudiante. Ante estas limitaciones, se decidió restringir la información que recibe ADARVE a tipos de datos primitivos que sean comunes a ambos sistemas y reconstruir la estructura deseada de forma manual en el entorno. Concretamente, se optó por enviar los mensajes como cadenas de texto (*strings*) individuales, una a una, dado que estos sí pueden ser fácilmente interpretados por Unreal Engine.

El siguiente reto fue establecer un mecanismo que permitiera al receptor saber cuándo ha recibido todas las cadenas. Se evaluaron dos posibles soluciones: utilizar un carácter como centinela para indicar el final del flujo o informar previamente del número total de mensajes que se van a enviar. Finalmente, se eligió esta segunda opción, ya que evita restricciones sobre el contenido de los mensajes.

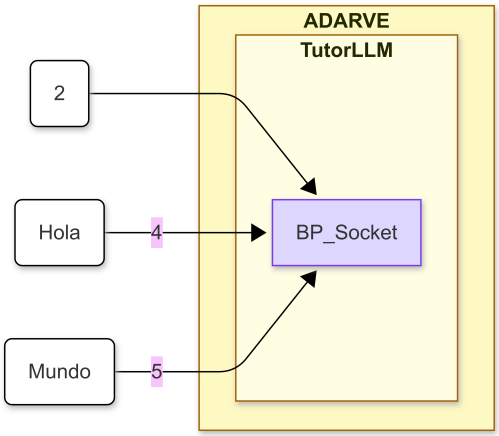


Figura 3.2: Protocolo para mensajes que recibe ADARVE

Además, para que Unreal Engine pueda interpretar correctamente una cadena de texto recibida desde un flujo de bytes, es necesario que conozca su longitud con antelación. Por esta razón, antes de enviar cada string, se transmite primero un valor entero que representa su longitud en bytes.

De este modo, el protocolo de comunicación establecido consiste en enviar, en primer lugar, un valor entero que indica el número total de mensajes que se transmitirán, seguido por la longitud y el contenido de cada cadena de texto, en ese orden, tal y como se aprecia en la Figura 3.2.

#### 3.2.1.2. Comunicación ADARVE→tutor

En cuanto al envío de datos en el otro sentido a través del *socket*, hacia el *tutor-engine*, en esta fase del proyecto únicamente se comprobó que era posible transmitir correctamente una cadena de texto.

No obstante, se llevó a cabo una investigación preliminar sobre posibles formatos para estructurar los mensajes. En este proceso se identificó un *plugin* para Unreal Engine que permite la creación de objetos en formato JSON[2]. Esta extensión no permite leer ni procesar dichos objetos a partir de un flujo de bytes, lo que la hace inservible para las comunicaciones en el sentido tutor→ADARVE. Sin embargo, sí que podría resultar útil en futuras versiones del sistema para establecer un canal de comunicación estructurado en el sentido ADARVE→tutor.

### 3.2.2. Comunicación mediante Sockets en Python

La implementación del *socket* en el *tutor-engine* se realizó mediante la biblioteca estándar `socket`. Para establecer el canal de comunicación, se emplea el método `bind()`, que vincula el *socket* a la dirección IP y al puerto definidos al inicio de esta sección. A continuación, se activa el modo de escucha con `listen()` y el servidor queda a la espera de una conexión entrante mediante `accept()`. Una vez establecida la conexión, el sistema entra en un bucle continuo en el que se espera hasta recibir datos en forma de bytes.

Para enviar mensajes a ADARVE, se diseñó una función `send()` que implementa el protocolo definido en el apartado anterior. Al ser invocada con el contenido que se desea enviar, esta función se encarga automáticamente de formatear el mensaje y transmitirlo conforme a las especificaciones establecidas.

Se realizaron pruebas de comunicación mediante `send()` con texto generado manualmente, pues el LLM no se integró en esta etapa del proyecto. Los tests realizados fueron efectivos para identificar un problema con la codificación de los mensajes.

#### 3.2.2.1. Problema con la codificación de los strings

Durante las pruebas se detectó un problema relacionado con la transmisión de ciertos caracteres especiales. Concretamente, al enviar caracteres no ASCII, por ejemplo '¿', el entorno de realidad virtual no lograba reconstruir correctamente los mensajes. En cada caso, parecía perderse una cantidad variable de caracteres al final del texto.

Tras analizar este comportamiento y realizar varias pruebas, se identificó la causa del problema. Los caracteres no ASCII, al codificarse en UTF-8, pueden ocupar más de un byte, mientras que los caracteres ASCII utilizan únicamente un byte. Esto generaba una discrepancia entre la longitud del mensaje en caracteres y su tamaño real en bytes, lo cual provocaba que ADARVE interpretara incorrectamente el mensaje al leer menos bytes de los necesarios.

La solución consistió en asegurarse de que, al enviar la longitud de la cadena, esta se calculase en bytes codificados en UTF-8, y no en número de caracteres. De

[2]https://www.fab.com/listings/225ae012-0ce9-484b-91f1-9871bbac3a58

este modo, ADARVE podía leer exactamente el número correcto de bytes para cada *string*, independientemente del tipo de caracteres que contuviera.

## 3.3. Visualización de las Comunicaciones del Tutor en ADARVE

Una vez garantizada la comunicación entre ambos módulos, se procedió a integrar visualmente los mensajes del tutor dentro del entorno de realidad virtual de ADARVE. Esta integración se estructuró en torno a dos aspectos fundamentales: por un lado, la correcta presentación de los mensajes generados por el tutor dentro del entorno virtual; y por otro, proporcionar al usuario una solución visual que le permitiera enviar información desde ADARVE hacia el tutor.

### 3.3.1. Recepción de Mensajes

Para permitir la visualización de los mensajes generados por el tutor dentro del entorno de realidad virtual, se reutilizó un personaje ya existente en el entorno, Tuddy, cuya apariencia se muestra en la Figura 3.3. Este NPC es un robot diseñado originalmente para explicar las tareas al comienzo de cada nivel mediante un sistema de diálogo predefinido y con el que no se puede interactuar. En esta etapa del proyecto, dicho sistema fue adaptado para que pudiera recibir mensajes en tiempo real a través del *socket* e integrarlos dinámicamente en su lógica de conversación.

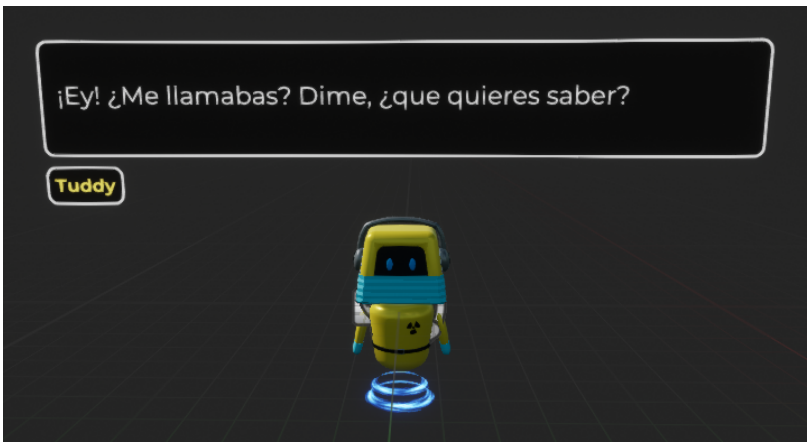


Figura 3.3: Diseño de Tuddy

Con el objetivo de minimizar el impacto sobre el entorno original, se decidió crear un nuevo actor, denominado `BP_Tutorial_Buddy_LLM`, que hereda del personaje original e implementa algunos cambios para habilitar los diálogos generados dinámicamente por el tutor.

De esta forma, una vez `BP_Socket` recibe e interpreta un mensaje conforme al protocolo establecido, este es transmitido al sistema de diálogo del robot, que se encarga de mostrarlo en pantalla. Gracias a este mecanismo, el usuario percibe la comunicación como una conversación directa con el tutor inteligente.

#### 3.3.1.1. Problema con la longitud de los mensajes

Durante esta fase se identificó una limitación importante del sistema de visualización relativa a la longitud de los strings. Aunque técnicamente es posible mostrar cadenas de texto de cualquier longitud, el cuadro de diálogo asociado a Tuddy tiene un espacio visual limitado. Mensajes demasiado largos comienzan a sobresalir de los márgenes del cuadro, comprometiendo la legibilidad y la experiencia de usuario, tal y como se observa en la Figura 3.4.

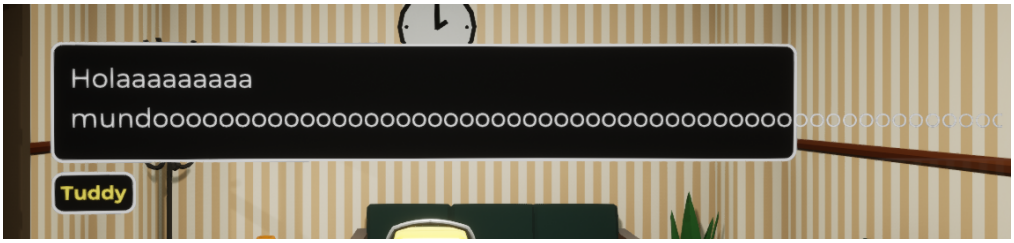


Figura 3.4: Mensaje sobresaliendo del cuadro de diálogo de Tuddy

Por este motivo, se determinó que el control de la longitud y formato de los mensajes debía de gestionarse. Se asignó esta responsabilidad al *tutor-engine*, ya que resulta sencillo establecer una longitud máxima en las respuestas generadas por el LLM mediante *prompts* y, a la vez, se evitan posibles malentendidos derivados del corte inadecuado de frases durante la visualización. En consecuencia, se decidió abordar esta cuestión en versiones posteriores, una vez que el LLM fuera incorporado al proyecto.

Además, con vistas a una posible integración en otros entornos, se consideró conveniente que la longitud máxima permitida pudiera modificarse fácilmente.

### 3.3.2. Envío de Mensajes

Aunque en esta primera fase del proyecto se consideró únicamente el envío de *strings* simples desde ADARVE al *tutor-engine*, una funcionalidad clave para validar la viabilidad del sistema era permitir que el usuario pudiera formular preguntas directamente al tutor. Esta capacidad representa uno de los objetivos centrales del presente trabajo, al introducir una vía activa de interacción por parte del estudiante.

Dado que ADARVE no contaba previamente con ningún sistema similar, fue necesario diseñar e implementar esta funcionalidad desde cero. Para ello, se desarrolló un nuevo *widget* denominado `WBP_TutorialBuddyInputPanel`, que actúa como panel interactivo donde el usuario puede introducir texto mediante teclado (ver Figura 3.5). Al pulsar la tecla `Enter`, el contenido del campo se envía al motor lógico y el panel se vacía automáticamente, quedando listo para una nueva entrada.

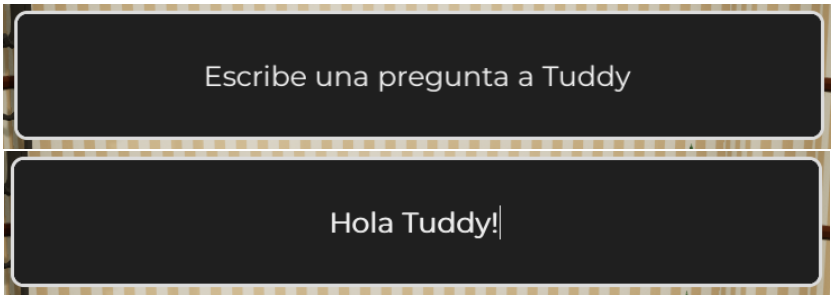


Figura 3.5: *WBP_ TutorialBuddyInputPanel* con y sin texto del usuario

## 3.4. Estado del Proyecto: Primer Prototipo

La comunicación efectiva entre ADARVE y el *tutor-engine* en ambos sentidos, junto con la implementación de una interfaz funcional para la interacción con el tutor, permite dar por cumplidos los objetivos planteados para esta primera iteración del proyecto.

El sistema desarrollado sugiere que es técnicamente viable integrar un modelo conversacional inteligente dentro del entorno de realidad virtual ADARVE mediante los dos módulos propuestos, cuyo estado en esta etapa queda reflejado en la Figura 3.6.

Durante el desarrollo también surgieron ciertas limitaciones que deberán abordarse en futuras fases, como la gestión de la longitud de los mensajes en pantalla, que puede afectar negativamente a la legibilidad de las respuestas del tutor.

En conjunto, este primer prototipo ha servido para comprobar que es posible llevar a cabo esta integración sin alterar la estructura del entorno original. Además, ha dejado una base técnica sobre la que seguir trabajando y mejorando el sistema en próximas versiones.

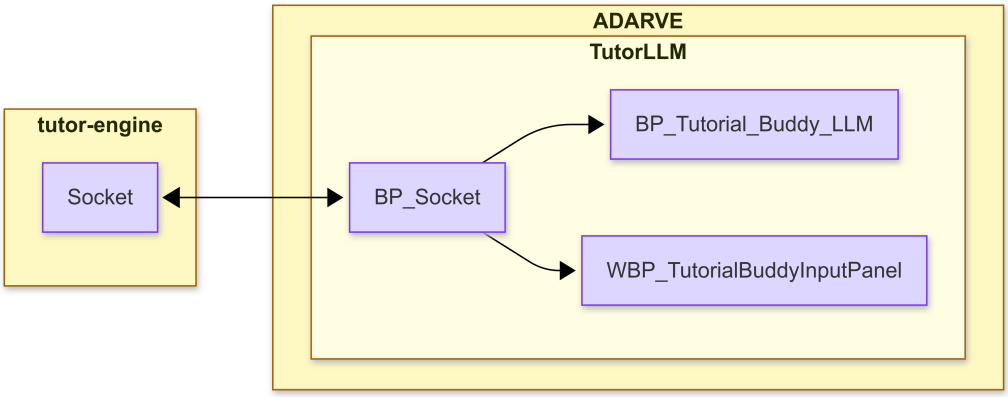


Figura 3.6: Diseño del prototipo inicial

Capítulo 4

# Desarrollo del Motor de Diálogos

Una vez completada la fase de validación técnica del proyecto y confirmada la viabilidad de integrar un tutor inteligente externo en el entorno de ADARVE, el siguiente paso fue el desarrollo de una versión funcional del sistema de diálogos. En este capítulo se detalla el trabajo realizado durante esta etapa, centrado en la construcción de un prototipo que, aunque todavía limitado, fue capaz de ofrecer respuestas útiles y coherentes dentro del contexto educativo.

## 4.1. Integración del LLM

En esta sección se describe el proceso seguido para seleccionar un LLM, tanto la plataforma más adecuada como el modelo concreto utilizado. Además, se detalla cómo se llevó a cabo su integración técnica con el sistema, incluyendo aspectos como la autenticación mediante clave de acceso.

### 4.1.1. Elección de la Plataforma

Entre las distintas opciones disponibles, se optó por utilizar los modelos GPT, ofrecidos por OpenAI, frente a otras alternativas como las mencionadas en la Sección 2.3.3.

Una de las principales razones para descartar la ejecución de un modelo local fue la limitación de recursos. Los modelos de código abierto que ofrecen un rendimiento comparable requieren una gran cantidad de espacio en disco, así como una carga computacional muy elevada. Teniendo en cuenta que ADARVE ya consume una parte significativa de los recursos del sistema, y que el servidor del tutor debe ejecutarse de forma paralela en el mismo equipo, utilizar un modelo local habría hecho inviable la solución desde el punto de vista del rendimiento y del coste computacional.

También se valoraron otras alternativas en la nube, como Gemini, desarrollado por Google. Sin embargo, la experiencia previa de otros proyectos, puso de manifiesto ciertas limitaciones. Como se detalló en la Sección 2.3.3, el acceso a la API de Gemini desde territorio europeo estuvo restringido por cuestiones legales relacionadas con la normativa de protección de datos, lo que obligó a utilizar una VPN para poder continuar con las pruebas. Aunque actualmente este tipo de restricciones ya no están

activas, esta experiencia generó cierta desconfianza que, sumada a la necesidad de una solución estable y sencilla, llevó a descartar esta opción.

En contraste, el principal motivo para elegir los modelos de OpenAI fue precisamente su estabilidad. Al tratarse de una de las primeras plataformas en lanzar herramientas de este tipo al público general, cuenta con una infraestructura consolidada y optimizada. En el momento de la elección, no se encontraron registros de interrupciones de servicio en Europa comparables a las experimentadas con Gemini.

### 4.1.2. Elección del Modelo

Aunque se optó por utilizar OpenAI como plataforma, la elección no termina ahí, pues esta ofrece varios modelos con distintos niveles de capacidad, coste y velocidad de respuesta. Dado que se trata de un servicio de pago, se decidió comenzar probando las opciones más económicas, `gpt-4o-mini` y `gpt-3.5-turbo`.

Para comprobar su rendimiento, se realizaron una serie de pruebas en las que se le formulaban preguntas al modelo utilizando el contexto generado por el tutor. El objetivo era evaluar la calidad y pertinencia de las respuestas.

A partir de los resultados obtenidos, se concluyó que el rendimiento del modelo `gpt-4o-mini` era superior al de `gpt-3.5-turbo`. Los ejemplos mostrados en la siguiente tabla corresponden a solo tres de las más de quince preguntas formuladas durante las pruebas realizadas.

| **Pregunta** | **gpt-4o-mini** | **gpt-3.5-turbo** |
|---|---|---|
| Hola Tuddy! ¿Dónde está el medidor? | Hola! El medidor de radiación está en la habitación, pero necesitarás salir de aquí para encontrarlo. Así que lo primero que tienes que hacer es salir de la habitación. ¿Puedes hacerlo? | Hola! El medidor se encuentra cerca de ti, sigue recto y lo encontrarás a tu derecha. ¡Buena suerte! |
| ¿Qué tareas tengo que hacer? | Tu tarea en este nivel llamado "Primeros pasos" es salir de la habitación. Este es el primer paso que debes completar. ¿Necesitas ayuda para encontrar la salida? | Para completar el nivel "Primeros pasos", las tareas que debes realizar en orden son las siguientes: 1. Sal de la habitación. Si necesitas más ayuda o tienes alguna pregunta, no dudes en decírmelo. ¡Estoy aquí para ayudarte en tu entrenamiento! |

| **Pregunta** | **gpt-4o-mini** | **gpt-3.5-turbo** |
|---|---|---|
| ¿Dónde estoy? | Actualmente te encuentras en una habitación dentro del entorno de entrenamiento de realidad virtual. Desde tu posición, puedes intentar salir de la habitación para avanzar en tu nivel. ¿Te gustaría que te guíe sobre cómo hacerlo? | ¡Hola! Te encuentras en el nivel "Primeros pasos"de tu entrenamiento en realidad virtual. Actualmente, estás en una posición donde puedes ver una serie de elementos y objetos a tu alrededor que te permitirán interactuar con el entorno. ¿En qué puedo ayudarte hoy en tu entrenamiento? |

Tabla 4.1: Comparación de respuestas entre `gpt-4o-mini` y `gpt-3.5-turbo`

En la primera pregunta, por ejemplo, el modelo mini fue capaz de deducir que el usuario se encontraba dentro de una habitación, mientras que `3.5-turbo` ofreció una indicación imprecisa sobre la localización del medidor, guiando hacia una dirección incorrecta.

En la segunda, `gpt-4o-mini` respondió de forma más natural, contextualizada y adaptada al diálogo, mientras que la respuesta de `gpt-3.5-turbo` reproducía literalmente una parte del contenido incluido en el *prompt*.

Finalmente, en la tercera pregunta, ambos modelos identificaron correctamente que el usuario se encontraba en un entorno de realidad virtual. No obstante, mientras que la respuesta de `gpt-3.5-turbo` volvía a incluir expresiones textuales del *prompt*, `gpt-4o-mini` ofrecía una respuesta más orientada a la acción y centrada en el objetivo educativo del entorno, lo que encaja mejor con el rol del tutor inteligente.

Debido a su mejor rendimiento y a un coste asequible, se decidió utilizar la versión `gpt-4o-mini` como modelo de lenguaje para este prototipo.

### 4.1.3. Llamadas a la API de GPT

Para realizar consultas al modelo GPT, se emplea la clase `OpenAI`, incluida dentro de la librería oficial `openai` para Python. Esta clase requiere como parámetro de inicialización una clave de acceso válida, comúnmente conocida como API Key.

El proceso de obtención de la clave se llevó a cabo directamente desde la página oficial de OpenAI. Tras registrarse en la plataforma, es posible generar una API Key desde el panel de usuario, configurando tanto el plan de uso como los permisos deseados. Para proteger la seguridad de esta clave, se definió como una variable de entorno en el sistema operativo (`OPENAI_API_KEY`), evitando así exponerla directamente en el código fuente.

Con el objetivo de simplificar las llamadas al modelo, se definió una función auxiliar denominada `ask`. Esta función, como se muestra a continuación, recibe como parámetros el contexto y la pregunta del usuario y realiza una consulta al modelo utilizando el método `create` de la API.

Listing 4.1: LLamadas a la API de GPT

```python
import os
from openai import OpenAI
client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

def ask(context, question):
    completion = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {
                "role": "system",
                "content": context},
            {
                "role": "user",
                "content": question
            }
        ]
    )
    return completion.choices[0].message.content
```

Además, esta abstracción permite mantener el código del tutor desacoplado del proveedor concreto del modelo de lenguaje. En caso de que en el futuro se decida utilizar otra API o integrar un modelo diferente, bastará con modificar la función `ask` sin necesidad de alterar el resto del sistema.

## 4.2. Sistema de Mensajes

Un aspecto clave para generar respuestas de calidad por parte del *tutor-engine* es proporcionarle información suficiente sobre el entorno de realidad virtual y sobre el propio estudiante.

Esto incluye, por ejemplo, conocer en qué nivel se encuentra el estudiante, qué tareas ha completado o qué diálogos ha recibido previamente. Sin estos datos, el tutor solo puede ofrecer respuestas genéricas que no guarden relación con ADARVE ni con la sesión de aprendizaje, puesto que los desconoce.

Para ello, fue necesario mejorar las comunicaciones entre ambos módulos mediante la incorporación de un sistema de mensajes, que permite el envío estructurado de distintos tipos de información desde ADARVE hacia el tutor.

### 4.2.1. Tipos de Mensajes Definidos

Con este objetivo, se definieron tres tipos de mensajes que ADARVE puede enviar al tutor:

1. **Mensaje de inicialización**. Se trata de un mensaje único que se envía en el momento en que se establece la conexión entre ambos sistemas. Contiene información estática del nivel actual, como el nombre del nivel, su descripción y los objetos presentes en el mapa.

2. **Mensaje de sincronización**. Este tipo de mensaje se envía de forma periódica e incluye datos sobre el estado del entorno y del estudiante, los cuales varían dinámicamente a lo largo de la sesión de aprendizaje y, por tanto, deben actualizarse. Entre los datos que transmiten se encuentran las tareas completadas, la información que el estudiante ha conocido mediante diálogos con NPCs y su posición actual.

   Lo ideal sería que el tutor recibiera notificaciones inmediatas después de algún cambio en estos datos. Sin embargo, el entorno ADARVE no está diseñado para emitir estas notificaciones, por lo que resulta necesario observar el estado de la sesión periódicamente.

   Se ha optado por una frecuencia de 5 segundos por ser lo suficientemente espaciado como para no saturar la comunicación, pero lo bastante frecuente como para reflejar la evolución del estudiante.

   En la práctica, todos los cambios que se producen hasta el envío del mensaje son interpretados por el tutor como simultáneos, por lo que a menor frecuencia de sincronización, menor granularidad sobre el progreso del alumno.

3. **Mensaje de pregunta**. Se genera cada vez que el usuario decide interactuar con el tutor para realizar una consulta. Este mensaje encapsula tanto el contenido de la pregunta como un mensaje de sincronización, con el fin de asegurar que el tutor dispone del contexto más reciente a la hora de generar una respuesta.

### 4.2.2. Formato de los Mensajes Enviados desde ADARVE

Uno de los primeros pasos para implementar un sistema de mensajes consiste en la elección de un formato adecuado, en este caso, para los mensajes generados desde ADARVE. Desde el inicio se contempló el uso de JSON, al tratarse de un estándar ampliamente utilizado para la transmisión de datos y que, en general, se puede usar fácilmente en prácticamente cualquier entorno de desarrollo.

Además, como ya se mencionó en la Sección 3.2.1, existe un *plugin* que permite la creación de objetos en formato JSON dentro de Unreal Engine, lo que reforzó la decisión de adoptar este formato para la construcción de los mensajes.

Para implementar esta funcionalidad, se desarrolló un nuevo actor denominado `BP_MessageBuilder`, encargado de generar los mensajes JSON y de recopilar en el entorno toda la información necesaria para conformar su contenido.

A continuación se detalla el formato de los tres tipos de mensajes enviados desde ADARVE, definidos en la sección anterior. Para cada uno de ellos se incluye, en primer lugar, una descripción de su estructura y contenido, seguida de un ejemplo representativo expresado en formato JSON.

#### 4.2.2.1. Mensaje de inicialización

Este mensaje inicial encapsula los pasos que el usuario debe seguir, la descripción del escenario, la ubicación inicial de cada elemento, la posición inicial del jugador y las instrucciones tutoriales, además de una marca temporal.

También incluye los nombres de todos los personajes presentes en el nivel junto con sus diálogos, ya que ADARVE almacena internamente el índice de la última frase mostrada por cada NPC. Por ello, resulta útil que el tutor disponga de todos los diálogos desde el inicio y reciba las actualizaciones de dicho índice a través de mensajes de sincronización.

Asimismo, se extraen todas las instrucciones correspondientes al primer nivel tutorial, ya que este explica cómo interactuar con el entorno 3D y con los objetos fundamentales utilizados en las sesiones de aprendizaje, como los dosímetros. Contar con esta información permite al tutor ofrecer asistencia si el estudiante presenta dudas sobre el funcionamiento de ADARVE.

Toda esta información se almacena en los siguientes campos de un JSON:

- **type**: Campo de tipo cadena. En este caso, su valor es *"init"*.
- **data**: Objeto principal que contiene todos los datos relevantes del entorno virtual. Se subdivide en varios elementos:
  - **steps**: Lista de pasos que el usuario debe completar durante el nivel. Cada paso es un objeto con:
    - `id`: Identificador único del paso.
    - `description`: Descripción textual de la acción a realizar.
  - **level**: Contiene la información general del nivel:
    - `name`: Nombre del nivel o tutorial.
    - `description`: Breve explicación de los objetivos del nivel.
  - **npc_dialogs**: Lista de personajes con los que el usuario puede interactuar. Cada NPC tiene:
    - `name`: Nombre del actor en el sistema.
    - `dialog`: Lista con los bloques de diálogo que el personaje puede decir.
  - **elements_location**: Lista de todos los elementos y objetos presentes en la escena virtual, junto con su posición tridimensional. Cada elemento se define por:
    - `name`: Nombre del objeto.
    - `position`: JSON que indica la ubicación en coordenadas X, Y y Z.
  - **player**: Información sobre la posición y orientación inicial del jugador en el entorno:
    - `position`: JSON con las coordenadas X, Y, Z.
    - `orientation`: Dirección en la que está mirando el jugador, en formato X, Y, Z.
  - **tutorial**: Conjunto de instrucciones didácticas que guían al usuario en el uso del entorno. Cada elemento incluye:
    - `title`: Título del concepto, funcionalidad u objeto.
    - `description`: Explicación de cómo funciona o cómo debe utilizarse.

- **timestamp**: Marca de tiempo que indica el momento exacto en el que se genera o registra el JSON, con formato `"YYYY/MM/DD_HH:MM:SS"`.

Listing 4.2: Ejemplo de mensaje de inicialización

```
{
  "type": "init",
  "data": {
    "steps": [
      {
        "id": "S1001",
        "description": "Sal de la habitacion"
      },
      {
        "id": "S1002",
        "description": "Habla con ..."
      }
    ],
    "level": {
      "name": "Primeros pasos",
      "description": "Aprenderas las funcionalidades..."
    },
    "npc_dialogs": [
      {
        "name": "BP_NPC_S1_Talk_C_1",
        "dialog": [
          "Madre mia, Quien ha desordenado todo esto?",
          "Tienes que recoger algunos objetos."
        ]
      }
    ],
    "elements_location": [
      {
        "name": "SM_wall_9",
        "position": {
          "X": 12510.0,
          "Y": -3300.0,
          "Z": 0
        }
      },
      {
        "name": "PlayerStart_1",
        "position": {
          "X": 12397.05,
          "Y": -3040,
          "Z": 100
        }
      }
    ],
    "player": {
```

```
      "position": {
        "X": 12397.05,
        "Y": -3040,
        "Z": 98.27
      },
      "orientation": {
        "X": 1,
        "Y": 0,
        "Z": 0
      }
    },
    "tutorial": [
      {
        "title": "Movimiento con teletransporte",
        "description": "Usa el joystick para..."
      },
      {
        "title": "Coger y soltar",
        "description": "Acercate a un objeto y..."
      }
    ]
  },
  "timestamp": "2025/4/19_18:10:15.100"
}
```


#### 4.2.2.2. Mensaje de sincronización

El mensaje de sincronización refleja el estado del entorno virtual en un momento específico de la sesión. Contiene información sobre el progreso del jugador, los diálogos activos con los demás personajes y la posición y orientación actual del usuario en el nivel. Sigue el siguiente formato:

- **type**: Campo de tipo cadena con valor *"sync"*.
- **data**: Objeto que agrupa toda la información sincronizada en el momento.
  - **progress**: Lista de identificadores de pasos ya completados por el jugador.
  - **npc_dialogs**: Lista de npcs con su estado. Cada objeto incluye:
    - `name`: Identificador del personaje.
    - `conversation_index`: Índice de la conversación activa.
    - `dialog_index`: Diálogo actual dentro de la conversación.
    - `is_talking`: Booleano que indica si el personaje está hablando actualmente.
  - **player**: Describe el estado actual del jugador.
    - `position`: Coordenadas espaciales X, Y y Z del jugador.
    - `orientation`: Orientación del jugador en formato X, Y, Z.

- **timestamp**: Marca de tiempo que indica cuándo se capturó la información, con el formato `"YYYY/MM/DD_HH:MM:SS.mmm"`.

Listing 4.3: Ejemplo de mensaje de sincronización

```
{
  "type": "sync",
  "data": {
    "progress": ["S1001"],
    "npc_dialogs": [
      {
        "name": "BP_NPC_S1_Talk_C_1",
        "conversation_index": 0,
        "dialog_index": 0,
        "is_talking": false
      }
    ],
    "player": {
      "position": {
        "X": 12654.271484375,
        "Y": -3908.970703125,
        "Z": 98.2750015258789
      },
      "orientation": {
        "X": 0.9779590368270874,
        "Y": 0.2087969034910202,
        "Z": 0
      }
    }
  },
  "timestamp": "2025/4/19_18:36:36.251"
}
```

#### 4.2.2.3. Mensaje de pregunta

Este tipo de mensajes son una extensión de los de sincronización, incorporando además la consulta formulada por el estudiante:

- **type**: Cadena con el valor *"question"*.
- **text**: Contenido textual de la consulta.
- **sync**: JSON que encapsula el estado del entorno al momento de la interacción. Tiene el mismo formato que un mensaje de sincronización.
- **timestamp**: Marca de tiempo en el mismo formato que en los otros dos tipos mensajes.

Listing 4.4: Ejemplo de mensaje JSON de pregunta al asistente virtual


```
{
  "type": "question",
  "text": "hola tuddy!",
  "sync": {
    "type": "sync",
    "data": {
      "progress": [],
      "npc_dialogs": [
        {
          "name": "BP_NPC_S1_Talk_C_1",
          "conversation_index": 0,
          "dialog_index": 0,
          "is_talking": false
        }
      ],
      "player": {
        "position": {
          "X": 12397.056640625,
          "Y": -3040,
          "Z": 98.2750015258789
        },
        "orientation": {
          "X": 0.9984694719314575,
          "Y": 0.05530586466193199,
          "Z": 0
        }
      }
    },
    "timestamp": "2025/4/19_18:45:47.872"
  },
  "timestamp": "2025/4/19_18:45:47.872"
}
```


### 4.2.3. Generación del Contenido de los Mensajes

Como ya se ha adelantado, el responsable de generar el contenido de los mensajes es el actor `BP_MessageBuilder`. Aunque en apariencia pueda parecer una tarea sencilla, su implementación ha resultado especialmente compleja. Esto se debe a que ADARVE gestiona gran parte de su información a través de objetos internos a los que no es posible acceder fácilmente sin modificar directamente su código, lo cual se buscaba evitar.

El proceso ha requerido un intenso trabajo de investigación, basado en la exploración del entorno, técnicas de prueba y error, y observación directa, ya que no existe documentación oficial sobre ADARVE. En este contexto, la colaboración con su desarrollador fue clave para comprender el funcionamiento interno del sistema.

Gracias a este esfuerzo, se consiguió acceder de forma externa a los datos reque-

ridos para la construcción de los mensajes, permitiendo así al *tutor-engine* trabajar con información contextual y actualizada.

#### 4.2.3.1. Estructura de `BP_MessageBuilder`

El actor `BP_MessageBuilder` fue diseñado expresamente siguiendo el patrón de diseño *Builder* (Gamma et al., 1994), que permite componer mensajes estructurados de forma flexible, facilitando tanto su mantenimiento como su adaptación a futuras necesidades.

Su estructura interna se basa en dos tipos principales de funciones: por un lado, aquellas que comienzan con el prefijo `Build`, responsables de crear un objeto JSON desde cero; y por otro, las funciones que empiezan por `Add`, destinadas a incorporar elementos concretos a un mensaje ya existente. Todas ellas se describen en la siguiente tabla:

| **Función** | **Descripción** |
|---|---|
| `BuildInitMessage()` | Construye el mensaje de inicialización con los datos estáticos del nivel. |
| `BuildSyncMessage()` | Genera un mensaje de sincronización con información dinámica del entorno. |
| `BuildQuestionMessage()` | Crea un mensaje de pregunta que incluye la consulta del usuario y el contexto necesario. |
| `AddTimestamp()` | Añade al mensaje la marca temporal actual. |
| `AddSyncData()` | Incorpora los datos de sincronización generales del entorno. |
| `AddStepsData()` | Añade información sobre los pasos para completar el nivel. |
| `AddDescriptionLevelData()` | Incluye la descripción del nivel actual. |
| `AddProgressData()` | Añade el estado de progreso del usuario en el entorno. |
| `AddDialogData()` | Añade todos los diálogos de los personajes del nivel. |
| `AddCurrentConversationData()` | Indica los diálogos ya conocidos por el estudiante. |
| `AddElementsLocationData()` | Incorpora las posiciones de los objetos del entorno. |
| `AddPlayerLocationData()` | Incluye la posición actual del jugador. |
| `AddTutorialData()` | Añade información relacionada extraída del tutorial sobre el entorno de realidad virtual. |
| `CoordsToJSON()` | Función de utilidad para convertir coordenadas del entorno a formato JSON. |

Tabla 4.2: Funciones de `BP_MessageBuilder` organizadas según el patrón *Builder*

#### 4.2.3.2. Problemas con la base de datos de ADARVE

Como se mencionó en la Sección 2.6.3, ADARVE cuenta con un sistema de almacenamiento basado en ficheros `.csv`, que registra automáticamente parte de la actividad del entorno. Sin embargo, desde el momento en que se otorgó acceso al proyecto, este módulo presentaba un error de compilación que impedía su uso, tal y como se muestra en la Figura 4.1.

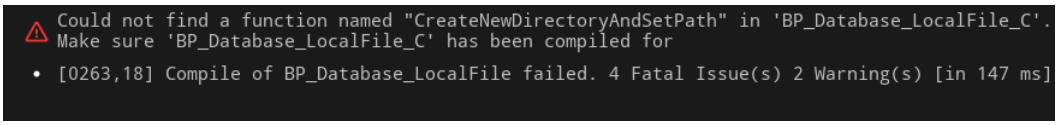


Figura 4.1: Error de compilación en la base de datos de ADARVE

Este componente forma parte del núcleo de ADARVE y queda fuera del ámbito de este trabajo, por lo que no se consideró viable modificar directamente su código para intentar solucionarlo. De hecho, esa es una de las premisas sobre las que se fundamenta este proyecto, trabajar de manera externa.

En su momento se contactó con el desarrollador principal del entorno, pero no se logró identificar la causa del error.

Además, aunque ADARVE incluye una funcionalidad para generar los archivos `.csv`, no proporciona ninguna herramienta integrada para abrirlos desde el propio entorno. Para poder acceder a esa información habría sido necesario desarrollar una funcionalidad adicional en *C++*, lo cual implicaba usar una versión de Visual Studio para la que no se tenía licencia.

Debido a estas limitaciones, se optó por obtener directamente los datos desde ADARVE en tiempo de ejecución, en lugar de depender del sistema de almacenamiento interno.

Más adelante, una vez completado este prototipo y con el desarrollo de la versión final en marcha, se descubrió que el problema estaba causado por un bug de Unreal Engine. Al parecer, se había generado de forma incorrecta una clase temporal (`SKEL`) utilizada internamente para compilar el código en C++. En este caso, dicha clase era la responsable de gestionar el módulo de base de datos.

La solución consistió en borrar las carpetas `Intermediate` y `Binaries` del proyecto, lo que forzó a Unreal Engine a regenerar todos los archivos temporales y permitió compilar nuevamente el proyecto sin errores.

### 4.2.4. Procesamiento de los Mensajes en el Tutor

Para gestionar los mensajes recibidos en el *tutor-engine*, se ha desarrollado un módulo en Python denominado `message`, que contiene una clase específica para representar y almacenar la información correspondiente a cada tipo de mensaje.

Cuando el servidor recibe un mensaje, este se clasifica automáticamente en función del campo `"type"` y se procesa según su finalidad:

- **Mensaje de inicialización**: se almacena toda la información estática de la partida en una instancia de la clase `Knowledge`, que representa el conocimiento

del tutor sobre el entorno.

- **Mensaje de sincronización**: se actualizan los atributos de la clase `Student` de acuerdo con el progreso del estudiante dentro del nivel.
- **Mensaje de pregunta**: como incluye internamente un mensaje de sincronización, primero se procesa dicha parte del mensaje de forma equivalente a cualquier otro mensaje de ese tipo. A continuación, se extrae el contenido de la pregunta, que se transfiere a la instancia de la clase `Tutor`, la cual se encarga de realizar la consulta al LLM. Una vez generada la respuesta, esta se envía de vuelta a ADARVE a través del *socket*.

En la siguiente sección se proporciona información más detallada sobre las clases `Knowledge`, `Student` y `Tutor`.

## 4.3. Diseño del Sistema de Tutoría

Para incorporar la nueva funcionalidad de responder a preguntas, fue necesario establecer un sistema de tutoría capaz de procesar las consultas del usuario y generar respuestas fundamentadas en datos relevantes.

Se decidió adaptar los modelos clásicos de tutoría, descritos en la Sección 2.5.1, integrando un LLM como núcleo encargado de la generación de respuestas, tal y como se muestra en la Figura 4.2.

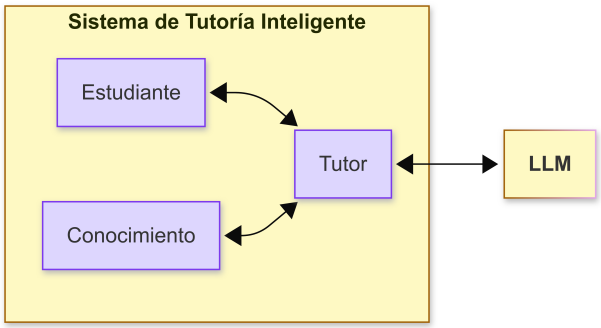


Figura 4.2: Sistema de tutoría inteligente en *tutor-engine*

De este modo, el sistema de tutoría se estructura en torno a cuatro componentes fundamentales: el modelo del conocimiento, el modelo del estudiante, el modelo del tutor y la interfaz con el usuario. Esta última está representada por ADARVE, cuyas mejoras específicas en esta versión se desarrollan en la Sección 4.6. En esta sección nos centraremos en los componentes desarrollados dentro del *tutor-engine*.

### 4.3.1. Conocimiento

El módulo del conocimiento es el encargado de almacenar toda la información estática del nivel. Estos datos se utilizan para proporcionar contexto al LLM sobre el entorno de realidad virtual y el nivel actual.

En un primer momento se valoró la posibilidad de utilizar una base de datos SQL. Sin embargo, dado que no existen relaciones complejas entre los datos (no es necesario realizar operaciones como JOIN), su principal aportación sería la persistencia. Esta característica pierde relevancia en el contexto actual, ya que ADARVE sigue en desarrollo activo y la información del entorno puede cambiar con frecuencia. Esto implicaría tener que mantener la coherencia entre los cambios en el entorno 3D y el contenido almacenado en la base de datos, lo cual complicaría innecesariamente el sistema.

También se consideró el uso de una base de datos NoSQL, dado que la información no es relacional y este tipo de sistemas ofrecen mayor flexibilidad frente a cambios en la estructura de los datos. No obstante, la cantidad de información gestionada es lo suficientemente pequeña como para poder ser almacenada de forma eficiente sin necesidad de incorporar una base de datos externa y, de nuevo, evitar su mantenimiento.

Por estos motivos, se optó por una solución más sencilla que consistió en crear una clase llamada `Knowledge`, cuyos atributos permiten almacenar y acceder directamente a la información necesaria durante la generación de respuestas por parte del tutor.

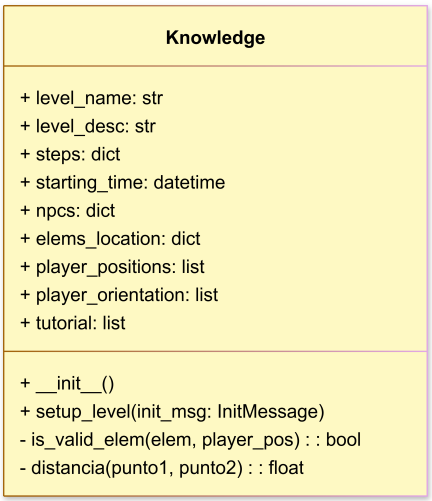


Figura 4.3: Diagrama UML de `Knowledge`

Dado que esta clase replica la estructura del mensaje de inicialización, sus atributos y métodos, que se pueden ver en la Figura 4.3, son en su mayoría autoexplicativos. Su funcionamiento ha sido descrito de forma indirecta en los apartados 4.2.2.1 y 4.2.4, donde se detalla el formato y cómo se procesa dicha información en el *tutor-engine*. No obstante, resulta pertinente profundizar en el procesamiento que se realiza sobre los elementos presentes en un nivel de ADARVE.

Cada nivel contiene una gran cantidad de objetos, muchos de los cuales no son relevantes para el estudiante, como gestores internos, controladores, materiales, entre otros. Por este motivo, se lleva a cabo un proceso de filtrado que permite conservar únicamente aquellos elementos considerados relevantes, según dos criterios principales:

- **Distancia al estudiante**: los elementos que se encuentren a una distancia mayor que un umbral configurable $d$ con respecto al estudiante no serán procesados. Este valor se puede ajustar mediante el archivo de configuración del tutor.
- **Nombre del elemento**: se analizaron los objetos existentes en ADARVE y se identificaron patrones comunes en los nombres de aquellos que no resultan significativos. Por ejemplo, si el nombre contiene términos como *render*, *child* o *volume*, es muy probable que se trate de elementos internos del entorno, y por tanto no se almacenan. La lista de palabras clave utilizadas para este filtrado también se establece mediante el archivo de configuración, donde pueden añadirse o modificarse los subcadenas no permitidas.

### 4.3.2. Estudiante

El módulo del estudiante se encarga de almacenar el progreso del usuario dentro del entorno virtual, representando así el conocimiento dinámico y los avances que este ha logrado a lo largo de la experiencia de aprendizaje.

Dado que se trata de información que cambia constantemente y que es única para cada sesión de entrenamiento, desde el inicio se planteó su implementación como una clase, denominada `Student` (ver Figura 4.4), cuya responsabilidad es actualizar sus atributos conforme el estudiante progresa en el entorno.

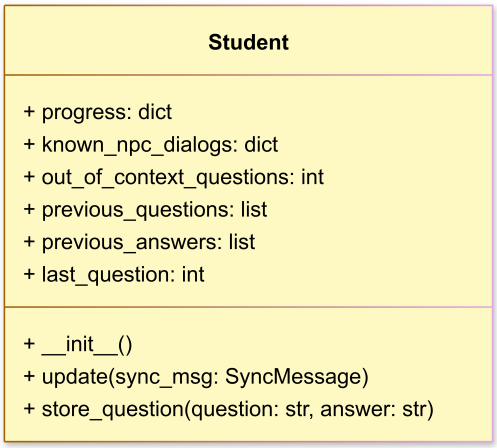


Figura 4.4: Diagrama UML de `Student`

La información que almacena la clase `Student` en esta versión incluye:

- **Tareas completadas.** A través del atributo `progress`, se registra qué tareas del nivel han sido finalizadas, así como el momento en que fueron realizadas.
- **Diálogos conocidos.** El atributo `known_npc_dialogs` almacena, para cada NPC presente en el nivel, los identificadores de los diálogos que el estudiante ya conoce. De este modo, el tutor puede conocer que información le ha sido proporcionada al alumno desde ADARVE y hacer referencia a ella en sus respuestas.

- **Consultas previas.** El historial de interacción entre el tutor y el estudiante se conserva mediante los atributos `previous_questions` y `previous_answers`, que recogen, respectivamente, las preguntas formuladas por el estudiante y las respuestas proporcionadas por el tutor. Esta información permite dar continuidad a las conversaciones previas.

- **Preguntas fuera de contexto.** Cantidad de preguntas del estudiante que fueron clasificadas como fuera de contexto (véase la Sección 4.3.3.1 para más detalles).

### 4.3.3. Tutor

El módulo del tutor es el encargado de recibir las preguntas del estudiante y generar un *prompt* adecuado para el LLM, consultando para ello los datos almacenados en las clases `Knowledge` y `Student`.

La clase `Tutor` puede entenderse como un generador de contexto para el modelo de lenguaje. Para ello, el método `generate_context()` utiliza una serie de mensajes predefinidos que actúan como conectores para la información procedente de los otros dos módulos. En esta versión, se incluyen todos los datos disponibles en `Knowledge` y `Student` con el objetivo de enriquecer al máximo el contexto de la consulta.

Para el diseño de este *prompt* se tuvo en cuenta el principio CLEAR definido en la Sección 2.3.4, destacando una estructura lógica bien definida. El contenido se organiza comenzando por la funcionalidad del tutor, seguido del contexto general del entorno de realidad virtual y avanzando progresivamente desde la información específica del nivel hasta los datos relacionados con el estudiante y sus interacciones.

A continuación, se muestra un ejemplo del contexto generado por la función en una partida en la que el usuario no ha completado ninguna tarea, pero ya ha realizado una consulta previa al tutor:

```
Eres Tuddy, el tutor de un agente de seguridad que esta entrenando en un entorno de
 realidad virtual para tratar situaciones de emergencias radiologicas cotidianas.
Aqui tienes un pequeno tutorial e indicaciones sobre como funciona el entorno:

 - Movimiento en la vida real: Si te mueves en el mundo real....
----------omitidos el resto de indicaciones----------

Actualmente se encuentra en un nivel llamado: "Primeros pasos" cuya descripcion es
: "Aprenderas las funcionalidades basicas de la realidad virtual que te permitiran
interactuar con el entorno.." El estudiante debe realizar EN ORDEN los siguientes
pasos para completar el nivel:

1. Sal de la habitacion
----------omitidos el resto de pasos----------

De los cuales ha completado: Ninguno

El mapa tiene una serie elementos situados en las situados en las siguientes
posiciones:

- BP_Dosimeter_SABG100_C_3 en la posicion [12693.74609375, -2985.272216796875,
47.42986297607422]

- BP_TrafficCone_C_0 en la posicion [12275.15625, -3766.976318359375,
1.6051748161771684e-06]
----------omitidos el resto de elementos----------
```

```
Ten en cuenta que los nombres de estos objetos son usados internamente por el
entorno. Nunca uses estos nombres, has de adaptarlos al lenguaje natural,
Estas son las ultimas preguntas que realizo el estudiante y las respuestas que
recibio:
- Pregunta: Hola Tuddy!
  Respuesta: En que puedo ayudarte hoy en tu entrenamiento?

El usuario ha mantenido las siguientes conversaciones con elementos del entorno:
- Personaje BP_NPC_S1_Talk_C_1: Ninguna

El usuario se encuentra en la siguiente posicion:{'X': 12397.056640625, 'Y': -3040,
 'Z': 98.2750015258789}.
El usuario esta mirando en la siguiente direccion:[1, 0, 0].

En caso de que tengas que decirle al usuario donde esta un objeto, debes considerar
 que no entiende de coordenadas. Nunca debes mostrarle la posicion en coordenadas
de un objeto al usuario, sino darle indicaciones del estilo: Sigue recto, esta
cerca de, gira a la derecha, sal de la habitacion,...
```

#### 4.3.3.1. Filtrado del contenido y generación de respuestas

La clase `Tutor` está directamente conectada con el LLM (ver Figura 4.5), de forma que las preguntas del usuario se le envían junto con el contexto extraído del entorno y del progreso del estudiante. Esta estrategia introduce el riesgo de que el usuario realice preguntas que no guarden relación con el entorno educativo, lo que puede provocar respuestas fuera de lugar.

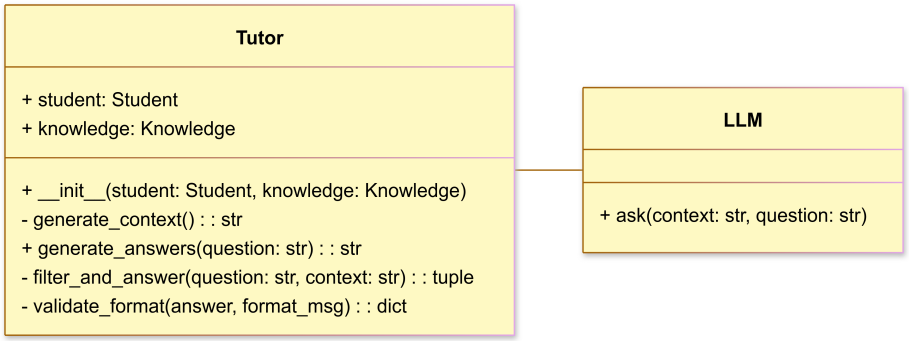


Figura 4.5: Diagrama de clases de `Tutor` y `LLM`

Para evitar esta situación, se incorporó un sistema de filtrado que limita las respuestas únicamente a aquellas preguntas que estén relacionadas con el aprendizaje sobre emergencias radiológicas, el cual se describe a continuación.

En una primera versión, se planteó realizar dos llamadas independientes al LLM. En la primera, se evaluaba si la pregunta estaba relacionada con el entorno. Solo en caso afirmativo, se realizaba una segunda consulta para generar la respuesta. Sin embargo, este enfoque resultó poco práctico. El tiempo de respuesta total para preguntas válidas, considerando el procesamiento del modelo y los retrasos en la comunicación con ADARVE, se situaba en torno a 6 segundos por interacción. Esta latencia afectaba negativamente a la experiencia del usuario.

Por ello, se diseñó una nueva solución basada en una única llamada al LLM. En esta llamada se pedía al modelo que valorase en qué medida la pregunta estaba relacionada con el nivel en base a la información incluida en el *prompt* o con las

emergencias radiológicas, utilizando una escala del 0 al 10 (donde 0 indica que no guarda relación y 10 que está totalmente alineada). Además, se le pedía que generase dos tipos de respuesta. Una sería de tipo informativo, que explicase de forma amable que la pregunta está fuera de contexto. La otra, una respuesta válida para el caso en que la consulta resulte pertinente.

El formato de la respuesta esperada por parte del LLM sigue la estructura siguiente:

```
{
"filtro" = numero,
"respuestas_filtro" = [respuesta1, respuesta2,..., respuestan],
"respuestas_mensaje" = [respuesta1, respuesta2,..., respuestan]
}
```

El valor de `filtro` se compara con el parámetro `context_sensibility`, definido en el archivo de configuración del tutor. Si este número es inferior al umbral establecido, Tuddy utilizará las respuestas de filtro. En caso contrario, se presentará la respuesta correspondiente a la pregunta.

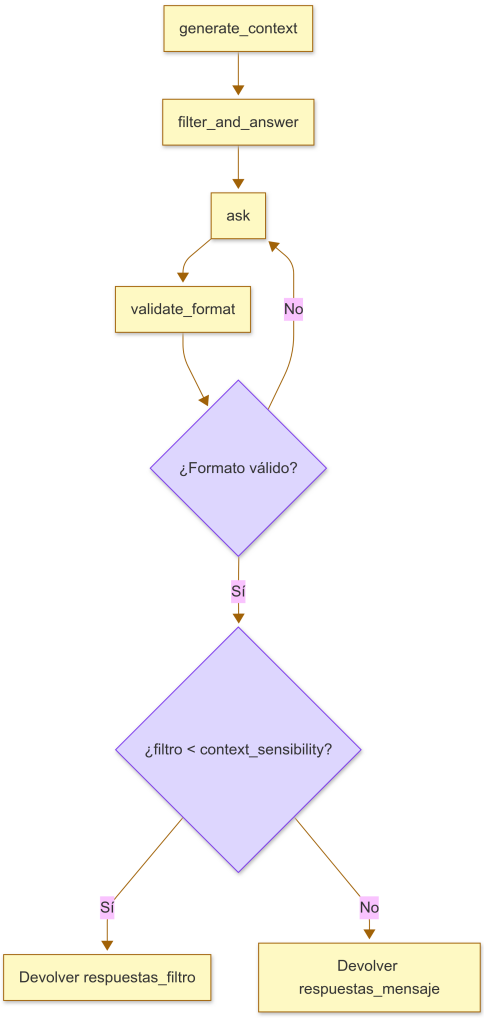


Figura 4.6: Flujo para generar una respuesta

Las respuestas del modelo se estructuran como una lista porque, como se explicó en la Sección 3.3.1.1, los cuadros de diálogo de Tuddy no admiten mensajes demasiado largos sin afectar a la visualización. Por ese motivo, se solicita al LLM que divida sus respuestas en mensajes de longitud reducida. El número máximo de caracteres por mensaje puede configurarse mediante el parámetro `max_char_per_msg`, también disponible en el archivo de configuración. Además, se establece un máximo de cinco mensajes por respuesta.

Para asegurarse de que el formato de la respuesta es correcto, se emplea la función `validate_format` para revisar la estructura recibida y vuelve a realizar la consulta al LLM si es necesario, hasta obtener una respuesta válida. La Figura 4.6 resume todo el proceso descrito en este apartado.

## 4.4. Mejoras en el Servidor del Tutor

Durante la segunda fase del proyecto, el servidor desarrollado en Python para gestionar las comunicaciones del *tutor-engine* fue sometido a una reestructuración. En versiones anteriores, el servidor funcionaba como un simple *socket* capaz de recibir mensajes, procesándolos de forma secuencial y sin capacidad para gestionar múltiples conexiones al mismo tiempo.

En esta nueva versión, se incorporó un sistema de concurrencia que mejora su escalabilidad. Cada vez que un cliente se conecta al servidor, se lanza un nuevo hilo de ejecución encargado exclusivamente de gestionar las comunicaciones con dicho cliente, como se aprecia en la siguiente figura:

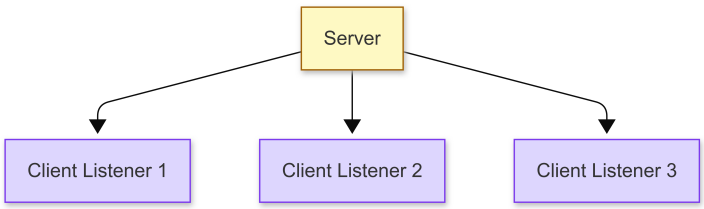


Figura 4.7: Diagrama del funcionamiento concurrente del servidor

La lógica de conexión y escucha de nuevos clientes se encuentra encapsulada en el archivo `server.py`. Este módulo permanece a la espera de nuevas conexiones entrantes y, al establecerse una conexión, deriva la responsabilidad al módulo `client_listener.py`. Este segundo componente es el encargado de gestionar la comunicación con cada cliente de forma individual, realizando las operaciones necesarias en función del tipo de mensaje recibido, según se indicó en la Sección 4.2.4.

Esta mejora agilizó el desarrollo del proyecto, ya que ante los cambios para realizar mejoras en el cliente, en este caso ADARVE, no era necesario reiniciar todo el servidor, sino únicamente el entorno de realidad virtual.

Además, los cambios permiten que múltiples usuarios puedan interactuar con el *tutor-engine* de manera simultánea sin que se bloqueen entre sí, algo que podría ser especialmente relevante en futuras versiones del sistema en las que se contemple el uso distribuido.

Aunque en esta versión todos los componentes se ejecutan en local y no se permite aún el acceso concurrente real desde distintas instancias del entorno, esta arquitectura deja preparado el sistema para futuros despliegues en red.

## 4.5. Archivo de Configuración

A lo largo de este capítulo se ha hecho referencia en varias ocasiones al archivo de configuración del tutor. Esta sección tiene como objetivo recopilar toda la información relacionada con dicho archivo, explicar su estructura y detallar el propósito de cada uno de sus parámetros.

El archivo de configuración, denominado `config.py`, es un script escrito en Python que contiene exclusivamente definiciones de variables. Gracias a esta estructura, el comportamiento del *tutor-engine* puede modificarse fácilmente sin necesidad de alterar el código principal del sistema. Esto facilita su mantenimiento y permite realizar ajustes rápidos adaptados a distintos contextos.

A continuación, se describen los campos disponibles:

- `enable_llm`. Permite activar o desactivar el uso del modelo de lenguaje. Es una variable pensada para ahorrar recursos durante el desarrollo.
- `max_char_per_msg`. Establece la longitud máxima permitida para cada mensaje generado por el LLM. Esta restricción está relacionada con las limitaciones del sistema de visualización de diálogos en ADARVE.
- `previous_questions`. Indica cuántas preguntas previas deben incluirse como parte del contexto de conversación. Si toma un valor negativo, se consideran todas.
- `context_sensibility`. Establece el nivel mínimo de relación que una pregunta debe tener con el contexto para ser respondida por el tutor. Si la valoración del modelo está por debajo de este umbral, se considera que la pregunta no es relevante y se indica al estudiante que se centre en el nivel.
- `invalid_elems`. Lista de subcadenas utilizadas para filtrar los elementos del entorno. Aquellos cuyo nombre contenga alguna de ellas se consideran irrelevantes y se descartan.
- `max_distance_to_player`. Establece la distancia máxima entre el jugador y un objeto para que dicho objeto sea considerado relevante. Los elementos más alejados se descartan.

## 4.6. Soporte del Entorno ADARVE al Sistema de Tutoría

En esta etapa, también se llevaron a cabo una serie de mejoras en el entorno de ADARVE con el objetivo de facilitar la integración del tutor inteligente y mejorar la experiencia de interacción por parte del usuario.

### 4.6.1. Sistema de Interacción con Tuddy

En la fase inicial del proyecto, se diseñó una versión adaptada del personaje Tuddy, que incluía un panel interactivo a través del cual el usuario podía escribir y comunicarse con el tutor-engine. Sin embargo, en ese momento, la interacción requería que el jugador se desplazara físicamente hasta la ubicación de Tuddy dentro del entorno virtual para iniciar el diálogo.

Uno de los objetivos de esta nueva iteración ha sido garantizar que el tutor esté siempre disponible y que pueda ser consultado en cualquier momento, sin depender de la localización del personaje dentro del nivel. Para ello, se implementó una funcionalidad que permite invocar al tutor pulsando la tecla `T`, asociada tanto al concepto de tutor como al propio personaje de Tuddy.


Figura 4.8: Tuddy con el panel interactivo

La pulsación de esta tecla genera un evento en Unreal Engine que lanza automáticamente la aparición del personaje Tuddy justo frente al usuario, junto con el panel de entrada ya activado y listo para recibir una consulta. La Figura 4.8 muestra el resultado de este proceso.

El comportamiento implementado está diseñado para ser intuitivo. Como se aprecia en la Figura 4.9, si el usuario pulsa la tecla `T` mientras Tuddy ya está presente en pantalla y no está hablando, este desaparecerá. En cambio, si el tutor se encuentra hablando en ese momento, la pulsación de la tecla simplemente lo reposiciona frente al jugador, permitiendo moverlo fácilmente de un lugar a otro sin perder la conversación en curso. Esta funcionalidad fue desarrollada desde cero dentro de ADARVE.

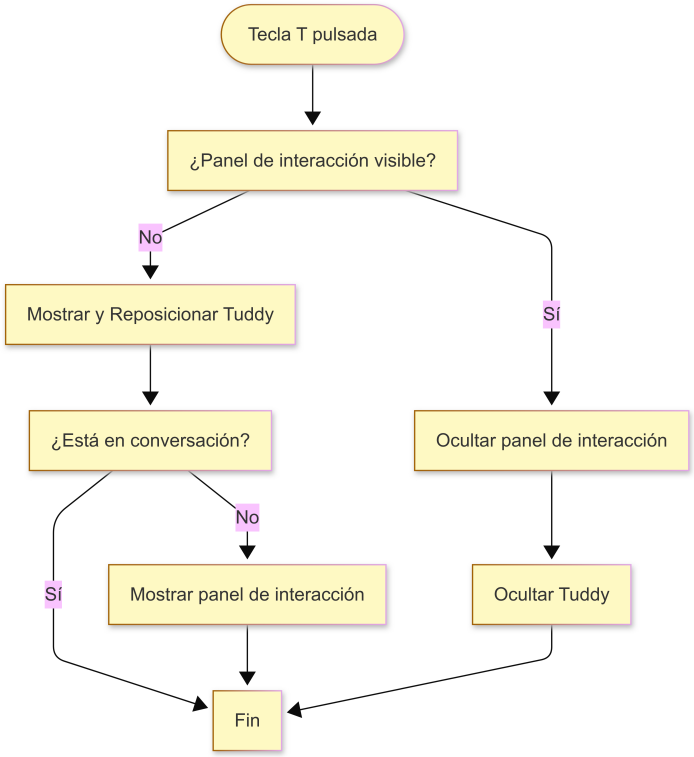


Figura 4.9: Funcionamiento de las interacciones con Tuddy al pulsar la tecla `T`

### 4.6.2. Encapsulamiento e Inicialización del Tutor

Como se ha indicado previamente, una de las prioridades en el desarrollo de este proyecto ha sido mantener el tutor lo más desacoplado posible del resto del entorno de ADARVE. Con este enfoque se busca encapsular su funcionamiento y facilitar su portabilidad, evitando introducir dependencias innecesarias entre módulos.

No obstante, para que el sistema funcione correctamente, es inevitable que exista al menos un punto de contacto entre el código de ADARVE y el módulo *tutor-ADARVE*. Esta conexión se realiza a través del actor `BP_LLM_Manager`, diseñado específicamente para servir de enlace entre el entorno original y el módulo del tutor dentro de ADARVE.

Gracias a esta arquitectura, se garantiza la portabilidad del sistema de tutoría y su integración sencilla en distintos niveles del entorno sin necesidad de modificar otros elementos. Para habilitar el tutor en cualquier nivel de ADARVE, basta con seguir dos pasos muy sencillos:

1. Incluir el actor `BP_LLM_Manager` dentro del mapa correspondiente.

2. Asegurarse de que, al iniciar el nivel, se llama a la función `startLLM()` de dicho actor.

Con estas dos simples acciones, el sistema de tutoría queda completamente funcional en cualquier nivel del entorno. La Figura 4.10 muestra el resultado de seguir estas instrucciones en dos niveles distintos.


Figura 4.10: Tuddy disponible en varios niveles de ADARVE

La responsabilidad del actor `BP_LLM_Manager` es coordinar todos los elementos necesarios para el funcionamiento del tutor dentro de ADARVE. Esto incluye la inicialización de componentes, su incorporación en la nivel y la invocación de sus funciones en los momentos adecuados, como se detallará en la siguiente sección.

### 4.6.3. Responsabilidades de `BP_LLM_Manager`

Una vez introducido el papel general del actor `BP_LLM_Manager` dentro del entorno, en esta sección se desglosan de forma más detallada las tareas específicas que lleva a cabo durante la ejecución de cada nivel, así como su relación con los distintos componentes del sistema de tutoría.

#### 4.6.3.1. Función `startLLM()`

Esta función debe ser invocada al inicio de cada nivel y tiene como objetivo inicializar correctamente todos los componentes implicados en el sistema de tutoría. Sus responsabilidades principales son las siguientes:

1. **Incorporación de actores y widgets.** El módulo *tutor-ADARVE* depende de una serie de actores y elementos visuales: `BP_Socket`, `BP_MessageBuilder`, `BP_TutorialBuddy_LLM` y `WBP_TutorialBuddyInputPanel`. Estos componentes son completamente ajenos al entorno original de ADARVE, por lo que es necesario incorporarlos manualmente al nivel desde esta función.

2. **Asociación de eventos a funciones.** Algunas funcionalidades dependen de la activación de eventos concretos. Por este motivo, es necesario establecer las asociaciones (*binding*) entre los eventos definidos y las funciones correspondientes que deben ejecutarse en respuesta.

3. **Establecimiento de la conexión con el *tutor-engine*.** Esta conexión es indispensable para habilitar la comunicación entre el entorno de ADARVE y el sistema externo encargado de generar las respuestas del tutor.

#### 4.6.3.2. Gestión de eventos

Existen funcionalidades que no pueden ejecutarse de forma inmediata y que requieren esperar a que se reciba una determinada respuesta. No resulta adecuado interrumpir la experiencia del usuario a la espera de estas operaciones, por lo que se gestionan de forma asincrónica mediante eventos. Los principales eventos gestionados por el `BP_LLM_Manager` son los siguientes:

- **onConnectionStablished.** Una vez establecida la conexión con el *tutor-engine*, se inicia el flujo de mensajes definido en la Sección 4.2.1. En concreto, se envía un mensaje de tipo *inicialización* utilizando el actor `BP_MessageBuilder`, y se activa un temporizador que, cada cinco segundos, desencadena el envío automático de mensajes de sincronización.

- **onTutorFeedbackReceived.** Cuando el tutor responde, Tuddy debe aparecer frente al estudiante para mostrar los mensajes generados. El comportamiento visual sigue el protocolo definido previamente en la Sección 3.3.1.

- **onCommitedQuestion.** Al detectar que el estudiante ha realizado una consulta, a través del actor `BP_MessageBuilder` se construye un mensaje de tipo *pregunta* y se envía al tutor utilizando el *socket* gestionado por `BP_Socket`.

## 4.7. Estado del Proyecto: Segundo Prototipo

Esta segunda fase del proyecto representó un avance decisivo en el desarrollo del sistema, culminando con la implementación de una primera versión funcional del tutor (cuyo diseño queda reflejado en la Figura 4.11) capaz de responder a las preguntas formuladas por los estudiantes.

Mientras que la fase anterior se centró únicamente en validar la viabilidad técnica de las comunicaciones, en esta etapa se consolidó el desarrollo del *tutor-engine*, que hasta entonces apenas contaba con lógica implementada. Se definió un sistema estructurado de mensajes, se adaptó un enfoque clásico de tutoría al uso de un modelo de lenguaje de gran escala y se incorporó concurrencia en el servidor, lo que dejó preparado el sistema para una posible extensión multiusuario en futuras versiones.

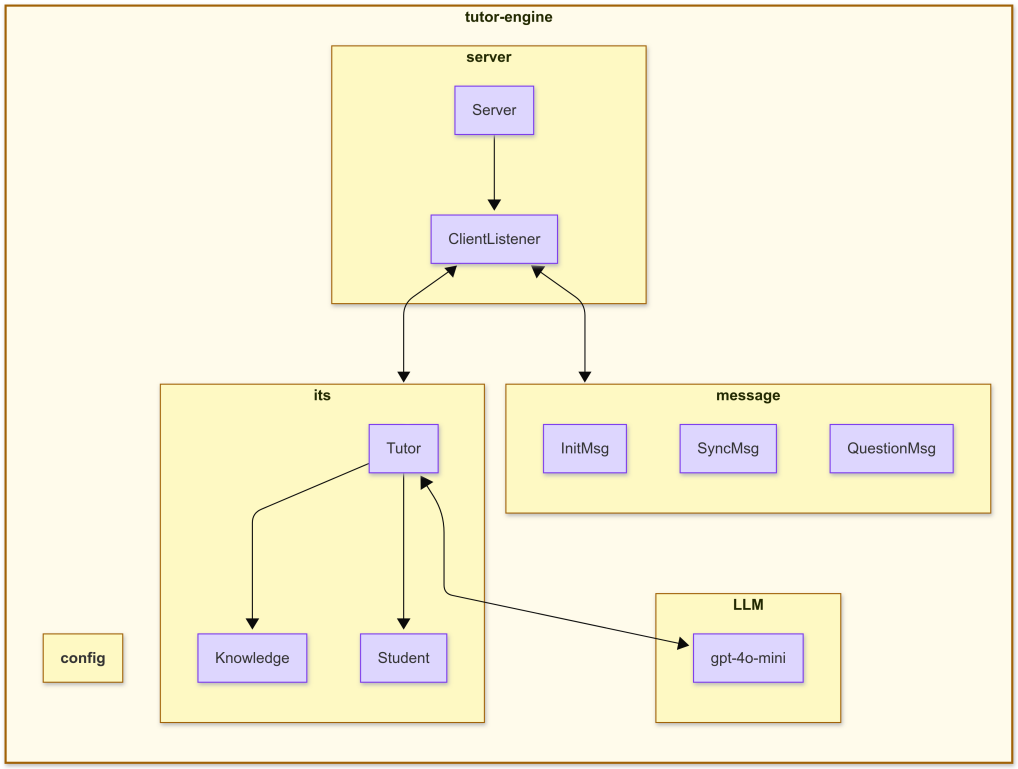


Figura 4.11: Diseño de *tutor-engine* en el prototipo mínimo viable

También se mejoró de forma notable la interacción entre el entorno ADARVE y el tutor, alcanzando un nivel de integración y usabilidad suficientemente sólido como para mantenerse sin modificaciones en la versión definitiva.

Capítulo 5

# Sistema de Tutoría Multiagente

Este capítulo está dedicado a la versión final del sistema de tutoría, en la que se consolidan todas las funcionalidades desarrolladas a lo largo del proyecto y se introduce una nueva arquitectura basada en agentes especializados.

La versión anterior del tutor contemplaba únicamente la funcionalidad de responder a las preguntas del estudiante. En ella, se generaba un contexto único para todas las consultas y se realizaba una única llamada al LLM acompañada de la pregunta del usuario.

En esta última iteración, se ha mejorado significativamente el sistema mediante la creación de contextos adaptados a cada situación para reducir los tiempos de respuesta, la mejora del sistema de filtrado de las consultas de los estudiantes y la incorporación de un mecanismo de reajuste dinámico de las temáticas permitidas. La responsabilidad de procesar las preguntas del usuario recae ahora en un agente específico.

Además, se han desarrollado agentes adicionales encargados de las otras funcionalidades clave que no habían sido abordadas hasta este momento, como la generación de informes y las intervenciones proactivas del tutor. Cada agente opera de forma autónoma y especializada, coordinándose con el resto para ofrecer una experiencia de tutoría completa.

## 5.1. Generación Adaptativa del Contexto

La eficacia en la interacción entre el tutor inteligente y el estudiante depende en gran medida de la calidad y precisión del contexto proporcionado al LLM. En esta sección se describe una estrategia adaptativa diseñada para mejorar la generación del contexto utilizado por el tutor durante cada interacción. Se explica cómo, mediante una reducción y especialización del contexto enviado al LLM, se logra mejorar significativamente el tiempo de respuesta.

### 5.1.1. Reducción del Contexto para Mejorar el Rendimiento

En la versión anterior del tutor se exploró la posibilidad de realizar múltiples llamadas al modelo de lenguaje durante una misma interacción, con el objetivo de implementar un sistema más complejo y preciso. Sin embargo, como se explicó en la Sección 4.3.3.1, esta solución resultó inviable debido a los elevados tiempos de respuesta, lo que afectaba negativamente a la experiencia de usuario. Como consecuencia, se optó por una única llamada al LLM por cada interacción.

En esta nueva versión se volvió a analizar esta limitación, y se identificó que el principal problema no era tanto el número de llamadas, sino la longitud excesiva del contexto enviado en cada una de ellas. Se observó que, en muchas ocasiones, no era necesario incluir toda la información disponible del entorno y del estudiante para generar una respuesta adecuada. Esto abría la puerta a mejorar el sistema sin comprometer la calidad de las respuestas.

A partir de esta observación surgió la idea de diseñar generadores de contexto especializados. Se identificaron los principales tipos de información con los que podría ser necesario alimentar al LLM y, para cada uno de ellos, se creó un generador específico que construía un contexto reducido.

Esta estrategia permitió disminuir significativamente el tiempo de respuesta y mantener la coherencia de las respuestas, adaptándolas mejor a cada situación concreta. Así, el tutor pasó de enviar siempre un único bloque de información extenso a construir contextos flexibles y personalizados según la necesidad de cada interacción.

### 5.1.2. Tipos de Generadores de Contexto

Para reducir la longitud del contexto enviado en cada interacción, se diseñó un sistema basado en generadores de contexto especializados. Todos estos generadores comparten una misma estructura base definida en la clase `ContextGenerator`, la cual actúa como clase padre. Esta clase almacena las referencias al conocimiento general del entorno (`Knowledge`) y al estado del estudiante (`Student`), y ofrece un método `generate()` que puede ser sobrescrito por cada clase hija para construir su propio fragmento de contexto. La Figura 5.1 ilustra esta jerarquía de clases.

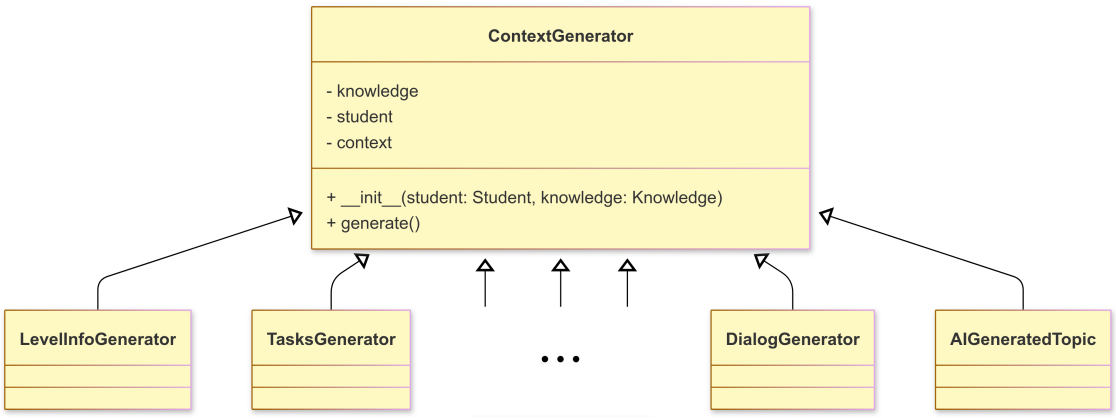


Figura 5.1: Diagrama de clases de los generadores de contexto

A continuación, se describen los distintos generadores implementados:

- **Información del nivel** (`LevelInfoGenerator`). Genera un fragmento de contexto que informa sobre el nombre y la descripción del nivel actual en el que se encuentra el estudiante.
- **Inicio de conversación** (`GreetingGenerator`). Se utiliza cuando el estudiante intenta iniciar una interacción con el tutor. El contexto sugiere al LLM que mantenga un tono cordial y amable, y que se presente solo si es necesario.
- **Entorno de realidad virtual** (`VREnvironmentGenerator`). Presenta una descripción general del entorno virtual, incluyendo explicaciones sobre elementos clave del escenario. Utiliza la información contenida en el tutorial del nivel.
- **Progreso de tareas** (`TasksGenerator`). Describe las tareas que el estudiante debe realizar en el nivel, indicando cuáles han sido completadas y en qué momento, así como las que aún están pendientes.
- **Posición del usuario y elementos del mapa** (`PositionGenerator`). Informa sobre la localización de los distintos elementos del entorno y la posición actual del estudiante. Además, advierte al LLM de que no debe mostrar coordenadas numéricas y que debe expresarse en lenguaje natural.
- **Diálogos con personajes** (`DialogGenerator`). Resume los fragmentos de diálogo que el estudiante ha recibido de los personajes del entorno. Los nombres internos de los personajes se ocultan y se recomienda emplear un lenguaje descriptivo o natural.
- **Historial de conversación** (`PreviousChatGenerator`). Muestra las últimas preguntas realizadas por el estudiante y las respuestas proporcionadas por el tutor. El número de interacciones incluidas está determinado por el archivo de configuración.
- **Contexto personalizado generado por IA** (`AIGeneratedTopic`). Este generador permite construir un contexto arbitrario definido directamente por el tutor, sin depender de una lógica preestablecida. Resulta especialmente útil en situaciones en las que se considere necesario generar dinámicamente un tipo de contexto no contemplado por los generadores existentes. Esta funcionalidad se analiza en mayor profundidad en la Sección 5.3.2.

### 5.1.3. Gestión de los Generadores de Contexto

Para coordinar y organizar de forma eficiente los distintos generadores de contexto descritos en la sección anterior, se diseñó una clase denominada `ContextManager`. Su función principal es centralizar la creación de los distintos fragmentos de contexto que se proporcionan al modelo de lenguaje, facilitando su reutilización, mantenimiento y extensión.

`ContextManager` actúa como una interfaz unificada para el resto del sistema, evitando que otros componentes tengan que gestionar directamente las clases individuales de cada generador. De esta forma, se reduce el acoplamiento entre módulos y se mejora la claridad del diseño general. Este enfoque sigue una estructura inspirada en el patrón de diseño *Facade* (Gamma et al., 1994), ya que proporciona un único punto de acceso simplificado a un conjunto de clases especializadas que gestionan el contenido contextual, tal y como se observa en la Figura 5.2.

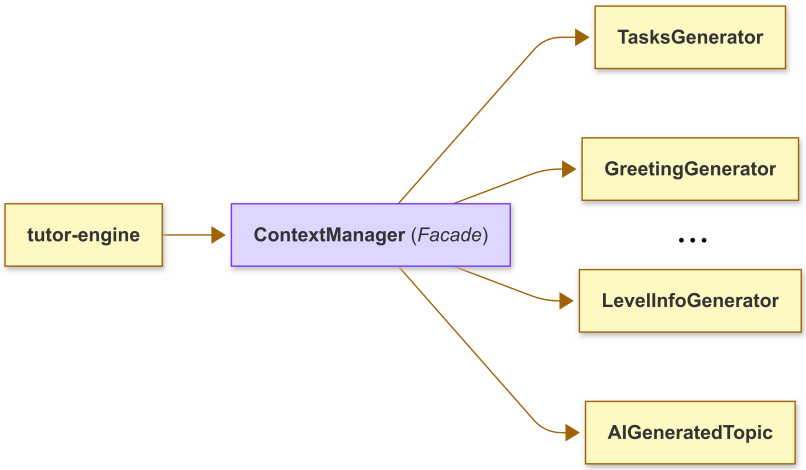


Figura 5.2: Patrón *Facade* aplicado a `ContextManager`

Internamente, la clase almacena instancias de todos los generadores de contexto necesarios, agrupándolos en tres categorías principales:

- **Generadores temáticos predefinidos**. Estos se almacenan en una lista interna y se activan dinámicamente en función del tipo de consulta recibida. Un ejemplo es `TasksGenerator`.

- **Generadores adicionales generados por IA**. Son los creados a través de la clase `AIGeneratedTopic`, que permiten construir contextos definidos por el propio tutor cuando se requiera un contenido que no encaje con las temáticas predefinidas.

- **Generadores complementarios**. Como es el caso de `LevelInfoGenerator`, que proporcionan información común a prácticamente todas las interacciones y, por tanto, no están asociados a ningún tema específico.

Como se muestra en la Figura 5.3, la clase expone una serie de métodos públicos, como `generate_context_question()`, que permiten generar fragmentos completos de contexto. Pero, además, dispone de métodos auxiliares para recuperar partes concretas del contexto, como `generate_tasks_context()`, o añadir nuevos generadores personalizados a través del método `add_topic()`.

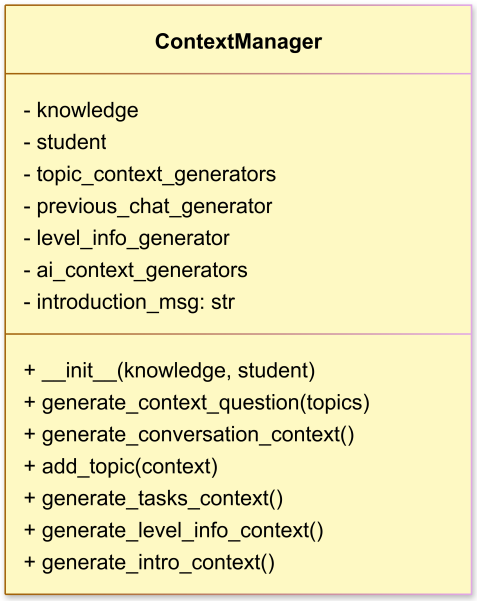


Figura 5.3: Diagrama UML de `ContextManager`

## 5.2. Evolución del Sistema hacia una Arquitectura Multiagente

Como se mencionó previamente en la Sección 2.5.1, los sistemas de tutoría suelen estructurarse en torno a cuatro módulos fundamentales: el módulo de conocimiento, el módulo del estudiante, el módulo del tutor y la interfaz.

Los dos primeros presentan responsabilidades bien delimitadas. El módulo de conocimiento representa la información general que maneja el sistema, mientras que el módulo del estudiante registra el progreso del usuario dentro del entorno de aprendizaje. En cambio, el módulo del tutor asume un papel más complejo, ya que debe coordinar el funcionamiento global del sistema de tutoría a partir de los datos que le proporcionan los otros dos.

En el caso de este proyecto, el módulo del tutor ha de encargarse de tres funciones principales: responder a las consultas realizadas por el usuario, intervenir de forma proactiva cuando sea pertinente y generar informes detallados sobre el progreso del estudiante. Aunque es técnicamente posible implementar estas funcionalidades dentro de un único módulo, esta aproximación implica importantes limitaciones en términos de escalabilidad, legibilidad del código y ejecución simultánea de tareas.

Por este motivo, y teniendo en cuenta los buenos resultados de las arquitecturas multiagente basadas en LLMs en otros proyectos educativos (véase la Sección 2.5.2), se optó por reorganizar el módulo del tutor según este enfoque. En la nueva versión, cada una de las responsabilidades mencionadas se delega en un agente especializado, manteniendo una coordinación central pero distribuyendo las tareas de manera modular. Los módulos de conocimiento y estudiante permanecen sin cambios respecto a la versión anterior.

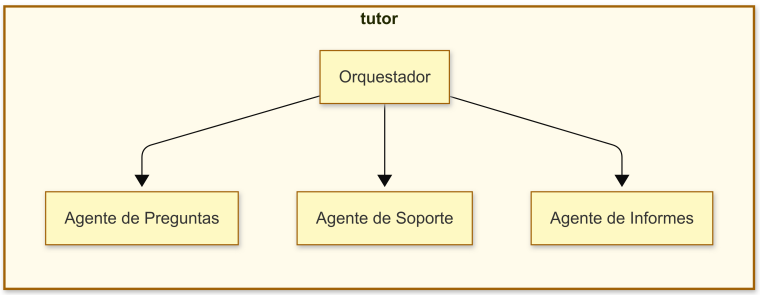


Figura 5.4: Diseño multiagente del tutor

La Figura 5.4 muestra el diseño adoptado. En él, un agente orquestador coordina las acciones de tres agentes especializados: el agente de preguntas, encargado de resolver las consultas del usuario; el agente de soporte, que interviene de manera proactiva para ofrecer ayuda en casos de desorientación; y el agente de informes, responsable de elaborar un informe sobre el estudiante. En las siguientes secciones se describirá el funcionamiento de los tres agentes.

## 5.3. Agente de Preguntas

El agente de preguntas es el responsable de gestionar las consultas realizadas por el estudiante y generar una respuesta adecuada en función del contexto. En versiones anteriores, la respuesta del tutor se generaba a partir de una única llamada al modelo de lenguaje, que incluía tanto el filtrado como la respuesta en una misma interacción.

En la arquitectura actual, al habilitarse la posibilidad de realizar múltiples llamadas al LLM por cada interacción, se ha desacoplado el filtrado de la generación de respuestas. Es más, el filtrado ha evolucionado hacia un sistema de clasificación.

En primer lugar, el agente analiza la pregunta del usuario y la categoriza según una serie de temas predefinidos. Si la consulta no encaja en ninguno de estos temas, se considera irrelevante en el contexto del entorno de aprendizaje y el tutor devuelve un mensaje indicándole al usuario que se centre en el nivel.

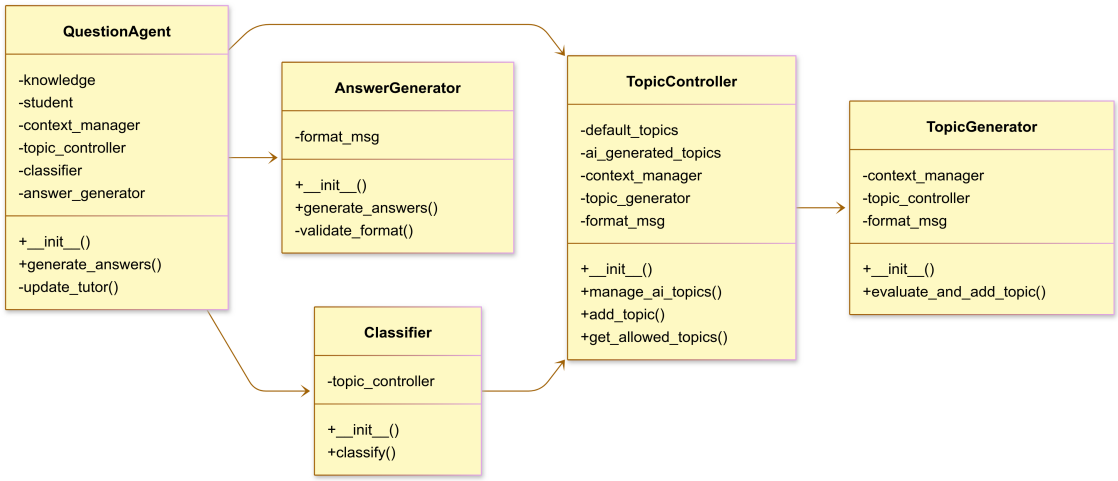


Figura 5.5: Diagrama de clases del agente de preguntas

Adicionalmente, como se explica en la Sección 5.3.2, el sistema incorpora un mecanismo de expansión dinámica de temas basándose en el historial de conversaciones. Esto permite que, a medida que avanza la conversación, se puedan identificar nuevas líneas temáticas relevantes y generar sobre la marcha contextos específicos adaptados a ellas.

La Figura 5.5 muestra todas las clases involucradas en el funcionamiento del agente. Por un lado, `Classifier` trabaja conjuntamente con `TopicController` y `TopicGenerator` para clasificar las preguntas del usuario, mientras que la clase `AnswerGenerator` se encarga de generar las respuestas correspondientes. En las siguientes secciones se detallan las responsabilidades específicas de cada una y sus relaciones dentro del sistema.

### 5.3.1. Temáticas Predefinidas y Clasificador

En esta versión, se definió un conjunto de temas preestablecidos que delimitan el ámbito de las preguntas al tutor. Estas temáticas cubren los principales tipos de consultas que se espera que el estudiante formule durante su interacción con el entorno de realidad virtual.

Los temas predefinidos cumplen una doble función. Por un lado, actúan como filtro, evitando que el tutor responda a preguntas que no sean pertinentes dentro del contexto educativo. Por otro, permiten determinar qué información debe incluirse en el contexto que se proporciona al modelo de lenguaje. Cada temática está asociada a un generador de contexto específico (descritos en la Sección 5.1.2) encargado de construir la información relevante sobre dicho tema.

Las temáticas definidas inicialmente son las siguientes:

1. **Saludos e introducciones**. Se refiere a frases básicas para iniciar una conversación con el tutor, como presentaciones o expresiones de cortesía. Este tipo de consultas activa el `GreetingGenerator`.

2. **Descripción de objetos y funcionamiento del entorno**. Abarca preguntas relacionadas con los objetos interactivos y el uso del entorno de realidad virtual. Se gestiona mediante el `VREnvironmentGenerator`.

3. **Tareas del nivel**. Agrupa las preguntas sobre qué acciones debe realizar el estudiante para avanzar en el nivel actual. Esta temática se gestiona a través del `TasksGenerator`.

4. **Ubicación de objetos**. Cubre las consultas relativas a la localización de elementos en el entorno. El `PositionGenerator` se encarga de construir el contexto para este tipo de preguntas.

5. **Diálogos del entorno**. Agrupa aquellas preguntas que hacen referencia a información proporcionada por los personajes del entorno y se resuelve mediante el `DialogGenerator`.

El clasificador, implementado en la clase `Classifier`, consulta al modelo de lenguaje para que devuelva una lista con los índices de todas las temáticas relevantes

identificadas en la frase del usuario. En caso de que ninguna temática sea reconocida, se devuelve una lista vacía.

**Ejemplo de clasificación de una consulta**

**Entrada del usuario:** `¡Hola! ¿Qué es un dosímetro?`

**Temáticas detectadas:** `[0, 1]`

- `0` → Saludos e introducciones
- `1` → Descripción de objetos y funcionamiento del entorno

Dado que esta versión del tutor realiza múltiples llamadas al modelo de lenguaje, se trató de reducir al máximo el tiempo de respuesta de cada interacción. Por este motivo, se reconsideró el uso del modelo `gpt-3.5-turbo` para las llamadas al LLM realizadas desde esta clase, al ser, según OpenAI su modelo más rápido. Sin embargo, tras realizar un estudio similar al descrito en la Sección 4.1.2, se concluyó que sus resultados no alcanzaban la calidad esperada, por lo que se decidió mantener `gpt-4o-mini` como modelo principal.

Además del conjunto base de temáticas, el sistema permite la incorporación dinámica de nuevas categorías cuando el tutor identifica líneas de conversación relevantes no contempladas inicialmente. Este mecanismo se describe con más detalle en la siguiente sección.

### 5.3.2. Generación Dinámica de Temas mediante Inteligencia Artificial

Una vez se ha generado una respuesta a la pregunta del usuario, el sistema evalúa el historial completo de interacciones con el objetivo de analizar la evolución de la conversación y determinar si ha surgido información relevante que deba incorporarse en futuras clasificaciones.

En primer lugar, se revisan las temáticas ya generadas por el tutor. En caso de que la conversación haya avanzado en alguna de esas direcciones, se actualiza su contexto correspondiente. Posteriormente, se analiza si es necesario incorporar una nueva temática, basada en la información aportada por el estudiante durante el diálogo.

#### 5.3.2.1. Creación de una nueva temática

La creación dinámica de temas está gestionada por la clase `TopicGenerator`, a través del método `evaluate_and_add_topic()`. En este método se realiza una consulta al modelo de lenguaje, proporcionándole la lista de temáticas existentes y el historial de conversaciones mantenidas entre el usuario y el tutor. Si el modelo detecta un nuevo tema relevante, debe devolver un objeto JSON con la siguiente estructura:

```
{
"topic": "nombre del nuevo tema",
"context": "contexto asociado al tema"
}
```

En caso de recibir una respuesta en este formato, se añade la temática al atributo `ai_generated_topics` de la clase `TopicController` y se incorpora un nuevo generador de contexto del tipo `AIGeneratedTopic` dentro del `ContextManager`, utilizando el contexto proporcionado por el LLM. De este modo, la nueva temática será tenida en cuenta en futuras clasificaciones y construcciones de contexto.

Durante el desarrollo de esta funcionalidad se detectaron ciertas limitaciones en la interpretación del modelo, que en ocasiones generaba temáticas demasiado específicas o innecesarias. Para evitarlo, se diseñó un prompt detallado que guía al modelo en la toma de decisiones. Aquí se muestra una parte:

```
Solo debes añadir un nuevo tema si es claramente necesario, es decir, si las
conversaciones incluyen preguntas relevantes y recurrentes que no encajan
razonablemente en ninguno de los temas ya existentes.

Los temas permitidos deben ser amplios y generales, no específicos o demasiado
concretos.
No propongas nuevos temas basados en una única pregunta aislada o en variantes
menores de temas ya existentes.
En general, no será necesario incluir ningún tema nuevo
```

La clase `TopicGenerator` también incorpora mecanismos de validación mediante la función `validate_format()`, que garantiza que el formato de la respuesta se ajusta al esperado. En caso contrario, se repite la llamada al modelo hasta obtener un resultado correcto. Si no se identifica ninguna temática nueva, la respuesta esperada es vacía.

Además, es importante destacar que el sistema distingue entre preguntas clasificadas como relevantes y aquellas fuera de contexto. Para evitar generar temáticas a partir de contenido irrelevante, solo se proporciona al modelo el conjunto de interacciones que han sido previamente clasificadas dentro de una temática válida.

A continuación, se presenta un ejemplo real en el que el tutor detecta una nueva temática a partir de una conversación con el estudiante:

**Usuario:** `Hola tuddy, ¿qué es un dosímetro?`
**Tuddy:** `['¡Hola! Un dosímetro es un dispositivo que mide la radiación.', 'El dosímetro EPD proporciona lecturas continuas.', 'Te ayuda a conocer la dosis acumulada y la tasa actual.', 'Es recomendable llevarlo siempre contigo.']`

—

**Usuario:** `Entonces, ¿los dosímetros son útiles en emergencias radiológicas?`
**Tuddy:** `['Esa es una excelente pregunta.', 'Sin embargo, en este nivel aprenderás lo básico de la realidad virtual.', 'Te animo a que sigas explorando las funcionalidades.', 'Más adelante podrás profundizar en temas como el dosímetro.']`

—

**Usuario:** `¿Qué otros objetos tengo que llevar encima en situaciones de emergencia radiológica?`
**Tuddy:** `['Además del dosímetro, es importante llevar un kit de primeros auxilios.', 'También es recomendable tener una linterna y baterías de repuesto.', 'No olvides llevar agua potable y alimentos no perecederos.', 'Por último, un mapa y una brújula pueden ser útiles para orientarte.']`

—

**Usuario:** `Dime objetos útiles en situaciones de emergencias radiológicas`
**Tuddy:** `['Además del dosímetro, lleva un kit de primeros auxilios.', 'Una linterna y baterías de repuesto son esenciales.', 'No olvides agua potable y alimentos no perecederos.', 'Un mapa y una brújula pueden ayudarte a orientarte.']`

A partir de esta conversación, el modelo de lenguaje devuelve el siguiente resultado en formato JSON, indicando tanto la nueva temática detectada como el contexto asociado:

- **Nuevo tema:** Objetos útiles en situaciones de emergencia radiológica
- **Contexto:** El estudiante ha preguntado sobre objetos que son importantes llevar en situaciones de emergencia radiológica. Se ha mencionado la importancia de un dosímetro, un kit de primeros auxilios, una linterna con baterías de repuesto, agua potable, alimentos no perecederos, un mapa y una brújula para la orientación.

#### 5.3.2.2. Actualización de temáticas ya existentes

Antes de proceder a la creación de nuevas temáticas, el sistema revisa las ya generadas dinámicamente por el tutor. La clase `TopicController` gestiona esta tarea mediante el método `manage_ai_topics()`. Para ello, se envía al modelo de lenguaje la misma información que en el proceso de detección de nuevos temas, solicitándole en este caso que actualice únicamente aquellos contextos cuya modificación resulte necesaria.

Durante las primeras pruebas, se observó que el modelo tendía a reescribir los contextos con cambios superficiales o irrelevantes, como la sustitución de conectores o el uso de sinónimos que no aportaban valor. Para evitar esta conducta, fue necesario ajustar el prompt, indicando explícitamente que las modificaciones deben limitarse a aquellas situaciones en las que la información nueva sea realmente significativa.

El formato esperado para la respuesta es una lista de objetos JSON, cada uno de los cuales contiene el índice del tema a modificar y el nuevo contexto asociado:

```
[
    {
    "topic_index": 0,
    "context": "nuevo contexto"
    },
    ...
]
```

En caso de que no sea necesario modificar ningún contexto, el modelo debe devolver una lista vacía.

Continuando con el ejemplo de la subsección anterior, supongamos que el estudiante plantea la siguiente consulta adicional:

**Usuario:** ¿Es un avión útil en una situación de emergencia radiológica?
**Tuddy:** ['Un avión no suele estar disponible para la mayoría de las personas en una emergencia radiológica.', 'Sin embargo, los medios de transporte pueden ser útiles para evacuar zonas peligrosas si las autoridades lo indican.', 'Lo más importante es seguir las instrucciones oficiales y tener tu equipo de emergencia listo.']

A raíz de esta nueva interacción, el sistema actualiza el contexto del tema ya existente de la siguiente manera:

- **Nuevo contexto:** El estudiante ha preguntado sobre objetos que son importantes llevar en situaciones de emergencia radiológica. Se ha mencionado la importancia de portar un dosímetro para medir la radiación, un kit de primeros auxilios, una linterna con baterías de repuesto, agua potable, alimentos no perecederos, un mapa y una brújula para la orientación. También se ha considerado el uso potencial de un avión en evacuaciones estratégicas, teniendo en cuenta su disponibilidad y autorización dentro del protocolo de emergencia.

### 5.3.3. Generación de Respuestas

Una vez conocidos en detalle los actores que intervienen en el funcionamiento del agente de preguntas, en este apartado se expone de forma clara cómo se combinan entre sí para generar una respuesta coherente y adaptada al contexto. Se busca con ello proporcionar una visión general del flujo interno del sistema desde que el

*tutor-engine* recibe una consulta hasta que se devuelve una respuesta, proceso que se ilustra mediante el diagrama de la Figura 5.6.

El proceso de generación de respuestas del agente comienza cuando el componente `Orchestrator`, del que se hablará en la Sección 5.6.1, ejecuta el método `generate_answers(question)`. Esta acción desencadena un flujo de trabajo que inicia con la clasificación de la pregunta mediante el `Classifier`. Para ello, se emplea el método `classify(question)`, que devuelve una lista con los índices correspondientes a las temáticas relevantes relacionadas con la consulta del estudiante.

Una vez obtenida la clasificación, el `ContextManager` construye el contexto adecuado mediante el método `generate_context_question(classification)`. Este contexto se compone combinando los generadores específicos asociados a cada temática identificada. Independientemente del resultado de la clasificación, siempre se incluye un contexto introductorio general, información sobre el nivel actual del estudiante proporcionada por `LevenInfoGenerator` y el historial de conversaciones anteriores facilitado por `PreviousChatGenerator`. Si la pregunta no puede clasificarse bajo ninguna temática, el método incluye un contexto específico para indicarle al estudiante, de forma amable, que su consulta no está relacionada con el contenido del nivel actual.

Con el contexto ya preparado, el componente `AnswerGenerator` genera la respuesta, recibiendo tanto la pregunta original como el contexto elaborado. Este componente interactúa con el modelo de lenguaje y produce una respuesta que cumple con las restricciones de extensión mencionadas en la Sección 3.3.1.1.

Para garantizar rapidez en la respuesta al estudiante, el resultado generado se envía inmediatamente al `Orchestrator`, reduciendo al máximo las demoras perceptibles por el usuario. Simultáneamente, se ejecuta en segundo plano el método `update_tutor(...)`, encargado de realizar un análisis adicional de la interacción.

Este análisis posterior almacena la interacción para futuras referencias si la pregunta fue clasificada en alguna temática válida. De lo contrario, la consulta se registra como fuera de contexto. Finalmente, el hilo secundario también invoca al método `manage_ai_topics()` del componente `TopicController`, encargado de gestionar las temáticas existentes y promover la creación de nuevas cuando sea necesario.

## 5.4. Agente de Soporte

El agente de soporte es el encargado de intervenir de forma proactiva en el entorno de ADARVE, sin necesidad de que el estudiante solicite ayuda explícitamente. Su objetivo es ofrecer orientación ante situaciones como la inactividad prolongada o signos de desorientación por parte del usuario.

Este comportamiento se basa en el principio pedagógico del *scaffolding*, expuesto en la Sección 2.4.1, y en las investigaciones señaladas en la Sección 2.5.2, que respaldan el uso de un agente de soporte.

Para ello, el agente mantiene constantemente actualizadas las instancias de las clases `Student` y `Knowledge`, cada vez que se recibe un mensaje de sincronización desde el entorno de realidad virtual. Estas actualizaciones permiten monitorizar el estado del alumno en tiempo real.

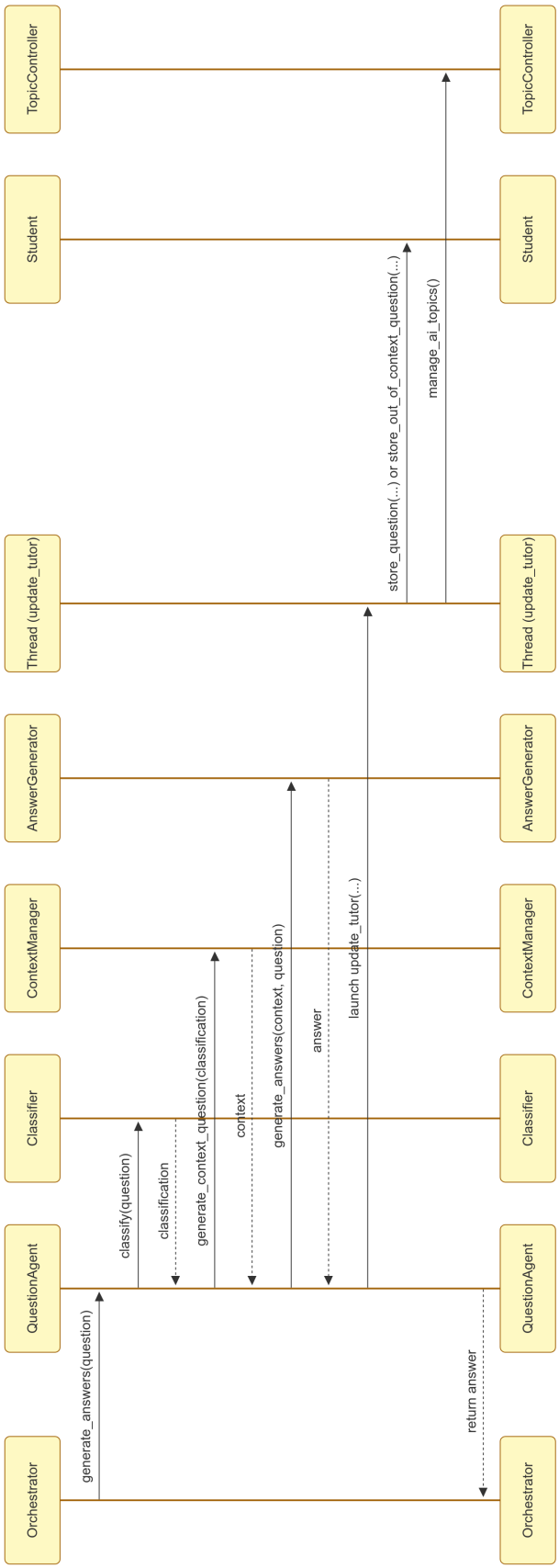


Figura 5.6: Diagrama de secuencia de generación de respuestas

A través del archivo de configuración del tutor, es posible establecer ciertos umbrales para parámetros clave relacionados con el comportamiento del estudiante. Si alguno de estos valores excede los límites definidos, Tuddy aparece frente al jugador para ofrecerle una intervención adecuada.

Aunque los límites iniciales se configuran manualmente, el tutor dispone de mecanismos para adaptarlos dinámicamente en función del progreso y las necesidades observadas durante la sesión, que se describen más adelante en esta sección.

### 5.4.1. Umbrales de Intervención

Para garantizar intervenciones oportunas del agente de soporte, es necesario establecer ciertos parámetros que indiquen cuándo debe actuar proactivamente. Estos parámetros funcionan como indicadores del estado del estudiante dentro del entorno ADARVE, ayudando a identificar situaciones como inactividad prolongada, desorientación o comportamiento desorganizado, activando así las intervenciones del tutor.

Los límites para cada parámetro son administrados por la clase `Thresholds`, cuya responsabilidad es encapsular los umbrales configurables de manera estructurada. Dichos umbrales se definen en el archivo de configuración `config.py` y se cargan automáticamente al iniciar cada sesión.

Los parámetros contemplados son los siguientes:

- **Tiempo de inactividad**. Es el periodo en segundos desde la última acción del estudiante. Las acciones incluyen cambiar de posición o dirección de la mirada, completar tareas e interactuar con otros personajes, incluido Tuddy. Se asocia al campo `inactivity_time` del archivo de configuración.

- **Preguntas fuera de contexto**. Indica cuántas preguntas del usuario han sido clasificadas como no relacionadas con los temas permitidos. Este valor se controla mediante el campo `out_of_context_questions`.

- **Permitir tareas fuera de orden**. Determina si se permite que el estudiante realice tareas en un orden diferente al planificado. Este comportamiento se configura a través del parámetro `allow_out_of_order_tasks`.

- **Tiempo desde la última tarea completada**. Mide el número de segundos que han pasado desde que el estudiante finalizó la última tarea. Su valor límite se define mediante el campo `last_completed_task`.

### 5.4.2. Ajuste Dinámico de los Límites de Intervención

La clase `InterventionController` contiene la lógica necesaria para tomar decisiones sobre cuándo intervenir y actualizar dinámicamente los umbrales establecidos.

Tras procesar un mensaje de sincronización, se envían al modelo de lenguaje los valores actuales de los parámetros, junto al historial previo, para evaluar si se necesitan ajustes.

Además, se proporcionan instrucciones explícitas al modelo para evitar ajustes arbitrarios que generen comportamientos oscilantes, principalmente incrementos y decrementos continuos sin razón aparente. Este tipo de comportamiento fue detectado durante las pruebas iniciales y se logró corregir indicándole que no debe revertir cambios recientes ni modificar los umbrales sin motivo.

Una buena razón para actualizar los umbrales es que el tutor haya decidido intervenir en ADARVE. De no hacerlo, podrían producirse interrupciones sucesivas no deseadas, ya que el parámetro en cuestión seguiría excediendo el mismo valor límite en cada comprobación. Por ejemplo, si el estudiante supera el número máximo permitido de preguntas fuera de contexto y dicho umbral no se incrementa, el sistema continuará activando intervenciones de manera reiterada.

Durante las pruebas también se observó que, en ocasiones, el LLM proponía incrementos insuficientes, lo que resultaba en intervenciones demasiado frecuentes. Por este motivo, se estableció como regla que cualquier nuevo valor para un umbral superado debe, como mínimo, duplicar el valor anterior.

A continuación, se muestra una parte del contexto proporcionado al modelo para ayudarle en su toma de decisiones:

```
Eres un sistema inteligente encargado de ajustar dinámicamente los parámetros de
intervención de un tutor en un entorno de aprendizaje en realidad virtual, enfocado
 en situaciones de emergencia radiológica.

Tu objetivo es garantizar que el tutor intervenga en el momento justo: ni demasiado
 pronto (para evitar interrupciones innecesarias), ni demasiado tarde(para prevenir
 la frustración del estudiante)

Cuando ciertos datos superan los umbrales establecidos, el tutor interviene.Tu
tarea consiste en ajustar dichos umbrales según el contexto actual. A continuación,
 se muestran los umbrales que puedes modificar:

(...)

Puedes modificar un umbral, varios o ninguno, según consideres necesario. Solo
debes modificar un umbral si hay una razón clara basada en el nuevo contexto del
jugador. No alternes valores de forma arbitraria ni reviertas cambios recientes sin
 justificación. Evita devolver valores diferentes si el contexto no ha cambiado
significativamente.

Además, siempre que el tutor intervenga deberás actualizar los umbrales que se han
sobrepasado, por lo menos duplicando su valor en caso de ser numéricos. Por ejemplo
, si un jugador supera el umbral de preguntas fuera de contexto, hay que
incrementar dicho umbral para que no se le siga interrumpiendo constantemente por
el mismo motivo.

(...)
```

Cuando el modelo decide realizar cambios, responde con un JSON que debe seguir el formato:

```
{
"Tiempo de inactividad": segundos,
"Preguntas fuera de contexto": numero,
"Permitir tareas fuera de orden": booleano,
"Tiempo desde la ultima tarea completada": segundos
}
```

Cada vez que se actualizan los umbrales, se crea una nueva instancia de la clase `Thresholds` con los nuevos valores. Esta estrategia permite conservar un historial

de los ajustes aplicados a lo largo del tiempo que posteriormente se le proporciona al LLM.

A modo de ejemplo, extraído directamente de los *logs* del *tutor-engine*, tras superarse el tiempo máximo de inactividad permitido, el `InterventionController` actualizó los umbrales de intervención duplicando el valor asociado al tiempo de inactividad, que inicialmente era de 60 segundos. El resto de parámetros se mantuvieron sin modificaciones:

```
{
"Tiempo de inactividad": 120,
"Preguntas fuera de contexto": 4,
"Permitir tareas fuera de orden": False,
"Tiempo desde la ultima tarea completada": 70
}
```

### 5.4.3. Acciones de Soporte

La comprobación de los umbrales definidos se realiza de manera convencional en el `InterventionController`, mediante sentencias de tipo `if/else`. En caso de que alguno de los valores actuales supere el umbral correspondiente, el *tutor-engine* ejecutará una intervención adecuada.

Estas intervenciones han sido modeladas como acciones, siguiendo el principio de diseño *Command* (Gamma et al., 1994). Como se observa en la Figura 5.7, cada acción está representada por una clase que hereda de la clase abstracta `Action`, la cual contiene un único método `perform()`. Para añadir nuevos tipos de acciones, basta con crear una nueva clase hija con su propia implementación del método `perform()`, sin necesidad de modificar el resto del sistema.

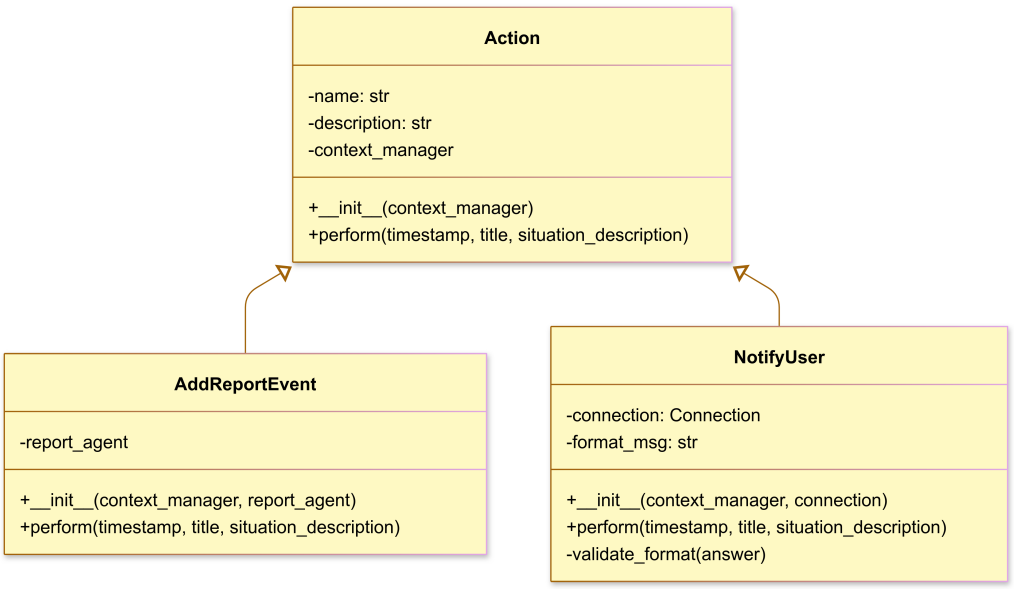


Figura 5.7: Diagrama de clases de las acciones

En esta versión se han implementado dos tipos de acciones:

- `NotifyUser`. Hace que Tuddy aparezca en el entorno de ADARVE para ofrecer ayuda al usuario. Debe pasar al menos un minuto desde la última vez que se eligió esta acción para que se pueda volver a tomar.

- `AddReportEvent`. Registra un evento en el agente de informes, para que este lo tenga en cuenta a la hora de generar el informe final sobre el desempeño del estudiante.

La lógica que determina qué acciones tomar se gestiona en la clase `ActionManager` (véase la Figura 5.8) que, de forma similar a lo explicado en secciones anteriores, consulta al LLM indicando el listado de acciones disponibles, su descripción y el historial de intervenciones previas, para decidir cuáles ejecutar en cada momento.

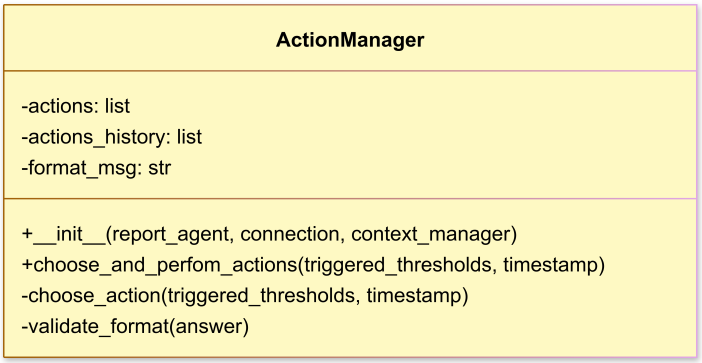


Figura 5.8: Diagrama UML de `ActionManager`

Continuando con el ejemplo de la sección anterior, en ese caso el tutor decidió ejecutar ambas acciones. Por un lado, Tuddy intervino ofreciendo ayuda directamente en el nivel mediante el siguiente diálogo:

**Tuddy:** `['Hola, parece que estás un poco desorientado.', 'Recuerda que debes salir de la habitación primero.', 'Luego, habla con la persona en la habitación grande.', 'Estoy aquí para ayudarte, ¡ánimo!']`

Esta intervención se visualiza en ADARVE de la misma forma que el resto de conversaciones, tal y como se aprecia en la Figura 5.9, solo que Tuddy aparece de manera repentina frente al usuario sin ser invocado.


Figura 5.9: Intervención proactiva de Tuddy tras superarse el umbral de inactividad

Al mismo tiempo, el tutor registró un evento para el agente de informes con la siguiente descripción:

> 'El alumno ha estado inactivo durante 61.125 segundos, superando el umbral establecido de 60 segundos. Esto puede indicar señales de desorientación. Se procede a registrar esta situación en el informe y se interviene directamente con el estudiante para ofrecerle ayuda.'

A continuación, se presentan los resultados obtenidos al repetir la misma prueba, pero esta vez configurando un umbral de inactividad de 10 segundos, con el fin de evaluar la capacidad de adaptación del sistema. Los siguientes *logs* fueron generados por el *tutor-engine* durante dicha ejecución:

**Se ha añadido un evento al reporte:** 'El alumno ha estado inactivo durante 10.096 segundos, superando el umbral establecido de 10 segundos. Esto indica señales de desorientación. Se registrará esta situación en el informe y se intervendrá directamente con el estudiante para ofrecerle ayuda.'
**Intervención de Tuddy:** `['¡Hola! Parece que no tienes claro el siguiente paso.', 'Lo primero es salir de esta habitación.', 'Después, busca a la persona que está en la sala grande.', 'Estoy aquí para ayudarte, ¡vamos a continuar!']`
**El tutor ha actualizado los umbrales de intervención:** 'Tiempo de inactividad': 25, 'Preguntas fuera de contexto': 4, 'Permitir tareas fuera de orden': False, 'Tiempo desde la última tarea completada': 60
**Se ha añadido un evento al informe:** 'El alumno ha estado inactivo durante 25.099 segundos, superando el umbral de 25 segundos. Se ha intervenido previamente, por lo que se registra la situación actual en el informe para su análisis posterior.'
**El tutor ha actualizado los umbrales de intervención:** 'Tiempo de inactividad': 50, 'Preguntas fuera de contexto': 4, 'Permitir tareas fuera de orden': False, 'Tiempo desde la última tarea completada': 60

**Se ha añadido un evento al informe:** 'El estudiante ha superado el umbral de tiempo de inactividad, alcanzando 50.103 segundos sin realizar ninguna acción. Anteriormente, se realizaron intervenciones y registros en el informe, pero no se puede intervenir de nuevo hasta que pase un minuto desde la última intervención. Se procederá a registrar la situación actual en el informe.'
**El tutor ha actualizado los umbrales de intervención:** 'Tiempo de inactividad': 100, 'Preguntas fuera de contexto': 4, 'Permitir tareas fuera de orden': False, 'Tiempo desde la última tarea completada': 60

De estos registros se observa que, tras cada intervención, el tutor tiende a duplicar el umbral de inactividad, aunque no siempre es el caso, pues inicialmente lo aumentó de 10 segundos a 25 segundos. Esta diferencia en el primer aumento puede deberse a que se le pide al LLM "*incrementar dicho umbral para que no se le siga interrumpiendo constantemente*" combinado con que 10 segundos de inactividad es un valor bajo. Además, también respeta la regla de no volver a intervenir en ADARVE hasta pasado un minuto desde la última vez.

### 5.4.4. Sistema de Intervención

Veamos de manera clara el funcionamiento del agente de soporte, una vez se recibe un mensaje de sincronización desde ADARVE.

El proceso de actuación del agente de soporte, ilustrado en la Figura 5.10, se inicia cuando el `Orchestrator`, invoca el método `manage_sync(sync_msg)` del `SupportAgent`, proporcionándole un mensaje de sincronización.

Una vez recibido el mensaje, el agente de soporte actualiza internamente el estado del estudiante y del conocimiento mediante las funciones `student.update(sync_msg)` y `knowledge.update(sync_msg)`, respectivamente.

Con ambos módulos actualizados, el `SupportAgent` solicita a su instancia de `InterventionController` que analice si se ha alcanzado alguno de los límites establecidos para desencadenar una intervención. Esta comprobación se realiza mediante la llamada a `decide_intervention(timestamp)`, que devuelve una lista con los umbrales superados.

Si el sistema detecta que se ha superado al menos uno de estos límites, el agente de soporte recurre al `ActionManager` para decidir y ejecutar las acciones correspondientes. Para ello, se llama a `choose_and_perform_actions(triggered_thresholds, timestamp)`, que decide y actúa en consecuencia sobre si Tuddy debe intervenir directamente en ADARVE para asistir al usuario, si se debe registrar un evento en el informe de seguimiento o ambas.

Como paso final, el `SupportAgent` solicita al `InterventionController` que actualice los valores de los umbrales que se han superado mediante la función `update_thresholds(timestamp, triggered_thresholds)`. Esta actualización garantiza que no se produzcan múltiples intervenciones consecutivas por una misma causa, adaptando los parámetros del sistema a las necesidades del estudiante.

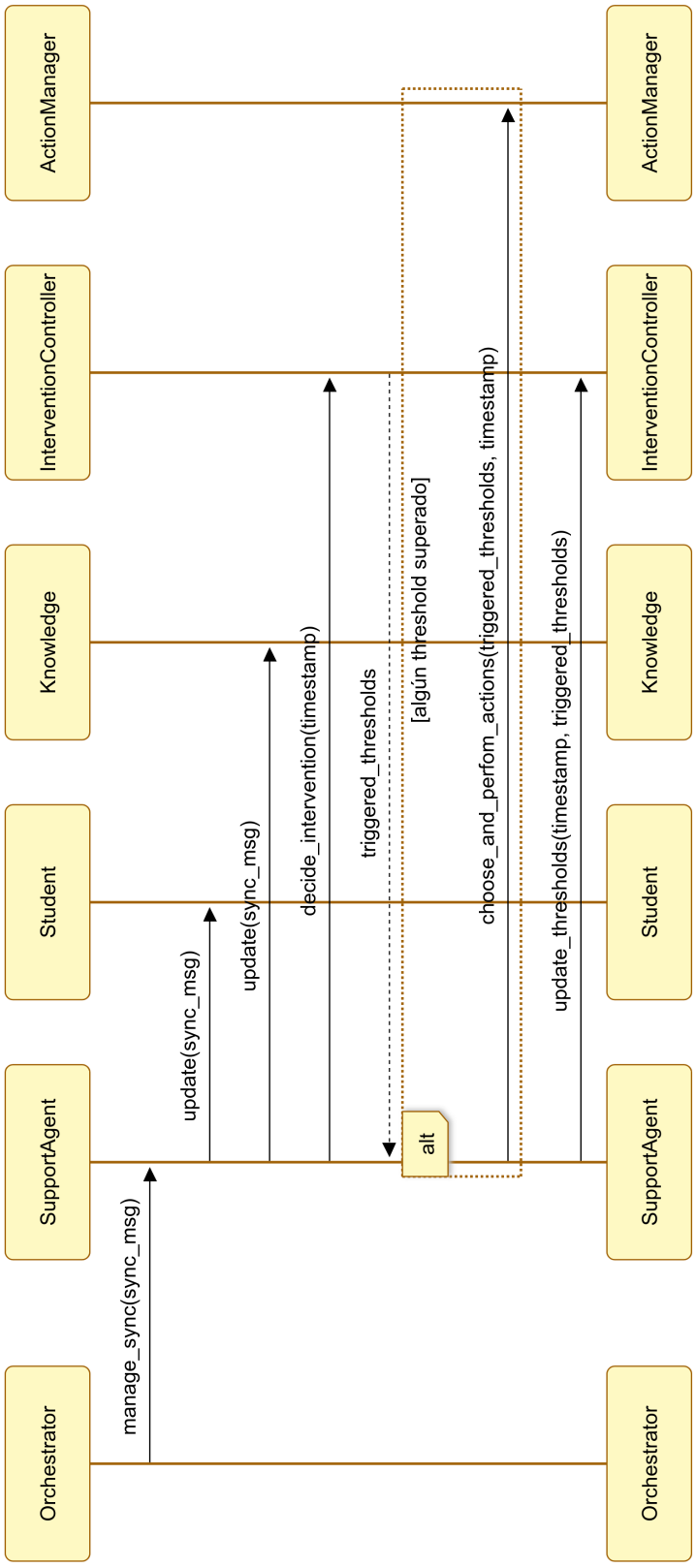


Figura 5.10: Diagrama de secuencia del agente de soporte

## 5.5. Agente de Informes

El agente de informes es el responsable de elaborar un documento que sintetiza el desempeño del estudiante durante la sesión. A diferencia de los otros agentes, que intervienen activamente durante la experiencia de aprendizaje, este agente permanece en segundo plano y su intervención se limita al cierre de la partida.


Figura 5.11: Diagrama UML de `ReportAgent`

Durante el desarrollo del nivel, únicamente se registran eventos relevantes, tal y como se detalló en la Sección 5.4.3. Una vez finalizada la sesión, el agente usa estos datos, junto a la información recogida por las instancias de las clases `Student` y `Knowledge` para generar un informe del proceso formativo, mediante la función `generate_report()`. El resto de métodos y atributos se muestran en la Figura 5.11.

### 5.5.1. Formato del Informe

El informe es un archivo PDF cuyo contenido es generado por el LLM. Si bien el formato puede sufrir ligeras variaciones en cada sesión de aprendizaje, se proporcionan al modelo unas instrucciones bien definidas para estructurar el contenido, siguiendo una serie de secciones obligatorias:

- **Título**. Debe comenzar con la frase "Informe de Desempeño del Estudiante en el Nivel...", seguido del nombre del nivel.
- **Descripción del Nivel**. Incluye una breve introducción al objetivo del nivel actual y lo que se espera que el estudiante aprenda.
- **Progreso del Estudiante**. Muestra un listado con las tareas propuestas y su estado (completadas o no), junto con los tiempos en los que se realizaron si corresponde.
- **Eventos Destacables**. Expone los momentos más relevantes de la sesión. Aunque se parte de los eventos registrados durante la partida (véase la Sección 5.4.3), el modelo genera eventos adicionales basados en las conversaciones con el estudiante y en el estado de su progreso.

- **Recomendaciones**. Se incluyen consejos personalizados enfocados a mejorar el desempeño futuro del estudiante, considerando su actuación en la partida.
- **Conclusión**. Resume el desempeño general del estudiante en el nivel, destacando los logros y las áreas de mejora.

Adicionalmente, el modelo recibe también una serie de restricciones estilísticas. Debe ser claro, conciso y profesional, evitando referencias a su propio rol como tutor y enfocándose únicamente en el progreso y comportamiento del estudiante. Puede emplear elementos de formato como listas, negritas o saltos de línea para mejorar la legibilidad.

Una vez generado el contenido, ya sin recurrir al LLM, se aplica un estilo propio, que emplea la fuente `Arial` por su buena legibilidad, acompañada de un margen interno de `2em`. Los encabezados se colorean con un tono gris oscuro que proporciona jerarquía visual sin resultar estridente.

Finalmente, una vez el contenido ha sido formateado correctamente, el informe se exporta como un archivo PDF. Este se guarda automáticamente en el directorio correspondiente utilizando un nombre identificativo que sigue el siguiente formato: `AÑO_MES_DIA_HORA_MINUTO_SEGUNDO_NOMBRENIVEL.pdf`. Un ejemplo siguiendo esta nomenclatura sería `2025_04_18_23_15_42_Primeros_pasos.pdf`. Este sistema de nombramiento permite organizar fácilmente los informes generados y facilita su trazabilidad temporal.

### 5.5.2. Generación del Informe

Como ya se ha indicado, el contenido del informe es generado por el modelo de lenguaje a partir de toda la información recopilada durante el desarrollo del nivel.

La respuesta del modelo se devuelve en formato Markdown, que posteriormente se transforma a HTML para facilitar la generación del documento final en formato PDF. Esta decisión responde a varias razones.

Por un lado, el Markdown es mucho más legible y manejable para los desarrolladores humanos, lo que facilita el proceso de depuración. Por otro, delegar la conversión a HTML en una librería especializada (en este caso, la librería `markdown` de Python) ayuda a evitar errores comunes que podrían surgir si el modelo intentara generar directamente código HTML, como etiquetas mal cerradas o estructuras mal anidadas.

Una vez transformado el contenido, se le añade una cabecera con estilos CSS personalizados que definen aspectos como la fuente tipográfica, el espaciado interno, el color de los títulos o el formato de bloques de código y citas, tal como se detalla en la sección anterior.

Finalmente, se utiliza la librería `pdfkit` para convertir el HTML resultante en un archivo PDF. Esta librería depende internamente de `wkhtmltopdf`, una herramienta externa que también debe estar instalada previamente en el dispositivo. Para su correcto funcionamiento, es necesario especificar en la configuración de `pdfkit` la ruta exacta en la que se encuentra instalado `wkhtmltopdf`.

Una vez configurado el entorno, `pdfkit` se encarga de guardar automáticamente el documento generado en la carpeta `./output-reports/`, utilizando como nombre

de archivo un identificador único que combina la fecha y hora de generación con el título del nivel correspondiente, tal y como se explicó en la Sección 5.5.1.

Puede consultarse un informe real en el Apéndice A.

## 5.6. Mejoras en la Concurrencia

### 5.6.1. Orquestador

En versiones anteriores, el `client_listener` del servidor concurrente se encargaba directamente de ejecutar las operaciones correspondientes según el tipo de mensaje recibido. Sin embargo, este enfoque presentaba ciertos inconvenientes. Por ejemplo, en caso de recibir de forma simultánea un mensaje de sincronización y uno de pregunta, no era posible atender a ambos al mismo tiempo, lo que podía provocar errores inesperados.

Para resolver este problema se introdujo el orquestador, materializado en la clase `Orchestrator`, que actúa como núcleo central del sistema de tutoría multiagente. Su función principal es coordinar los distintos agentes especializados y gestionar de forma eficiente la recepción y el procesamiento de los mensajes procedentes de ADARVE.

Desde el punto de vista arquitectónico, el `Orchestrator` encapsula y mantiene referencias a las instancias compartidas de los módulos principales del tutor (`Student`, `Knowledge` y `ContextManager`), así como a los tres agentes especializados (`QuestionAgent`, `SupportAgent` y `ReportAgent`). Estos componentes se inicializan tras la recepción de un mensaje de tipo `init`, que contiene la información estática del nivel.

El funcionamiento del `Orchestrator` se basa en un patrón productor-consumidor. Cada vez que ADARVE envía un mensaje, el `client_listener` lo añade a una cola gestionada por el orquestador. Esta cola permite procesar los mensajes de forma secuencial, evitando bloqueos y reduciendo el uso de recursos cuando no hay mensajes pendientes. Por cada mensaje recibido, se invoca el método `manage_msg()`, que delega la acción en el agente correspondiente, en función del tipo de mensaje:

- **Mensaje de inicialización**. Inicializa todos los módulos del tutor con la información recibida y deja preparado el entorno para comenzar a operar.
- **Mensaje de pregunta**. Actualiza el estado del estudiante con la sincronización incluida en la pregunta y delega en el `QuestionAgent` la generación de una respuesta. Esta se envía de vuelta a ADARVE.
- **Mensaje de sincronización**. Reenvía el mensaje al `SupportAgent`, que evaluará la necesidad de realizar una intervención, en función de los umbrales establecidos.

Al igual que en la versión anterior, el sistema sigue permitiendo múltiples usuarios simultáneamente. Cada vez que se conecta un nuevo cliente al servidor, el `client_listener` crea una nueva instancia de `Orchestrator` dedicada exclusivamente a gestionar los mensajes de ese cliente.

Además, se lanzan dos hilos en segundo plano que ejecutan el método `run()` del `Orchestrator`, actuando como *workers* que permiten un procesamiento concurrente de los mensajes de un mismo usuario, tal y como se ilustra en la Figura 5.12.

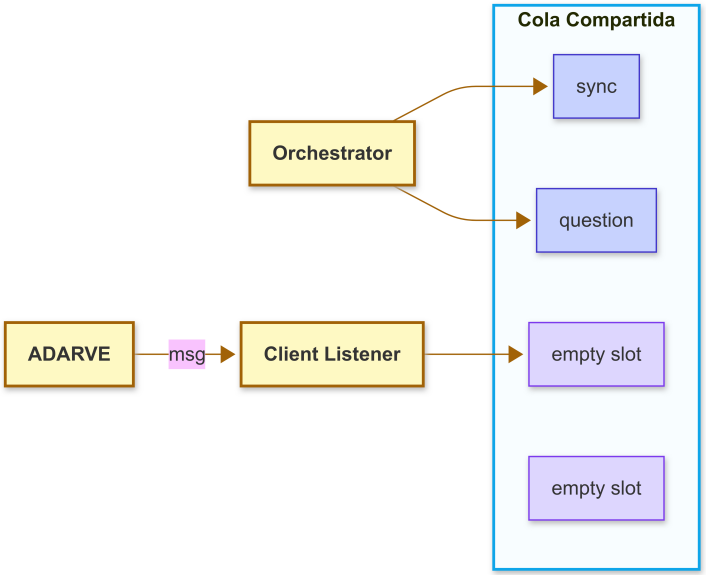


Figura 5.12: Esquema del funcionamiento de la cola compartida

La inclusión de múltiples hilos capaces de modificar estructuras de datos de forma concurrente ha requerido la incorporación de mecanismos adicionales de sincronización, que se describen en detalle en la siguiente sección.

### 5.6.2. Mecanismos de Concurrencia

Las principales estructuras que almacenan datos en el *tutor-engine* son `Student` y `Knowledge`. Debido a la existencia de múltiples hilos que acceden y modifican estas estructuras de forma concurrente, surge la necesidad de proteger su integridad para evitar condiciones de carrera y resultados inconsistentes.

Para abordar este problema, se ha decidido implementar ambas estructuras como monitores. Esto significa que cada instancia dispone de su propio cerrojo, el cual se activa automáticamente al entrar en cualquier método público de la clase y bloquea el acceso simultáneo desde otros hilos. El cerrojo se libera al finalizar la ejecución del método.

El tipo de cerrojo empleado es `threading.RLock()`, incluido en la biblioteca estándar de Python. Se ha optado por este tipo en lugar del tradicional `Lock` porque `RLock` permite que un mismo hilo adquiera el cerrojo varias veces sin provocar un bloqueo. Esta característica es esencial porque hay métodos que se llaman entre sí dentro de la misma clase.

Además, existe otra estructura crítica que debe protegerse frente al acceso concurrente, la cola de mensajes del `Orchestrator`. Esta cola es compartida por tres hilos: los dos *workers* que procesan los mensajes y el hilo principal del `client_listener` encargado de insertar nuevas peticiones.

Para gestionar esta concurrencia, se ha optado por utilizar una instancia de `queue.Queue()`, una cola incluida en la biblioteca estándar de Python diseñada específicamente para entornos multihilo. Esta cola incorpora mecanismos internos de bloqueo que garantizan que las operaciones de inserción y extracción se realicen de forma atómica, evitando condiciones de carrera y otros errores de sincronización.

## 5.7. Contenerización del Tutor con Docker

Siguiendo el principio de portabilidad que ha caracterizado a este proyecto, en esta última versión se ha optado por contenerizar el sistema de tutoría utilizando Docker. Hasta este punto, el *tutor-engine* se había ejecutado de forma local sobre una única máquina, lo que, si bien era funcional en un entorno de desarrollo, limitaba su despliegue en otras plataformas.

Una alternativa habría sido instalar manualmente todas las dependencias necesarias en cada dispositivo de destino. Sin embargo, este proceso resulta laborioso y propenso a errores, especialmente por la posibilidad de conflictos con versiones previas de bibliotecas.

Frente a esta situación, la creación de una imagen de Docker encapsula todo el entorno de ejecución requerido por el tutor, incluyendo sus dependencias, y lo hace de forma aislada del sistema anfitrión. De este modo, se garantiza que el sistema pueda ejecutarse sin necesidad de ajustes adicionales en cualquier entorno de ejecución.

### 5.7.1. Instalación de Docker

El desarrollo de este proyecto se ha llevado a cabo sobre un equipo con sistema operativo Windows 11. En este entorno, la forma más sencilla y recomendada de instalar Docker es mediante la herramienta oficial *Docker Desktop*, disponible gratuitamente para uso educativo.

La instalación comienza descargando el instalador desde el sitio oficial de Docker[1]. Una vez descargado, se deben seguir los pasos habituales del asistente de instalación. Al finalizar, es necesario iniciar sesión con una cuenta de Docker.

Una vez instalado y configurado, *Docker Desktop* proporciona una interfaz gráfica desde la que se pueden gestionar las imágenes, contenedores y redes, además de incluir todas las herramientas necesarias para trabajar desde línea de comandos, como `docker build` o `docker run`.

### 5.7.2. Creación de la Imagen. Dockerfile

Para llevar a cabo la contenerización del tutor, se diseñó un archivo `Dockerfile` que define, paso a paso, el proceso necesario para construir una imagen funcional, ligera y portátil. Esta imagen encapsula todo el entorno de ejecución necesario, garantizando que el tutor pueda funcionar en cualquier dispositivo que tenga instalado Docker, sin necesidad de realizar configuraciones adicionales. A continuación, se describe detalladamente el contenido del `Dockerfile`.

[1] https://www.docker.com/products/docker-desktop/

```
FROM python:3.11-slim AS base
LABEL maintainer="Leonardo Macías Sánchez"
```

El `Dockerfile` comienza utilizando como base una imagen oficial de Python, concretamente la versión `3.11-slim`, que ofrece una distribución reducida y optimizada del lenguaje, lo que ayuda a minimizar el tamaño final de la imagen.

```
FROM base AS builder
WORKDIR /install
COPY ../requirements /install/requirements

#Instalo las librerias requeridas
RUN pip install --prefix=/install -r requirements/requirements.txt
```

A partir de esta base se construyen dos etapas diferenciadas. En primer lugar, se define una etapa intermedia llamada `builder`, que tiene como único propósito instalar todas las dependencias del sistema. Para ello, se parte de un archivo `requirements.txt` que lista todas las librerías externas de Python requeridas por el sistema. Este fichero se copia al contenedor y se utiliza para instalar los paquetes en una ruta aislada, sin contaminar la imagen final con archivos temporales. El contenido de `requirements.txt` es el siguiente:

pdfkit==1.0.0

markdown==3.6

openai==1.72.0

```
FROM base
#Para que python no genere archivos de cache (pyc) en el contenedor
ENV PYTHONDONTWRITEBYTECODE 1
#Para que los logs no tengan retardo
ENV PYTHONUNBUFFERED 1
COPY --from=builder /install /usr/local
```

Una vez instaladas las dependencias, se construye la imagen definitiva, tomando de nuevo la imagen base y copiando únicamente lo esencial desde la etapa anterior. Se establecen también algunas configuraciones de entorno que optimizan la ejecución: se desactiva la generación de archivos `.pyc` y se deshabilita el *buffer* de salida estándar de Python, permitiendo que los *logs* se impriman en tiempo real.

```
#Instalo wkhtmltopdf y dependencias
RUN apt-get update && apt-get install -y --no-install-recommends\
      wkhtmltopdf \
      libqt5webkit5 \
      libqt5printsupport5 \
      libqt5svg5 \
      libqt5widgets5 \
    && rm -rf /var/lib/apt/lists/*
ENV WKHTMLTOPDF_PATH=/usr/bin/wkhtmltopdf
ENV OPENAI_API_KEY=insert_key
```

Además de las librerías instaladas mediante `pip`, el tutor requiere una herramienta externa para la generación de informes en formato PDF (`wkhtmltopdf`). Esta utilidad depende de varias bibliotecas del sistema, como `libqt5webkit5`, que no vienen instaladas por defecto al estar usando la versión *slim*.

Idealmente, esta herramienta se habría incluido como una dependencia más dentro del archivo `requirements.txt`; sin embargo, no está disponible en los repositorios de `pip`, lo que impide su instalación mediante dicho gestor. Tampoco se instala en la etapa builder del `Dockerfile`, ya que las bibliotecas que requiere son numerosas y su migración a la imagen final resulta compleja.

Por este motivo, se instala directamente en la etapa final del contenedor, donde se requiere su funcionalidad. Adicionalmente, se establece la variable de entorno `WKHTMLTOPDF_PATH`, con el fin de indicar a la librería `pdfkit` la ubicación exacta del ejecutable, tal y como se explicó en la Sección 5.5.2. También se ha de especificar la variable `OPENAI_API_KEY` para habilitar el LLM, como se mencionó en la Sección 4.1.3.

```
WORKDIR /workdir
COPY ./tutor-app ./tutor-app
COPY ./entrypoint.sh ./entrypoint.sh
RUN mkdir output-reports
#Exponemos el puerto para las comunicaciones con ADARVE
EXPOSE 65431
```

El contenido del tutor se copia al contenedor, incluyendo tanto el código fuente como un script de arranque denominado `entrypoint.sh`. También se crea el directorio donde se almacenarán los informes generados. El contenedor se configura para escuchar en el puerto `65431`, que es el que se utiliza para establecer la comunicación con ADARVE.

```
#Creamos un nuevo usuario con permisos de lectura/escritura/ejecucion en
    workdir
RUN adduser -u 5678 --disabled-password --gecos "" tutoruser && chown -R
    tutoruser /workdir && chmod -R 777 /workdir
USER tutoruser
```

Como medida de seguridad, el contenedor no se ejecuta como usuario `root`. En su lugar, se crea un usuario específico llamado `tutoruser`, con los permisos necesarios para operar en el entorno de trabajo.

```
ENTRYPOINT [ "./entrypoint.sh" ]
```

Por último, el contenedor está configurado para arrancar automáticamente ejecutando el script de entrada, que lanza el servidor del tutor y lo deja listo para recibir conexiones. El contenido de `entrypoint.sh` es el siguiente:

```
#!/bin/sh

python ./tutor-app 0.0.0.0
```

### 5.7.3. Configuración de Red y Volúmenes

Aunque la imagen de Docker generada es completamente funcional por sí sola, se optó por utilizar Docker Compose para simplificar y estandarizar el proceso de ejecución del contenedor, así como para facilitar la gestión de recursos adicionales como la red local y los volúmenes compartidos.

```
services:
  app:
    build:
        context: ..
        dockerfile: docker/Dockerfile
    image: tutor
    container_name: tutor-app-container
    ports:
      - "127.0.0.1:65431:65431"
    volumes:
      - ../output-reports:/workdir/output-reports
```

#### 5.7.3.1. Redireccionamiento de puertos

El *tutor-engine* necesita exponer un puerto concreto para establecer comunicación con ADARVE. Aunque esto podría configurarse manualmente mediante el comando `docker run` con la opción `-p`, repetir dicha configuración en cada ejecución resulta poco práctico y propenso a errores.

Mediante Docker Compose, esta configuración queda definida de forma declarativa en el archivo `docker-compose.yml`, donde se especifica claramente que el puerto `65431` del contenedor debe redirigirse al mismo puerto en el host, limitado a la IP local `127.0.0.1`.

Esta configuración permite mantener la compatibilidad con ADARVE incluso cuando el tutor se ejecuta dentro de un contenedor, ya que desde el punto de vista de ADARVE, la conexión sigue realizándose a una dirección local. De este modo, se facilita tanto la ejecución en un único equipo como una futura separación entre módulos en distintos dispositivos.

#### 5.7.3.2. Volumen persistente para informes

El tutor genera informes en formato PDF que deben ser accesibles fuera del contenedor. Para ello, se ha montado un volumen compartido entre el sistema anfitrión y el contenedor, de modo que los archivos creados por el tutor dentro del directorio `/workdir/output-reports` estén disponibles directamente en el directorio `./output-reports` del entorno de desarrollo. Gracias a esta configuración, es posible consultar los informes sin necesidad de acceder manualmente al interior del contenedor.

## 5.8. Estado del Proyecto: Tercer Prototipo

En este punto, el sistema de tutoría, ahora con arquitectura multiagente, ha alcanzado su estado final de desarrollo en el marco de este trabajo. El resultado es un sistema que aborda tres ejes principales, cada uno gestionado por un agente. El estado final de los agentes se resume en la Figura 5.13.

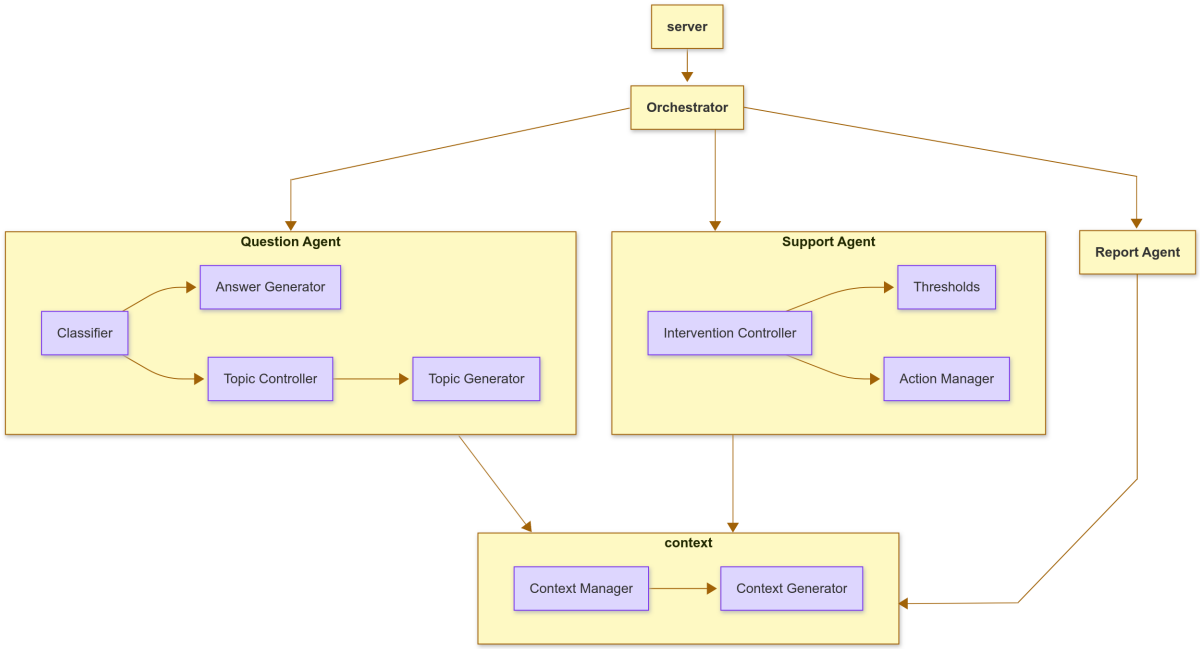


Figura 5.13: Estado final del sistema multiagente

El agente de preguntas clasifica las consultas del estudiante en función de un conjunto de temáticas predefinidas, que pueden ampliarse dinámicamente con otras que surjan a lo largo de la conversación. A partir de esta clasificación, genera un *prompt* adaptado que se utiliza para obtener una respuesta del LLM.

El agente de soporte monitoriza en tiempo real el desarrollo de la sesión mediante mensajes de sincronización y actúa proactivamente cuando detecta indicios de desorientación o inactividad. Los umbrales que determinan cuándo intervenir se ajustan a lo largo del nivel según el comportamiento del estudiante.

El agente de informes elabora un documento PDF al finalizar la sesión en el que se incluye información sobre el trascurso de la sesión de aprendizaje, como el progreso del estudiante.

La transición hacia una arquitectura multiagente ha estado acompañada de importantes mejoras en la generación de contexto, lo que ha permitido realizar múltiples llamadas al LLM en una misma interacción sin comprometer los tiempos de respuesta.

Asimismo, se han implementado mecanismos de concurrencia para asegurar el funcionamiento paralelo de los agentes sin interferencias y el sistema completo ha sido contenerizado mediante Docker.

Capítulo 6

# Conclusiones y Trabajo Futuro

A lo largo de todo un curso académico, se ha llevado a cabo el diseño e implementación de un sistema de tutoría inteligente basado en modelos extensos de lenguaje, orientado a su integración en un entorno de realidad virtual para la formación en situaciones de emergencia radiológica. El proceso ha supuesto no solo un reto técnico, sino también una oportunidad para explorar el potencial de las tecnologías actuales de inteligencia artificial aplicadas al ámbito educativo.

En las siguientes secciones se presenta, en primer lugar, el estado final del sistema a partir del análisis de los objetivos iniciales; a continuación, se exponen las principales conclusiones del trabajo y, finalmente, se proponen posibles líneas de ampliación y mejora para futuras investigaciones.

## 6.1. Análisis del Cumplimiento de los Objetivos

Este Trabajo de Fin de Grado ha consistido en el diseño e implementación de un sistema de tutoría inteligente basado en un modelo de lenguaje de gran escala, integrado en un entorno de realidad virtual de simulación de emergencias radiológicas (ADARVE).

El objetivo principal era desarrollar un sistema capaz de asistir al estudiante durante su formación sobre emergencias radiológicas, que se concreta en otros más específicos (veáse la Sección 1.2). Analicemos cada uno de ellos:

1. **Consultas**. El sistema desarrollado permite al estudiante realizar preguntas relacionadas con su proceso de aprendizaje directamente desde el entorno de realidad virtual. El agente de preguntas se encarga de analizar estas consultas, clasificarlas en una serie de temáticas predefinidas vinculadas a su formación (véase la Sección 5.3.1) y generar una respuesta adecuada (véase la Sección 5.3.3). Si la consulta no se ajusta a ninguna de las temáticas permitidas, se insta al usuario a centrarse en el nivel, cumpliendo así con el requisito de responder únicamente preguntas pertinentes.

   Las respuestas válidas se generan de forma contextualizada, ya que el agente se mantiene actualizado sobre lo que ocurre en el entorno ADARVE gracias al sistema de mensajería (véase la Sección 4.2). Además, las temáticas disponibles

se ajustan dinámicamente en función de la evolución de la conversación (véase la Sección 5.3.2).

2. **Proactividad**. El sistema incorpora un agente de soporte que analiza de forma continua la evolución del estudiante mediante mensajes de sincronización periódicos. Este agente es capaz de detectar indicadores de desorientación o inactividad, como periodos prolongados sin avance o un número elevado de preguntas fuera de contexto (véase la Sección 5.4.1), y responder con intervenciones automáticas sin ser invocado por el estudiante (véase la Sección 5.4.3). Los umbrales que desencadenan estas intervenciones se ajustan dinámicamente según el comportamiento del alumno (véase la Sección 5.4.2).

3. **Disponibilidad**. El estudiante puede consultar al tutor en cualquier momento. Para ello, únicamente debe presionar una tecla y escribir su consulta para recibir una respuesta unos segundos después por parte del tutor (véase la Sección 4.6.1).

4. **Informe**. Al finalizar la sesión, el agente de reporte genera automáticamente un informe en formato PDF con información sobre el progreso del alumno. Este informe incluye las tareas completadas, eventos destacables durante la sesión y recomendaciones personalizadas (véase la Sección 5.5).

5. **Independencia**. El tutor está desacoplado del código original de ADARVE, tanto el módulo *tutor-ADARVE* como el *tutor-engine* (véase la Sección 3.1). Además, su integración en un nivel se reduce a la inclusión de un único actor (`BP_LLM_Manager`) que se encarga de gestionar todos los elementos necesarios para su funcionamiento (véase la Sección 4.6.2). Esto permite añadir el tutor a cualquier nivel de ADARVE sin necesidad de modificar su lógica interna.

6. **Portabilidad**. Gracias a su contenerización mediante Docker (véase la Sección 5.7), el tutor puede ejecutarse en cualquier entorno compatible con esta tecnología, eliminando la necesidad de instalaciones manuales. Todo el sistema está encapsulado en una imagen que puede desplegarse fácilmente tanto en servidores locales como en la nube. Además, cuenta con parámetros configurables a través de un fichero de configuración (véanse las Secciones 4.5 y 5.4.1).

En conjunto, el sistema desarrollado cumple satisfactoriamente con todos los objetivos planteados al inicio del proyecto. Además, se ha implementado una funcionalidad adicional que no estaba contemplada originalmente.

Esta funcionalidad consiste en el uso de un servidor concurrente capaz de gestionar múltiples conexiones de estudiantes en paralelo (véase la Sección 4.4). Esto permite que, con una única ejecución del sistema y un equipo lo suficientemente potente, se puedan atender varias sesiones simultáneamente, lo cual resulta especialmente útil para entornos de formación en grupo.

En cuanto al rendimiento, los tiempos de respuesta obtenidos son muy satisfactorios, situándose en torno a los tres segundos por cada consulta del usuario. Este buen desempeño es el resultado de una arquitectura optimizada, basada en técnicas

de concurrencia, como la ejecución de múltiples *workers* y tareas en segundo plano (véase la Sección 5.6); la elección de un modelo de lenguaje adecuado (véase la Sección 4.1.2) y el uso de generadores de contexto especializados, que reducen la carga computacional (véase la Sección 5.1.2); entre otras optimizaciones desarrolladas a lo largo del proyecto.

Por último, cabe destacar que el diseño multiagente del sistema facilita su ampliación, pues nuevos agentes con funcionalidades adicionales pueden integrarse sin modificar los demás. Además, las funcionalidades existentes han sido implementadas siguiendo principios de diseño y patrones software que favorecen el mantenimiento, la extensión y la adaptación futura del sistema.

## 6.2. Conclusiones

El desarrollo de este proyecto ha sido progresivo, comenzando por establecer comunicaciones entre el tutor y ADARVE y pasando por una primera versión funcional basada en una arquitectura de tutor tradicional, hasta alcanzar una solución final estructurada mediante una arquitectura multiagente. Durante el proceso, se han adquirido numerosos aprendizajes que merecen la pena destacar:

- A través de los generadores de contexto, se ha constatado la importancia de emplear *prompts* breves y bien formulados para obtener respuestas relevantes del LLM en tiempos de respuesta reducidos (véase la Sección 5.1.1).
- La concurrencia y el paralelismo han resultado ser elementos fundamentales para ofrecer una experiencia fluida en sistemas de tutoría interactivos en tiempo real. Por ejemplo, gracias a las mejoras detalladas en la Sección 5.6, se evitan demoras si un usuario realiza una consulta al tutor a la vez que se está procesando un mensaje de sincronización.
- Utilizar un LLM como núcleo de procesamiento de lenguaje ha abierto nuevas posibilidades en el diseño de software inteligente, demostrando que es viable construir programas centrados en la interacción continua con este tipo de modelos.
- La incorporación de un sistema de tutoría basado en LLMs dentro de un entorno de realidad virtual ha demostrado ser técnicamente viable y refuerza el potencial de estas tecnologías para enriquecer experiencias educativas inmersivas.
- La arquitectura multiagente resulta especialmente adecuada para sistemas de tutoría, ya que permite abordar una misma sesión de aprendizaje desde múltiples perspectivas, asignando a cada agente una función específica dentro del proceso.
- Resulta sorprendente que, a pesar de su potencia y popularidad, Unreal Engine presente ciertas limitaciones importantes para este tipo de desarrollos, como errores en la generación de archivos internos (véase la Sección 4.2.3.2) y la

falta de herramientas integradas para la comunicación con otros sistemas, lo que obliga a depender de soluciones de terceros (véase la Sección 3.2.1).

## 6.3. Futuras Líneas de Investigación

El presente trabajo ha abordado un amplio abanico de aspectos relacionados con la integración de un sistema de tutoría inteligente basado en LLMs dentro de un entorno de realidad virtual. No obstante, aún quedan abiertas diversas vías de mejora e investigación futura que permitirían enriquecer el sistema desarrollado.

Una de las líneas más relevantes sería la validación empírica del sistema mediante pruebas reales con miembros de los cuerpos de seguridad del Estado e instructores especializados en emergencias radiológicas. Hasta el momento, las pruebas se han limitado al equipo de desarrollo, por lo que la evaluación en escenarios reales permitiría obtener un *feedback* valioso.

Más allá de realizar estas pruebas, existen mejoras técnicas que podrían implementarse. Uno de los principales desafíos ha sido la extracción de información desde el entorno ADARVE. Aunque se ha conseguido recopilar bastante información, existen datos, como las interacciones del estudiante con los objetos del entorno, que podrían usarse para mejorar el conocimiento del tutor. Esta información se almacena en la base de datos interna de ADARVE, por lo que podría ser muy conveniente considerar su uso como fuente complementaria para obtener datos, especialmente después identificar en este trabajo el bug que impedía su uso (véase la Sección 4.2.3.2).

En la misma línea, desarrollar una base de datos persistente que almacene los datos recopilados en cada sesión de aprendizaje permitiría identificar perfiles de usuario y personalizar aún más el sistema de tutoría empleando dicha información. Así, cada nueva sesión contribuiría a la mejora continua del sistema.

Desde el punto de vista funcional, otra ampliación interesante sería la incorporación de nuevas acciones dentro del agente de soporte. Por ejemplo, Tuddy podría desplazarse por el escenario para guiar físicamente al estudiante o intervenir mediante señales más sutiles como sonidos.

Finalmente, dado que el entorno se ejecuta en realidad virtual, una mejora significativa sería permitir la interacción por voz, evitando la necesidad de escribir mediante teclado, lo cual puede resultar incómodo si se realiza el entrenamiento con gafas de realidad aumentada.

# Introduction

*"If we teach today's students as we taught yesterday's, we rob them of tomorrow"*
— John Dewey

## Motivation

From our early years through to adulthood, as humans we are constantly engaged in a process of learning knowledge and skills that allow us to face a constantly changing world. This dynamic environment, in line with scientific and technological advances, forces us to continually rethink the way we learn and encourages the development of new learning techniques designed to address today's challenges.

Virtual Reality (VR) stands out among these innovative technologies by providing the opportunity to train students in situations that would be difficult to simulate in a traditional classroom. In some cases, recreating a real scenario could involve putting human lives at risk, making VR environments the only viable alternative for learning practical knowledge.

Radiological emergencies are a prime example, where adequate preparation is essential and any error could lead to severe consequences. In response to this challenge, the Spanish Nuclear Safety Council (CSN) is working in partnership with the Universidad Complutense de Madrid on the ADARVE project (in Spanish, *Análisis de Datos de Realidad Virtual para formación en Emergencias Radiológicas, SUBV-20/2021*), which aims to train State Security Forces in a 3D environment so they can respond effectively to these kinds of situations (CSN, 2021).

In digital education, Intelligent Tutoring Systems (ITS) are one of the most popular classic tools for adaptive learning (Sleeman and J. Brown, 1982). These systems leverage Artificial Intelligence (AI) techniques to dynamically customize the educational process to the needs and abilities of each student (Alkhatlan and Kalita, 2018).

In recent years, the field of AI has experienced a major transformation, due to advances in natural language processing and generation. At the heart of this revolution lies the development of cutting-edge Large Language Models (LLMs). These are deep learning models with a large number of parameters, trained with self-supervised learning on a vast amount of text (Wikipedia, 2025a). Thanks to their powerful data-processing and language-generation capabilities, LLMs have completely changed the

way computers interact with human language.

The results provided by these language models open up a new line of research aimed at improving learning sessions with intelligent tutors. The goal of this project is to innovate in the field of digital learning by enhancing the performance of ITS through the integration of LLMs and enabling their inclusion into ADARVE's VR environment.

## Objectives

The purpose of this thesis is to develop an intelligent tutor based on Large Language Models (LLMs) integrated into ADARVE's 3D environment, aimed at training the Spanish State Security Forces in the procedures to follow in the event of a radiological emergency.

The fulfillment of this goal translates into the following requirements for the tutor:

1. **Queries**. The system must be capable of receiving questions from students related to their training (i.e., the nuclear field and the virtual reality environment) and providing coherent and appropriate answers according to each student's learning context. Furthermore, it must restrict responses strictly to topics relevant to the learning process; in other words, it should not allow unrelated or off-topic queries.

2. **Proactive behaviour**. The system must include a proactive assistance component that dynamically adapts to the student's level of knowledge. It should be capable of detecting situations of disorientation or inactivity and responding accordingly.

3. **Availability**. The tutor must be available at any time within the VR environment.

4. **Report**. It should generate an evaluation report with relevant information about the student progress.

5. **Independence**. The system must operate independently from the rest of the ADARVE project and be easy to integrate in any level without requiring significant changes.

6. **Portability**. The system must be compatible with diverse execution environments and easily configurable without the need for extensive modifications.

## Work Plan

This section outlines the methodological plan to be followed throughout the development of the project until the intended objectives are achieved. The design of the intelligent tutor is conceived as an iterative process, structured around three successive prototypes:

- **Prototype 1 (P1)**. This prototype focuses on demonstrating the technical feasibility of integrating an intelligent tutor within ADARVE's 3D environment.
- **Prototype 2 (P2)**. The goal is to integrate all the technologies required for the project and use them to enable a functional dialogue system with the student.
- **Prototype 3 (P3)**. This is the final version of the tutor, fully equipped with all planned functionalities.

The following describes the planned development stages and corresponding prototypes of the project. It should be noted that implemented features may be subject to revision and improvement in later stages:

1. **Connectivity test (P1)**. An initial version will be developed to ensure proper connectivity between ADARVE and the tutor's logic. Given the use of different technologies, verifying the bidirectional transmission of data is essential to ensure the system's operability.
2. **ADARVE's adaptation (P1)**. Once data transmission is ensured, it will be verified that the user can send text messages from ADARVE and that the responses generated by the tutor's logic are displayed correctly. This step is important because ADARVE is developed by an independent team, and it is necessary to confirm that the platform includes the required tools to support this integration.
3. **Data extraction from ADARVE (P2)**. The tutor must be able to collect data from the VR environment, as well as from the user and their progress, in order to provide contextualized responses and personalized guidance.
4. **Selection of an LLM model (P2)**. A comparative analysis of the available language models will be conducted to identify the most suitable option based on the project's requirements.
5. **Development of the dialog system (P2)**. With data retrieved from ADARVE, a selected LLM model, and the interaction interface integrated, an dialog system capable of effectively responding to student questions will be developed.
6. **Pedagogical research (P2)**. Pedagogical theories and intelligent tutoring approaches will be analyzed to adapt the assistance system to the users' learning needs and enhance the overall educational experience.
7. **Integration of the adaptive assistance system (P3)**. An autonomous assistance system will be implemented based on the pedagogical insights gained during the research, adapting its responses to the user's level and progress.
8. **Evaluation report generation (P3)**. Finally, tracking data from the VR environment and the student's completed learning goals will be compiled into a session-based performance report.

Furthermore, as progress was made in the development of each prototype, this document was written in parallel, recording achievements, technical decisions and the adjustments made at each stage of the project.

## Structure of the Document

This document is divided into six chapters, structured to guide the reader from the theoretical foundations to the technical aspects of the project's development and final conclusions.

- **Chapter 1: Introduction**. It is the current chapter. It presents the objectives of the work, the methodology followed to achieve them and the motivation behind the project.
- **Chapter 2: State of the art**. This chapter reviews the most recent advances in the field of artificial intelligence applied to natural language processing. It then introduces key pedagogical theories and discusses how they are implemented in computer science through intelligent tutoring systems. Finally, a brief description of the ADARVE project is provided.

After these two introductory chapters, the document focuses on describing in detail the different development phases of the project. For ease of understanding, the process has been divided into three distinct chapters, each focusing on one of the system prototypes.

Each stage of development is presented in chronological order, highlighting the main challenges encountered, the technical decisions made and the progress achieved in the system.

- **Chapter 3: Validation of the connectivity between ADARVE and the tutor**. The design and development of a first functional model capable of establishing communication between ADARVE and the basic logic of the tutor is described.
- **Chapter 4: Dialogue Engine Development**. This phase involves the implementation of a first working version of the system, enabling students to send queries from within ADARVE and receive contextually appropriate responses from the intelligent tutor.
- **Chapter 5: Multiagent tutoring system**. The system evolves toward a more advanced agent-based architecture. The tutor enhances its question-answering capabilities, acquires autonomous intervention functions and generates personalized reports based on student progress. Moreover, its portability is improved through containerization.

Finally, there is a bibliography section that brings together all the sources used throughout the work and an appendix with a real report generated by the tutor in the simulation of a learning session.

# Conclusions and Future Work

Throughout an entire academic year, the design and implementation of an intelligent tutoring system based on large language models has been carried out, with the goal of integrating it into a VR environment for training in radiological emergency scenarios. The project presented a significant technical challenge, as well as a valuable opportunity to explore the educational applications of state-of-the-art AI technologies.

The following sections first present the final state of the system in relation to the initial objectives, then outline the main conclusions drawn from the project and finally propose possible directions for future research and improvement.

## Objective Completion Analysis

This Bachelor's Thesis involved designing and implementing an intelligent tutoring system based on LLMs integrated within ADARVE's VR environment for radiological emergency simulations.

The main objective was to develop a system capable of assisting students during their training in radiological emergencies. This general goal is broken down into more specific objectives (see Section 1.2). Each of them is analyzed below:

1. **Queries**. The developed system enables students to ask questions related to their learning process directly from the virtual reality environment. A query agent is responsible for analyzing these questions, classifying them into a set of predefined topics linked to the training content (see Section 5.3.1) and generating an appropriate response (see Section 5.3.3). If the query does not fit any of the allowed topics, the user is encouraged to focus on the level, thus fulfilling the requirement to answer only relevant questions.

   Valid answers are generated in a contextualised way, is kept up to date on what is happening in the ADARVE's environment thanks to the messaging system (see Section 4.2). In addition, the available topics are dynamically adjusted according to the evolution of the conversation (see Section 5.3.2).

2. **Proactive behaviour**. The system includes a support agent that constantly analyzes the student's progress via regular synchronization messages. This

agent is able to detect indicators of disorientation or inactivity, such as long periods without progress or a high number of out-of-context questions (see Section 5.4.1) and respond with automatic interventions without being invoked by the learner (see Section 5.4.3). The thresholds that trigger these interventions are dynamically adjusted according to the learner's behaviour (see Section 5.4.2).

3. **Availability**. The student can interact with the tutor at any time. To do so, they only need to press a key and submit their query, receiving an answer from the tutor a few seconds later (see Section 4.6.1).

4. **Report**. At the end of each session, the reporting agent automatically generates a PDF report detailing the learner's progress. This report includes information on completed tasks, noteworthy events during the session and customized recommendations (see Section 5.5).

5. **Independence**. The tutor is completely decoupled from the original ADARVE code, including both the *tutor-ADARVE* and *tutor-engine* modules (see Section 3.1). Furthermore, its integration within a level is streamlined, requiring only the inclusion of a single actor, the `BP_LLM_Manager`, which handles all necessary operational elements. This design allows the tutor to be integrated at any level of ADARVE without requiring changes to its internal logic (see Section 4.6.2).

6. **Portability**. Thanks to its Docker containerization (see Section 5.7), the tutor is platform-independent and can be executed in any Docker-compatible environment without requiring manual setup. The whole system comes as a single image, simplifying deployment on both local servers and cloud infrastructure. Moreover, it includes a configurable file for parameter tuning (see Sections 4.5) and 5.4.1)

Overall, the developed system satisfactorily fulfills all the initial project objectives. Additionally, an extra functionality was implemented that was not initially planned.

This added feature involves the implementation of a concurrent server capable of managing multiple student connections in parallel (see Section 4.4). This means a single system instance on a sufficiently powerful computer can manage numerous sessions simultaneously, which is especially beneficial for group training scenarios.

From a performance perspective, the system response times are highly satisfactory, averaging around three seconds per user query. This efficiency is attributed to an optimized architecture based on concurrency techniques, such as multiple background workers and tasks (see Section 5.6); the selection of a suitable language model (see Section 4.1.2) and specialized context generators that reduce computational load (see Section 5.1.2), among other optimizations developed throughout the project.

Finally, the system's multi-agent design facilitates expansion, as new agents with additional functionalities can be integrated without modifying existing ones. Moreover, all current functionalities were implemented following design principles and

software patterns that ensure the system's maintainability, extensibility and future adaptability.

# Conclusions

The project's development followed an iterative methodology, starting with the establishment of communication between the tutor and the ADARVE's environment, followed by the implementation of a functional prototype based on a traditional tutoring architecture, and ultimately evolving into a multi-agent architecture as the final solution. Throughout the process, numerous valuable lessons have been learned and are worth highlighting:

- The implementation of context generators has highlighted the significance of building concise and well-structured prompts to ensure relevant responses from the LLM while minimizing response times (see Section 5.1.1).
- Concurrency and parallelism are key elements in achieving a smooth operational state for real-time interactive tutoring systems. For instance, thanks to the improvements detailed in Section 5.6, delays are avoided when a user submits a query to the tutor while a synchronization message is being processed.
- Using an LLM as the core language processing engine has opened up new possibilities in intelligent software design, proving the viability of developing systems predicated on continuous interaction with these large models.
- The incorporation of an LLM-based tutoring system within a virtual reality environment has proven to be technically feasible and reinforces the potential of these technologies to enrich immersive educational experiences.
- Multi-agent architectures are especially appropriate for tutoring systems, as they allow a learning session to be addressed from different viewpoints, assigning distinct responsibilities to each agent.
- Despite its widespread use and powerful capabilities, Unreal Engine exhibits notable limitations in the context of this development. These include observed errors in the generation of internal project files (see Section 4.2.3.2) and the lack of built-in tools for inter-system communication, requiring the use of third-party solutions (see Section 3.2.1).

# Future Work

This work has addressed a wide range of aspects related to the integration of an intelligent tutoring system based on LLMs within a VR environment. Nevertheless, several areas remain open for improvement and future research, which could significantly enhance the capabilities of the system.

One of the most relevant lines of future work would be the empirical validation of the system through real tests involving members of the State Security Forces and

instructors specialized in radiological emergencies. To date, testing has been limited to the development team; thus, evaluation in real scenarios would provide valuable feedback.

In addition to the proposed evaluations, there are technical improvements that could be implemented. One of the main challenges encountered has been extracting information from the ADARVE's environment. While a considerable amount of information has been retrieved, specific types of data, such as student interactions with objects in the environment, could enrich the tutor's knowledge base. This data is available in ADARVE's internal database, so it may be worth considering its use as a complementary data source, particularly after identifying the bug that prevented access to it (see Section 4.2.3.2).

Likewise, building a persistent database to store session data would make it possible to identify user profiles and further tailor the tutor's behaviour. In this way, each new session would contribute to the continuous improvement of the system.

From a functional perspective, another promising extension would involve the incorporation of novel actions within the support agent. For instance, Tuddy could navigate the environment to offer physical guidance or use subtle signals like sounds to intervene.

Finally, given that the environment runs in virtual reality, a significant improvement would be to enable voice interaction. This would eliminate the reliance on keyboard input, which can be particularly inconvenient when using augmented reality headsets.

# Bibliografía


Alkhatlan, Ali y Jugal Kalita (2018). *Intelligent Tutoring Systems: A Comprehensive Historical Survey with Recent Developments*. arXiv: 1812.09628 [cs.HC]. URL: https://arxiv.org/abs/1812.09628.

Almeida, Felipe y Geraldo Xexéo (2023). *Word Embeddings: A Survey*. arXiv: 1901.09069 [cs.CL]. URL: https://arxiv.org/abs/1901.09069.

Bandy, Jack y Nicholas Vincent (2021). *Addressing "Documentation Debt" in Machine Learning Research: A Retrospective Datasheet for BookCorpus*. arXiv: 2105.05241 [cs.CL]. URL: https://arxiv.org/abs/2105.05241.

Bang, Yejin et al. (2025). *HalluLens: LLM Hallucination Benchmark*. arXiv: 2504.17550 [cs.CL]. URL: https://arxiv.org/abs/2504.17550.

Brown, Paul S. et al. (2021). *Refactoring the Whitby Intelligent Tutoring System for Clean Architecture*. arXiv: 2108.04621 [cs.PL]. URL: https://arxiv.org/abs/2108.04621.

Brown, Tom et al. (2020). «Language Models are Few-Shot Learners». En: *Advances in Neural Information Processing Systems*. Ed. por H. Larochelle et al. Vol. 33. Curran Associates, Inc., págs. 1877-1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Chen, Banghao et al. (2024). *Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review*. arXiv: 2310.14735 [cs.CL]. URL: https://arxiv.org/abs/2310.14735.

Chu, Zhendong et al. (2025). *LLM Agents for Education: Advances and Applications*. arXiv: 2503.11733 [cs.CY]. URL: https://arxiv.org/abs/2503.11733.

Clark, Kevin et al. (2020). *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. arXiv: 2003.10555 [cs.CL]. URL: https://arxiv.org/abs/2003.10555.

Cruciani, Leonardo (2022). *On the capacity of neural networks*. arXiv: 2211.07531 [cond-mat.dis-nn]. URL: https://arxiv.org/abs/2211.07531.

CSN (2021). *Línea Estratégica de I+D+i Principal: Gestión de emergencias*. URL: https://www.csn.es/documents/10182/2415586/Referencia-PR-044-2021.pdf/3c664ffe-9fd9-14de-b9d0-2045b30d8832.

Davis, Michelle L. et al. (2017). «Chapter 3 - Learning Principles in CBT». En: *The Science of Cognitive Behavioral Therapy*. Ed. por Stefan G. Hofmann y Gordon J.G. Asmundson. San Diego: Academic Press, págs. 51-76. ISBN: 978-0-

12-803457-6. DOI: https://doi.org/10.1016/B978-0-12-803457-6.00003-9. URL: https://www.sciencedirect.com/science/article/pii/B9780128034576000039.

Devlin, Jacob et al. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

Dong, Qingxiu et al. (2024). *A Survey on In-context Learning*. arXiv: 2301.00234 [cs.CL]. URL: https://arxiv.org/abs/2301.00234.

EpicGames (2025). *Unreal Engine 5.2 Documentation*. URL: https://dev.epicgames.com/documentation/es-es/unreal-engine/unreal-engine-5-2-documentation?application_version=5.2.

Gamma, Erich et al. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional. ISBN: 0201633612.

Goodfellow, Ian, Yoshua Bengio y Aaron Courville (2016). *Deep Learning*. MIT Press. URL: http://www.deeplearningbook.org.

Google (2025). *Gemini API Documentation*. URL: https://ai.google.dev/docs.

Grefenstette, Gregory (1999). «Tokenization». En: *Syntactic Wordclass Tagging*. Dordrecht: Springer Netherlands, págs. 117-133. ISBN: 978-94-015-9273-4. DOI: 10.1007/978-94-015-9273-4_9. URL: https://doi.org/10.1007/978-94-015-9273-4_9.

Hardesty, Larry (abr. de 2017). «Explained: Neural networks». En: *MIT News*. URL: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

Hendrycks, Dan et al. (2021). *Measuring Massive Multitask Language Understanding*. arXiv: 2009.03300 [cs.CY]. URL: https://arxiv.org/abs/2009.03300.

Hillocks, George (1995). *Teaching Writing as Reflective Practice*. New York, NY: Teachers College Press. ISBN: 9780807734339.

Hochreiter, Sepp y Jürgen Schmidhuber (nov. de 1997). «Long Short-Term Memory». En: *Neural Computation* 9, págs. 1735-1780. DOI: 10.1162/neco.1997.9.8.1735.

HuggingFace (2025). *Inference API Pricing*. URL: https://huggingface.co/docs/inference-providers/pricing.

Ji, JuYeong y Ravin Kumar (2024). *Gemma explained: An overview of Gemma model family architectures*. URL: https://developers.googleblog.com/en/gemma-explained-overview-gemma-model-family-architectures/.

Ji, Ziwei et al. (mar. de 2023). «Survey of Hallucination in Natural Language Generation». En: *ACM Computing Surveys* 55.12, págs. 1-38. ISSN: 1557-7341. DOI: 10.1145/3571730. URL: http://dx.doi.org/10.1145/3571730.

Kasneci, Enkelejda et al. (2023). «ChatGPT for good? On opportunities and challenges of large language models for education». En: *Learning and Individual Differences* 103, pág. 102274. ISSN: 1041-6080. DOI: https://doi.org/10.1016/j.lindif.2023.102274. URL: https://www.sciencedirect.com/science/article/pii/S1041608023000195.

Keith, Taber (sep. de 2018). *Scaffolding learning: Principles for effective teaching and the design of classroom resources*.

Lan, Zhenzhong et al. (2020). *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. arXiv: 1909.11942 [cs.CL]. URL: https://arxiv.org/abs/1909.11942.

Lee, Carol D. (1993). *Signifying as a Scaffold for Literary Interpretation: The Pedagogical Implications of an African American Discourse Genre*. NCTE Research Report. Urbana, IL: National Council of Teachers of English.

Lewis, Mike et al. (jul. de 2020). «BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension». En: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. por Dan Jurafsky et al. Online: Association for Computational Linguistics, págs. 7871-7880. DOI: 10.18653/v1/2020.acl-main.703. URL: https://aclanthology.org/2020.acl-main.703/.

Li, Qiaomu et al. (2024). «EduMAS: A Novel LLM-Powered Multi-Agent Framework for Educational Support». En: *2024 IEEE International Conference on Big Data (BigData)*, págs. 8309-8316. DOI: 10.1109/BigData62323.2024.10826103.

Lieb, Anna y Toshali Goel (2024). «Student Interaction with NewtBot: An LLM-as-tutor Chatbot for Secondary Physics Education». En: *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. CHI EA '24. Honolulu, HI, USA: Association for Computing Machinery. ISBN: 9798400703317. DOI: 10.1145/3613905.3647957. URL: https://doi.org/10.1145/3613905.3647957.

Liu, Bang et al. (2025). *Advances and Challenges in Foundation Agents: From Brain-Inspired Intelligence to Evolutionary, Collaborative, and Safe Systems*. arXiv: 2504.01990 [cs.AI]. URL: https://arxiv.org/abs/2504.01990.

Lo, Leo S. (2023). «The CLEAR path: A framework for enhancing information literacy through prompt engineering». En: *The Journal of Academic Librarianship* 49.4, pág. 102720. ISSN: 0099-1333. DOI: https://doi.org/10.1016/j.acalib.2023.102720. URL: https://www.sciencedirect.com/science/article/pii/S0099133323000599.

Meta (2025). *Centro de ayuda de Meta – Quest*. URL: https://www.meta.com/es-es/help/quest/1994971530885728/.

MetaAI (2024). *LlaMA Documentation*. URL: https://www.llama.com/docs/overview/.

Molina, Ismael Villegas et al. (2024). *Leveraging LLM Tutoring Systems for Non-Native English Speakers in Introductory CS Courses*. arXiv: 2411.02725 [cs.HC]. URL: https://arxiv.org/abs/2411.02725.

Murray, Tom (1999). «Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art». En: *International Journal of Artificial Intelligence in Education (IJAIED)* 10. Part II of the Special Issue on Authoring Systems for Intelligent Tutoring Systems (Editors: Tom Murray and Stephen Blessing), págs. 98-129.

Naveed, Humza et al. (2024). *A Comprehensive Overview of Large Language Models*. arXiv: 2307.06435 [cs.CL]. URL: https://arxiv.org/abs/2307.06435.

Nielsen, Michael A. (2015). *Neural Networks and Deep Learning*. Determination Press. URL: http://neuralnetworksanddeeplearning.com/.

Nwana, Hyacinth S. (1990). «Intelligent Tutoring Systems: An Overview». En: *Artificial Intelligence Review* 4.4, págs. 251-277. DOI: 10.1007/bf00168958.

OpenAI (2025). *OpenAI API Platform*. URL: https://platform.openai.com/.

OpenAI, : et al. (2024). *GPT-4o System Card*. arXiv: 2410.21276 [cs.CL]. URL: https://arxiv.org/abs/2410.21276.

OpenAI, Josh Achiam et al. (2024). *GPT-4 Technical Report*. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

Piaget, Jean (1964). «Part I: Cognitive development in children: Piaget development and learning». En: *Journal of Research in Science Teaching* 2.3, págs. 176-186. DOI: https://doi.org/10.1002/tea.3660020306. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/tea.3660020306. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/tea.3660020306.

Radford, Alec, Karthik Narasimhan et al. (2018). «Improving language understanding by generative pre-training». En: *OpenAI Blog*. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Radford, Alec, Jeff Wu et al. (2019). «Language Models are Unsupervised Multitask Learners». En: *OpenAI Blog*. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Raffel, Colin et al. (2020). «Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer». En: *Journal of Machine Learning Research* 21.140, págs. 1-67. URL: https://arxiv.org/abs/1910.10683.

Russell, Stuart y Peter Norvig (2022). *Artificial Intelligence: A Modern Approach*. 4rd. Pearson Education. ISBN: 9781292401133.

Siddiqui, Momin N. et al. (2024). *HTN-Based Tutors: A New Intelligent Tutoring Framework Based on Hierarchical Task Networks*. ACM. URL: https://arxiv.org/abs/2405.14716.

Sleeman, D. y J.S. Brown (1982). *Intelligent Tutoring Systems*. Computers and people series. Academic Press. ISBN: 9780126486803. URL: https://books.google.es/books?id=pjqcAAAAMAAJ.

Smagorinsky, Peter (2018). «Deconflating the ZPD and instructional scaffolding: Retranslating and reconceiving the zone of proximal development as the zone of next development». En: *Learning, Culture and Social Interaction* 16, págs. 70-75. ISSN: 2210-6561. DOI: https://doi.org/10.1016/j.lcsi.2017.10.009. URL: https://www.sciencedirect.com/science/article/pii/S2210656117301794.

Stamper, John, Ruiwei Xiao y Xinying Hou (2024). *Enhancing LLM-Based Feedback: Insights from Intelligent Tutoring Systems and the Learning Sciences*. arXiv: 2405.04645 [cs.HC]. URL: https://arxiv.org/abs/2405.04645.

Sun, Edward y LeAnn Tai (2025). «MultiTutor: Collaborative LLM Agents for Multimodal Student Support». En: *Proceedings of Machine Learning Research* 273.174, pág. 190.

Sweller, John (1988). «Cognitive Load During Problem Solving: Effects on Learning». En: *Cognitive Science* 12.2, págs. 257-285. DOI: https://doi.org/10.1207/s15516709cog1202_4. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog1202_4. URL: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1202_4.

Sweller, John, Jeroen J. G. van Merrienboer y Fred G. W. C. Paas (1998). «Cognitive Architecture and Instructional Design». En: *Educational Psychology Review* 10.3, págs. 251-296. ISSN: 1573-336X. DOI: 10.1023/A:1022193728205. URL: https://doi.org/10.1023/A:1022193728205.

Talebirad, Yashar y Amirhossein Nadiri (2023). *Multi-Agent Collaboration: Harnessing the Power of Intelligent LLM Agents*. arXiv: 2306.03314 [cs.AI]. URL: https://arxiv.org/abs/2306.03314.

Tunstall, Lewis, Leandro von Werra y Thomas Wolf (2022). *Natural Language Processing with Transformers, Revised Edition*. O'Reilly Media. ISBN: 9781098136796.

Udemy (2021). *An Introduction to Using Vygotsky Scaffolding in the Classroom*. URL: https://blog.udemy.com/vygotsky-scaffolding/.

Vaswani, Ashish et al. (2023). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

Vicente, Marta (2024). *Desarrollo de un chatbot de apoyo a la terapia basada en reminiscencia*. URL: https://docta.ucm.es/entities/publication/4e5a4227-dc9e-4f0f-a3c1-ce1ae1e95abe.

Vygotsky, L. S. (1978). *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press. ISBN: 9780674576285. URL: http://www.jstor.org/stable/j.ctvjf9vz4.

Wang, Alex, Yada Pruksachatkun et al. (2019). «SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems». En: *Advances in Neural Information Processing Systems*. Ed. por H. Wallach et al. Vol. 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/4496bf24afe7fab6f046bf4923da8de6-Paper.pdf.

Wang, Alex, Amanpreet Singh et al. (2019). *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. arXiv: 1804.07461 [cs.CL]. URL: https://arxiv.org/abs/1804.07461.

Wenger, Etienne (1987). *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 9781483207681.

White, Jules et al. (2023). *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT*. arXiv: 2302.11382 [cs.SE]. URL: https://arxiv.org/abs/2302.11382.

Wikipedia (2025a). *Modelo extenso de lenguaje*. URL: https://es.wikipedia.org/wiki/Modelo%5C_extenso%5C_de%5C_lenguaje.

— (2025b). *Zone of proximal development. Scaffolding*. URL: https://en.wikipedia.org/wiki/Zone%5C_of%5C_proximal%5C_development%5C#Scaffolding.

Wolf, Thomas et al. (2020). *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. arXiv: 1910.03771 [cs.CL]. URL: https://arxiv.org/abs/1910.03771.

Xu, Ziwei, Sanjay Jain y Mohan Kankanhalli (2025). *Hallucination is Inevitable: An Innate Limitation of Large Language Models*. arXiv: 2401.11817 [cs.CL]. URL: https://arxiv.org/abs/2401.11817.

Xue, Linting et al. (2021). *mT5: A massively multilingual pre-trained text-to-text transformer*. arXiv: 2010.11934 [cs.CL]. URL: https://arxiv.org/abs/2010.11934.

Yang, Jingfeng et al. (2023). *Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond*. arXiv: 2304.13712 [cs.CL]. URL: https://arxiv.org/abs/2304.13712.

Zhong, Tianyang et al. (2024). *Opportunities and Challenges of Large Language Models for Low-Resource Languages in Humanities Research*. arXiv: 2412.04497 [cs.CL]. URL: https://arxiv.org/abs/2412.04497.

Zhu, Wanrong et al. (2023). «Multimodal C4: An Open, Billion-scale Corpus of Images Interleaved with Text». En: *Advances in Neural Information Processing Systems*. Ed. por A. Oh et al. Vol. 36. Curran Associates, Inc., págs. 8958-8974. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/1c6bed78d3813886d3d72595dbecb80b-Paper-Datasets_and_Benchmarks.pdf.

Apéndice A

# Informe generado por el tutor

En la siguiente página se muestra un ejemplo de un informe creado por el tutor al finalizar la simulación de una sesión de aprendizaje. En dicho entrenamiento, el estudiante solo completó las dos primeras tareas y realizó una pregunta.

# Informe de Desempeño del Estudiante en el Nivel 'Primeros pasos'

## Descripción del Nivel

El nivel 'Primeros pasos' está diseñado para familiarizar al estudiante con las funcionalidades básicas de la realidad virtual. Durante este nivel, el estudiante debe aprender a interactuar con el entorno y realizar tareas iniciales que son fundamentales para su progreso en situaciones de emergencia radiológica.

## Progreso del Estudiante

El estudiante ha completado las siguientes tareas:

1. **Sal de la habitación**: Completado en el instante **2025-04-19 01:14:36.759000**.
2. **Habla con la persona de la habitación grande**: Completado en el instante **2025-04-19 01:14:46.762000**.
3. **Ordena los objetos que hay por la habitación**: Aún no completado.
4. **Informa a tu compañera sobre tu labor**: Aún no completado.
5. **Dirígete a la salida del edificio**: Aún no completado.

### Preguntas Realizadas

El estudiante ha realizado la siguiente pregunta:

- **Pregunta** (2025-04-19 01:14:55.753000): "¿Cómo puedo coger objetos?"
- **Respuesta**: "Acércate al objeto que deseas recoger. Usa el botón de interacción para recogerlo. Asegúrate de tener espacio en tu inventario."

## Eventos Destacables

- El estudiante ha mostrado iniciativa al interactuar con el entorno, completando las primeras dos tareas de manera efectiva.
- La pregunta sobre cómo recoger objetos indica que el estudiante está intentando comprender mejor las mecánicas del entorno virtual, lo cual es positivo.
- Es importante destacar que el estudiante no ha completado las tareas restantes, lo que sugiere que puede necesitar más práctica o guía en la interacción con los objetos.

## Recomendaciones

- **Práctica de Interacción**: Se sugiere que el estudiante practique la recogida de objetos en el entorno. Esto le ayudará a familiarizarse con las mecánicas de interacción y a completar las tareas pendientes.
- **Organización de Objetos**: El estudiante debería concentrarse en ordenar los objetos en la habitación, ya que esto es crucial para avanzar al siguiente nivel.
- **Comunicación**: Es recomendable que el estudiante informe a su compañera sobre su labor, lo que fomentará el trabajo en equipo y la comunicación efectiva en situaciones de emergencia.

## Conclusión

El estudiante ha demostrado un buen inicio en el nivel 'Primeros pasos', completando las primeras dos tareas con éxito. Sin embargo, es fundamental que se enfoque en completar las tareas restantes para avanzar en su entrenamiento. Con un poco más de práctica y atención a las instrucciones, el estudiante podrá mejorar su desempeño y estar mejor preparado para situaciones de emergencia radiológica.