

Trabajo de Fin de Grado

Grado en Ingeniería Informática

“Diálogos generados por IA para personajes no jugables en entornos virtuales con caso de uso en la reconstrucción histórica de La Laguna en el s. XVI”.

AI-Generated dialogs for NPCs in virtual environments with a use case in the historical reconstruction of La Laguna in the sixteenth century

Juan García Santos

D. **ISABEL SANCHEZ BERRIEL**, con N.I.F. 42.885.838-S profesora contratada doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. **FERNANDO PÉREZ NAVA**, con N.I.F. 42.091.420-V profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

CERTIFICA (N)

Que la presente memoria titulada:

“Diálogos generados por IA para personajes no jugables en entornos virtuales con caso de uso en la reconstrucción histórica de La Laguna en el s. XVI”

ha sido realizada bajo su dirección por D. **Juan García Santos**,
con N.I.F. 42.233.778-M.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 26 de mayo de 2023

Agradecimientos

A mi familia, amigos y pareja, que han estado a lo largo de toda mi carrera apoyándome en todo momento y animándome a seguir adelante.

A los profesores, y en especial a mis tutores, por su ayuda, paciencia y dedicación.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional.

Resumen

El objetivo de este trabajo es investigar sobre el uso de modelos de aprendizaje profundo generativos y otras tecnologías en el desarrollo de una interfaz multimodal para habilitar una comunicación natural entre usuario y personajes no jugables dentro de una experiencia virtual.

La interfaz implementada permite a un usuario interactuar con un personaje virtual a través de lenguaje natural y un micrófono, recibiendo una respuesta coherente en un tiempo asequible, mejorando así los sistemas de conversación respecto de aquellos implementados dentro de experiencias virtuales comerciales.

Se pretende estudiar la viabilidad de diversas alternativas a emplear en el desarrollo de los tres componentes principales de la interfaz: Traducción de voz a texto, generación de respuesta textual y traducción de texto a voz.

Finalmente, se integra la interfaz desarrollada en un proyecto real: Reconstrucción virtual de la ciudad de La Laguna en el Siglo XVI. Con esto, se estudia el rendimiento del sistema, así como la calidad de la interacción implementada.

Palabras clave: Interfaz multimodal, Experiencias virtuales, Reconstrucción virtual de La Laguna, Inteligencia artificial generativa.

Abstract

The aim of this work is to investigate the use of generative deep learning models and other technologies in the development of a multimodal interface to enable natural communication between users and non-playable characters within a virtual experience.

The implemented interface allows a user to interact with a virtual character through natural language and a microphone, receiving a coherent response in a reasonable time, thus improving conversation systems compared to those implemented in commercial virtual experiences.

The viability of various alternatives for the development of the three main components of the interface will be studied: voice-to-text translation, text response generation, and text-to-speech translation.

Finally, the developed interface is integrated into a real project: the virtual reconstruction of the city of La Laguna in the 16th century. This allows studying the system's performance as well as the quality of the implemented interaction.

Keywords: Multimodal Interface, Virtual Experiences, Virtual Reconstruction of La Laguna, Generative artificial intelligence

Índice general

Capítulo 1	Introducción.....	1
1.1	Antecedentes.....	1
1.2	Objetivos.....	3
1.3	Alcance	3
Capítulo 2	Estado del arte	5
2.1	Speech to Text	5
2.2	Text to Speech	7
2.3	Modelos de lenguaje.....	8
2.3.1	Ejecución local	9
2.3.2	Uso de APIs web	9
Capítulo 3	Desarrollo e implementación	11
3.1	Chat Bot por consola con GPT-2.....	11
3.1.1	Objetivos de este desarrollo	11
3.1.2	Procesado del Input (Tokenización y <i>Padding</i>).....	11
3.1.3	Inferencia sobre el modelo de lenguaje	13
3.1.4	Procesado de la salida (<i>Muestreo Top-K</i>)	13
3.1.5	Resultados	14
3.2	Demo Interfaz Multimodal para Google Cardboard	14
3.2.1	Objetivos de este desarrollo	14
3.2.2	Especificación general del desarrollo	15
3.2.3	Clases Text-to-Speech, Speech-to-Text y Voice Controller: interfaz de usuario	15
3.2.4	Clases NPCData y NPC: contextualización e interacción.....	16
3.2.5	Clases OpenAI, OpenAI API, RequestData, Usage y Choice: generación de diálogos	17

3.2.6	Cambios en el escenario	18
3.2.7	Resultados	18
3.3	Funcionamiento y rendimiento de Whisper en Python	24
3.3.1	Objetivos de este desarrollo	24
3.3.2	Resultados	24
3.4	Corpus de Datos para Fine-Tuning	25
3.4.1	Objetivos de este desarrollo	26
3.4.2	Creación del conjunto de fine-tuning.....	26
3.4.3	Resultados	27
3.5	Demo Interfaz Multimodal para PC.....	27
3.5.1	Objetivos de este desarrollo	27
3.5.2	Características del desarrollo.....	28
3.5.3	Cambios en la interacción con los NPC	28
3.5.4	Cambios la interfaz visual	29
3.5.5	Cambios en la API para generación de diálogos.....	30
3.5.6	Cambios en el procesado de voz a texto	30
3.5.7	Resultados	31
Capítulo 4	Conclusiones y líneas futuras	37
Capítulo 5	Summary and Conclusions.....	38
Capítulo 6	Presupuesto	39
6.1	Recursos Humanos	39
6.2	Equipamiento	40
6.3	Coste total del proyecto.....	40
Apéndice 1: Enlace al código y videos de ejemplo	41	
Repositorio de GitHub.....	41	

Índice de figuras

Figura 1.1: Interfaz de ELIZA [1].....	2
Figura 1.2: Arquitectura Transformer [3]	2
Figura 1.3: Presentación del proyecto "Reconstrucción virtual de La Laguna" [4, p. 3]	4
Figura 2.1: Esquema de posible Interfaz de diálogo multimodal	5
Figura 2.2: Arquitectura del modelo Whisper [7]	6
Figura 2.3: Mensaje de error sobrecarga API OpenAI	10
Figura 3.1 Flujo de Inferencia sobre un modelo de lenguaje	11
Figura 3.2: Esquema general del procesado de la entrada del modelo [25]	12
Figura 3.3: Visualización de las características del modelo en NetRon	13
Figura 3.4: Traza de ejecución del Chat Bot por consola	14
Figura 3.5: Diagrama de clases del proyecto Unity con la interfaz multimodal.....	15
Figura 3.6: Repositorio del paquete de TTS y STT para Unity Android	15
Figura 3.7: imagen de la escena de prueba inicial	16
Figura 3.8: Vista del Script NPC desde el editor de Unity.....	17
Figura 3.9: Imagen de la escena final en Unity	18
Figura 3.10: Captura de la ejecución de la aplicación en dispositivo Android.....	19
Figura 3.11: Descripción del NPC Manuel, guía de la experiencia virtual.....	20
Figura 3.12: Traza de ejecución script de transcripción de archivo de audio usando Whisper.....	24
Figura 3.13: Tabla comparativa del tamaño y la velocidad de los diferentes modelos Whisper [30]	25
Figura 3.14: Formato del conjunto de datos para fine-tuning de modelos de OpenAI	26
Figura 3.15: Muestra del corpus generado para fine-tuning	27
Figura 3.16: Tabla de costes de fine-tuning de modelos de OpenAI [33].....	27
Figura 3.17: Diagrama de clases de la demo de Interfaz multimodal para PC.....	28
Figura 3.18: Collider del NPC que detecta la proximidad del usuario.	29
Figura 3.19: Imagen de la interfaz visual de la demo para PC.....	29
Figura 3.20: Ejemplo de formato de los mensajes que emplea el modelo gpt-3.5- turbo	30
Figura 3.21:Código para exponer elementos del dll al código en C#	31

Figura 3.22: Código donde se definen los callbacks para iniciar la transcripción y registrar resultados por segmentos..... 31

Figura 3.23: Captura de la ejecución de la demo de la interfaz multimodal para PC 32

Índice de tablas

Tabla 6.1: Presupuesto de recursos humanos 39

Tabla 6.2: Presupuesto de equipamiento 40

Tabla 6.3: Presupuesto total del proyecto..... 40

Capítulo 1

Introducción

Este trabajo se centra en el diseño, desarrollo e integración de una interfaz multimodal que emplee diálogos generados por inteligencia artificial (IA) para personajes no jugables (NPCs) en entornos virtuales. Dicha interfaz permitirá la creación de sistemas avanzados capaces de simular interacciones conversacionales realistas y fluidas entre los usuarios y los NPCs en escenarios virtuales.

En la era actual, los videojuegos y las aplicaciones de realidad virtual han alcanzado niveles de inmersión impresionantes, brindando a los usuarios experiencias interactivas y emocionantes. Sin embargo, uno de los desafíos más significativos para mejorar aún más esta inmersión radica en la interacción con los NPCs dentro de estos entornos virtuales. A menudo, los diálogos de los NPCs se sienten predecibles, repetitivos o carentes de naturalidad, lo que puede afectar la experiencia del jugador y limitar la sensación de inmersión.

Siendo conscientes de esta problemática, el presente proyecto se propone explorar e intentar desarrollar soluciones innovadoras basadas en la IA generativa para crear diálogos convincentes y auténticos en tiempo real, proporcionando a los NPCs una capacidad de respuesta y adaptabilidad más cercana a la de un interlocutor humano.

Para lograr esto, se estudiará el uso de diferentes y diversas tecnologías que puedan llegar a intervenir en los diferentes componentes de esa posible interfaz multimodal, con el objetivo principal de aumentar la inmersión del usuario final en las experiencias virtuales donde se integren este tipo de sistemas.

1.1 Antecedentes

En los últimos años, el campo de la inteligencia artificial (IA) ha experimentado avances significativos que han permitido el desarrollo de sistemas cada vez más sofisticados y capaces de simular comportamientos humanos. Dentro de este contexto, la generación de diálogos por parte de sistemas de IA ha sido un área de investigación activa y prometedora.

Los primeros intentos de generar diálogos automatizados se remontan a la década de 1960, con sistemas como Eliza [1], cuya interfaz aparece en la figura 1.1, y que utilizaba reglas predefinidas para mantener una apariencia de conversación con los usuarios. Sin embargo, estos sistemas basados en reglas carecían de flexibilidad y no podían producir respuestas coherentes y

contextualmente relevantes.

```
Welcome to
EEEEEE LL      IIII ZZZZZZ AAAAA
EE      LL      II   ZZ   AA  AA
EEEEEE LL      II   ZZZ  AAAAAA
EE      LL      II   ZZ   AA  AA
EEEEEE LLLLLL IIII ZZZZZZ AA  AA

Eliza is a mock Rogerian psychotherapist.
The original program was described by Joseph Weizenbaum in 1966.
This implementation by Norbert Landsteiner 2005.

ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

Figura 1.1: Interfaz de ELIZA [1]

Con el avance de las técnicas de aprendizaje automático y la popularización de las redes neuronales, surgieron nuevos enfoques para la generación de diálogos. Uno de los hitos más destacados en este campo fue la introducción de los modelos de lenguaje basados en la arquitectura denominada *Transformer* [2] (Fig. 1.2), como los modelos GPT (*Generative Pre-trained Transformer*) de la empresa OpenAI. Estos modelos, entrenados en grandes cantidades de datos, lograron generar texto coherente y contextualmente relevante, y abrieron la puerta a nuevas posibilidades en la generación de diálogos por parte de sistemas de IA.

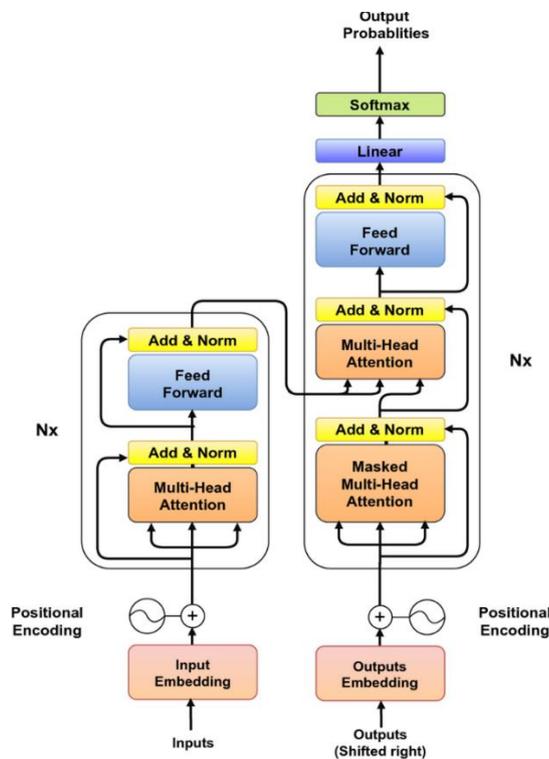


Figura 1.2: Arquitectura Transformer [3]

Sin embargo, a pesar de los avances en este campo, todavía existen desafíos significativos en la generación de diálogos. Se plantean problemas como:

- La necesidad de un hardware adecuado y costoso para ejecutar modelos computacionalmente intensivos
- La dificultad de integrar y ejecutar modelos de lenguaje en lenguajes de programación con poco o nulo soporte de librerías de redes neuronales
- La necesidad de que el modelo sea adaptable a diversos entornos de ejecución, pues la interfaz multimodal puede ser utilizada en una variedad de configuraciones; esto requiere que el modelo sea escalable, eficiente y portátil para funcionar de manera óptima en diferentes plataformas.

1.2 Objetivos

Los objetivos principales de este trabajo de fin de grado son, en primer lugar, investigar y comprender los antecedentes teóricos y prácticos relacionados con los sistemas de diálogos generados por IA, las interfaces multimodales y su aplicación en entornos virtuales.

En segundo lugar, integrar estas tecnologías en un proyecto real utilizando el motor de desarrollo de videojuegos Unity y el lenguaje de programación C#, a través del diseño e implementación de una interfaz multimodal que integre, como mínimo, modelos de lenguaje para la generación de diálogos de NPCs en tiempo real, teniendo en cuenta aspectos como la coherencia y la adaptabilidad contextual.

Por último, se llevará a cabo una evaluación del sistema desarrollado, analizando su desempeño en términos de calidad de los diálogos generados, rendimiento, dificultad de implementación y el hardware requerido.

1.3 Alcance

El alcance del proyecto abarca diversas actividades y acciones para lograr los objetivos establecidos. En primer lugar, se llevará a cabo una investigación exhaustiva para la comprensión de los antecedentes teóricos y prácticos relacionados con los sistemas de diálogos generados por IA, las interfaces multimodales y su aplicación en entornos virtuales. Esto implicará estudiar y analizar diferentes enfoques, técnicas y algoritmos utilizados en la generación de diálogos por IA.

Posteriormente, se procederá a integrar estas tecnologías en un proyecto real utilizando el motor de desarrollo de videojuegos Unity y, por requerimientos de dicho motor, el lenguaje de programación C#. Se diseñará e implementará una interfaz multimodal que permita a los usuarios comunicarse con personajes no jugables a través de su micrófono. El audio capturado será procesado y convertido a texto mediante técnicas de reconocimiento de voz. Este texto se enviará como entrada a un modelo de lenguaje que generará una respuesta adecuada. Finalmente, esta respuesta se traducirá nuevamente a audio para garantizar una experiencia inmersiva para los usuarios.

La integración del sistema desarrollado en un proyecto existente será específicamente en la reconstrucción virtual de La Laguna en el siglo XVI [4] (Fig. 1.3). Se dará vida a personajes estáticos del entorno mediante interacciones con el usuario y respuestas relevantes basadas en el contexto histórico. Esto implicará diseñar y desarrollar las interacciones necesarias para que los NPCs respondan de manera coherente y contextualizada, en línea con la temática histórica del proyecto.



Figura 1.3: Presentación del proyecto "Reconstrucción virtual de La Laguna" [4, p. 3]

Para llevar a cabo estas actividades, se probarán diferentes tecnologías, desde la ejecución local de modelos de código abierto, hasta el uso de tecnologías web o modelos alojados en la nube accesibles a través de una API como la de OpenAI. Esto permitirá evaluar la eficacia y la viabilidad de cada enfoque, considerando aspectos como la calidad de los diálogos generados, los recursos de hardware necesarios y la facilidad de implementación en lenguajes con poco soporte de librerías de redes neuronales.

Finalmente, se realizará una evaluación del sistema desarrollado, analizando su desempeño en términos de calidad de los diálogos generados, dificultad de implementación y hardware requerido. Esta evaluación se llevará a cabo mediante pruebas y análisis exhaustivos, buscando identificar posibles mejoras y áreas de oportunidad para futuras iteraciones del sistema.

Capítulo 2

Estado del arte

Siguiendo un esquema inicial de la interfaz de diálogo a desarrollar (Fig. 2.1), en este capítulo se detallará el estado del arte de los elementos más relevantes en la interfaz:

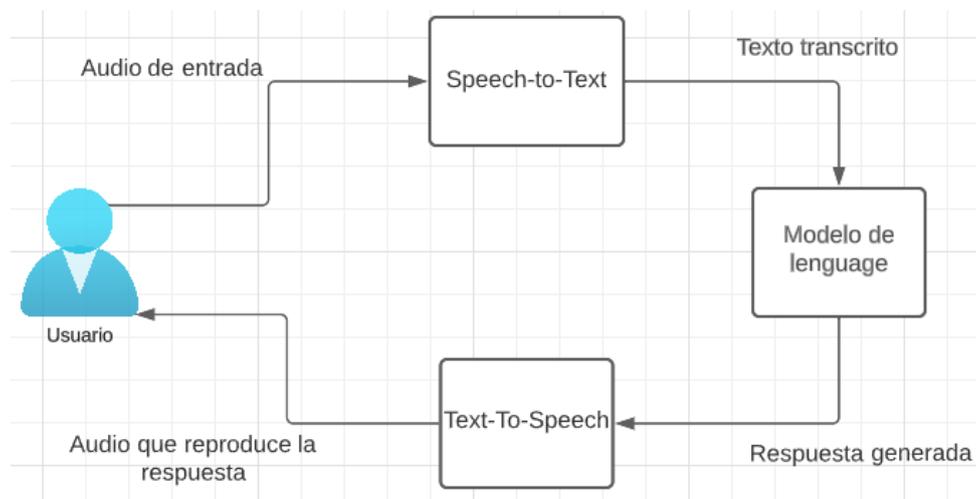


Figura 2.1: Esquema de posible Interfaz de diálogo multimodal

2.1 Speech to Text

El campo del reconocimiento de voz y conversión de voz a texto, conocido como *Speech-to-Text* (STT) o *Automatic Speech Recognition* (ASR), ha experimentado avances significativos gracias al desarrollo de técnicas de procesamiento de señales de audio y algoritmos de aprendizaje automático. Estas tecnologías permiten convertir de manera precisa y eficiente el habla humana en texto escrito, lo cual tiene aplicaciones en una amplia gama de campos, como la comunicación asistida por voz, la transcripción automática, la traducción en tiempo real y la interacción hombre-máquina.

Entre los enfoques más destacados [5] en el reconocimiento de voz, se encuentra el uso de modelos acústicos y modelos de lenguaje. Estos enfoques se han beneficiado de avances en algoritmos de aprendizaje profundo, como las redes neuronales recurrentes (RNN), tecnología que emplea *DeepSpeech* [6], las redes neuronales convolucionales (CNN) y modelos basados en la arquitectura *Transformer* como el modelo de código abierto *Whisper* de OpenAI [7].

DeepSpeech [8] es un sistema de reconocimiento automático de voz (ASR, por sus siglas en inglés), de código abierto, desarrollado por Mozilla. Utiliza técnicas de aprendizaje profundo para convertir el habla en texto. El sistema DeepSpeech se ha entrenado en grandes conjuntos de datos de voz y texto alineados, utilizando redes neuronales recurrentes que aprenden a modelar las relaciones entre las características acústicas del habla y las correspondientes transcripciones de texto. La implementación del sistema está pensada para ser ejecutada de manera local, sin dependencia de conexión a internet, y en una amplia horquilla de dispositivos, desde una raspberry Pi 4 hasta servidores de procesamiento gráfico. El mayor problema es que este proyecto ha sido abandonado [9].

En su lugar, los creadores originales de DeepSpeech desarrollaron una nueva herramienta llamada Coqui STT [10], que continuó el desarrollo donde lo había dejado DeepSpeech, mejorando los modelos y añadiendo funcionalidades. Finalmente, Coqui ha dejado de lado el Speech-to-Text para centrarse en el Text-to-Speech por varios motivos que ellos mismos comentan en su repositorio [10], entre los que mencionan el cambio de paradigma hacia nuevos modelos de Speech-to-Text como Whisper [7].

Whisper, es un modelo de reconocimiento de voz que ha mostrado excelentes resultados. El modelo utiliza técnicas de aprendizaje profundo para capturar características relevantes en las señales acústicas y realizar inferencias precisas en la transcripción del habla. Posee la capacidad de no solo transcribir audios en una gran cantidad de idiomas a texto, sino también de traducir en simultáneo audio de cualquiera de esos idiomas a texto en inglés; además, es capaz de distinguir el habla del ruido que pueda aparecer en diferentes situaciones, como en canciones o grabaciones en entornos ruidosos.

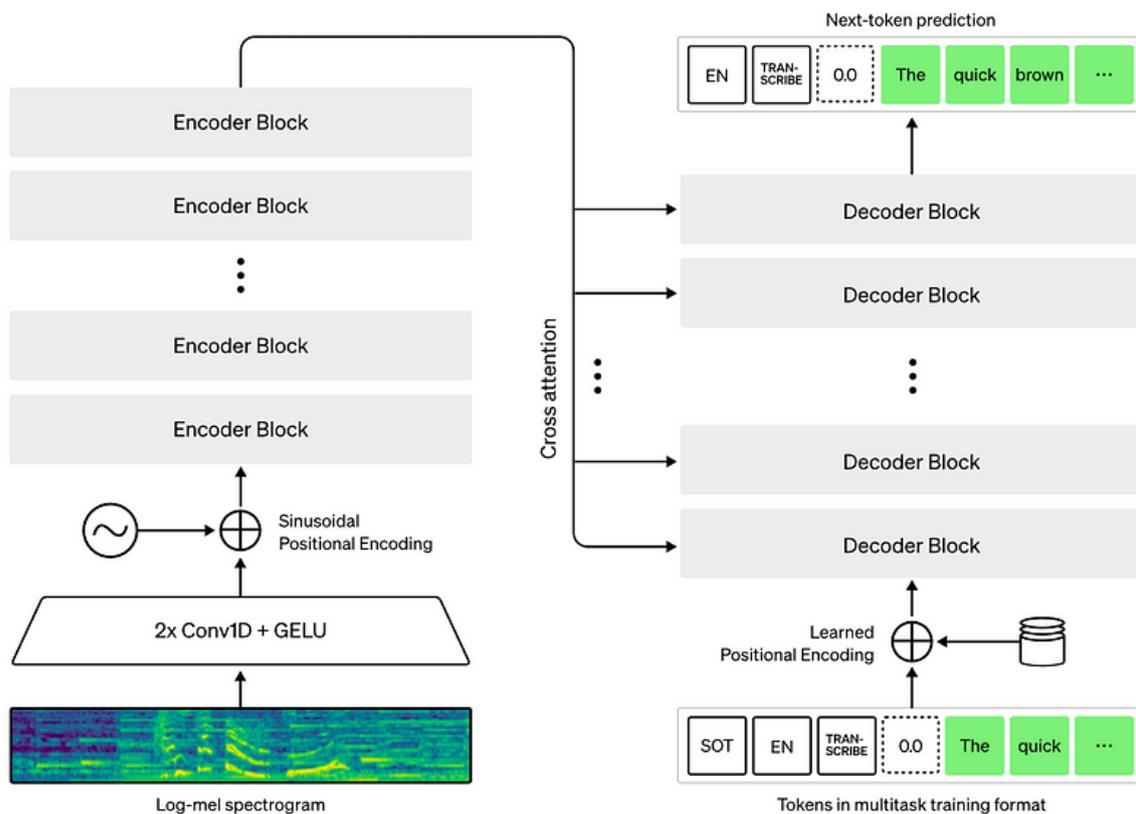


Figura 2.2: Arquitectura del modelo Whisper [7]

La arquitectura del modelo Whisper (Fig. 2.2) se basa en la arquitectura Transformer, que emplea capas de autoatención para capturar información relevante y generar representaciones contextualizadas. El modelo consta de un codificador que procesa la secuencia de entrada y un decodificador que genera la secuencia de salida utilizando las representaciones codificadas y los tokens generados previamente.

Muchas soluciones con ejecución local han adoptado el modelo Whisper no solo por su rendimiento y precisión, sino además por las ventajas que ofrece en términos de escalabilidad, disponibilidad y privacidad, permitiendo el procesamiento eficiente de grandes volúmenes de datos y el control sobre los datos sensibles. Todo esto se vio exacerbado con la implementación del modelo de forma abierta y nativa en el lenguaje C++ [11], lo que permitió obtener aumentos de rendimiento en la ejecución local cuando ya se consideraba uno de los métodos más robustos y veloces de traducción de voz a texto.

Puesto que la ejecución de estos modelos es difícilmente asumible por dispositivos con capacidad computacionalmente limitada, en este tipo de dispositivos los sistemas de transcripción de voz a texto se integran a través de servicios web. Por ejemplo, los dispositivos Android poseen y recomiendan en su documentación el uso de una librería nativa [12] que a bajo nivel emplea llamadas a una API de servicio de Google.

2.2 Text to Speech

En el campo de la síntesis de voz, se han desarrollado varios modelos de vanguardia que han mejorado significativamente la calidad y la eficiencia de la generación de voz. Dos de estos modelos destacados son Tacotron 2 y FastSpeech 2 [13].

Tacotron 2 es un modelo autorregresivo basado en redes neuronales recurrentes que tiene la capacidad de generar voz de manera secuencial. Utiliza un enfoque de atención para alinear el texto de entrada con los espectrogramas correspondientes, lo que permite una síntesis de voz más precisa y natural. Tacotron 2 ha demostrado una excelente calidad en la generación de voz, capturando matices y entonaciones con gran fidelidad. Sin embargo, debido a su naturaleza autorregresiva, la generación de voz con Tacotron 2 puede ser lenta y requerir una mayor potencia de cálculo.

Por otro lado, FastSpeech 2 es un modelo basado en *Transformer* que permite una generación de voz más rápida y eficiente. A diferencia de Tacotron 2, FastSpeech 2 no sigue un enfoque autorregresivo, sino que genera los espectrogramas de forma paralela. Esto significa que puede sintetizar la voz correspondiente al texto de entrada en tiempo real, lo que resulta en una generación de voz más rápida y menos costosa computacionalmente. FastSpeech 2 también ofrece la posibilidad de adaptar la voz sintética a diferentes estilos y características mediante técnicas de transferencia de estilo.

Ambos modelos, Tacotron 2 y FastSpeech 2, han avanzado significativamente en la síntesis de voz y han demostrado una calidad y una eficiencia notables en la generación de voz sintética. Tacotron 2 se destaca por su capacidad de capturar detalles y matices de la voz de manera precisa,

mientras que FastSpeech 2 se distingue por su velocidad de generación y su capacidad de adaptación a diferentes estilos y características. Estos avances en la síntesis de voz han abierto nuevas posibilidades en aplicaciones como asistentes virtuales, servicios de voz en línea y sistemas de accesibilidad.

Normalmente este tipo de elementos se emplean a través de APIs web, como es el caso de la librería nativa de Android que usan muchas aplicaciones para implementar esta funcionalidad en dispositivos móviles.

2.3 Modelos de lenguaje

Hasta hace no mucho el estado del arte en el procesamiento del lenguaje natural (NLP) se basaba principalmente en enfoques tradicionales como los modelos n-grama con cadenas de Markov, los clasificadores de máxima entropía (MaxEnt) y las redes neuronales recurrentes (RNN). Estos enfoques demostraron ser efectivos en muchas tareas del procesamiento del lenguaje, pero también presentaban limitaciones en cuanto a la capacidad de capturar la estructura a largo plazo de las secuencias de texto y generar una salida coherente.

Sin embargo, con la aparición de los grandes modelos de lenguaje (LLM en inglés), como GPT-3 [14], se ha producido un cambio paradigmático en el procesamiento del lenguaje natural. Estos modelos utilizan la arquitectura *Transformer*, que se basa en la atención y permiten capturar eficientemente las dependencias a larga distancia en el texto. Esto ha llevado a mejoras significativas en tareas como la generación de texto, la traducción automática, el resumen de texto y la respuesta a preguntas, entre otros.

Los LLM han sido entrenados en grandes cantidades de datos de texto y han aprendido a modelar la estructura y el significado del lenguaje de una manera más precisa y contextual. Estos modelos son capaces de generar texto coherente y humano, e incluso pueden exhibir cierto grado de creatividad en su salida. Además, su arquitectura permite la transferencia de conocimiento a través de tareas relacionadas, lo que ha llevado al desarrollo de modelos de lenguaje pre-entrenados que se pueden ajustar y afinar para tareas específicas con conjuntos de datos más pequeños, a través de una técnica conocida como *fine-tuning* [15].

Cuando se habla de la ejecución de este tipo de modelos pre-entrenados, se hace referencia al proceso de realizar inferencia sobre el modelo, es decir, darle una entrada y obtener una salida. Esa entrada se conoce como *prompt* y la salida como *completion*. Digamos que la salida viene influenciada no solo por los parámetros de configuración de cada modelo, sino también por la cadena de entrada. Al diseño y estructuración del *prompt* para obtener resultados en un formato determinado o con características específicas, se le conoce como *prompt engineering* [16]. Cuando los resultados que esperamos obtener no tienen que ver con la estructura de la salida, por ejemplo, que el modelo siempre responda con una lista numerada de elementos, sino que tienen que ver con la capacidad del propio modelo, digamos clasificar un texto de entrada como positivo o negativo, ahí en vez de emplear *prompt engineering* se tendría que realizar *fine-tuning*. Es importante realizar esta distinción pues en nuestro caso de uso hay requerimientos u objetivos de la salida del modelo para los que debemos saber cuál de las dos alternativas comentadas resulta mejor para alcanzarlos.

A la hora de plantear el desarrollo de un sistema que integre esta tecnología, existen dos vías claras:

2.3.1 Ejecución local

Los modelos ejecutables en local son aquellos que se pueden implementar y ejecutar directamente en un entorno local, lo que permite un mayor control sobre el proceso y la privacidad de los datos. Algunos ejemplos de estos modelos son GPT-2, BLOOM, BERT y Llama. Estos modelos *de código abierto* han alcanzado un rendimiento excepcional en tareas de generación de texto, comprensión del lenguaje y traducción automática. Sin embargo, debido a su complejidad y tamaño, la ejecución suele requerir un hardware potente y una configuración especializada para su implementación local.

Otra desventaja que posee esta forma de integrar los modelos de lenguaje es la fuerte dependencia de librerías con interfaces de alto nivel que faciliten al desarrollador la tarea. Esto genera una desventaja entre ciertos lenguajes que poseen una comunidad muy activa en este ámbito, como Python, frente a otros lenguajes que ni siquiera disponen de tales librerías, suponiendo un esfuerzo titánico y requiriendo un conocimiento a bajo nivel de los modelos de lenguaje y sus estructuras de datos asociadas para realizar una implementación nativa. Este es el caso que nos ocupa, puesto que debido al motor de desarrollo a emplear es Unity, el lenguaje que usaremos para la mayoría de los desarrollos es C#. En el momento de realizar este proyecto, el lenguaje C# dispone de librerías para el diseño y ejecución de modelos aprendizaje profundo, como ONNX [17] o TensorFlow.NET [18], que o están en estadios iniciales de desarrollo o todavía no disponen de interfaces de medio o alto nivel que faciliten el uso de estos nuevos modelos de lenguaje.

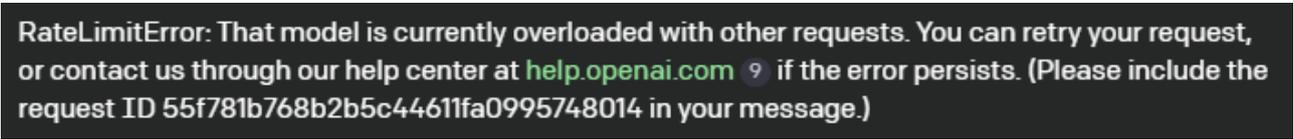
Sin embargo, en tiempos recientes, la comunidad *open source*, no solo ha creado modelos con una calidad equiparable a la de aquellos disponibles de forma comercial, como Chat-GPT [19] de la empresa OpenAI o PaLM 2 [20] de Google, sino que además han realizado implementaciones nativas en diferentes lenguajes relevantes como C++ [21], facilitando la integración de estos modelos en varios y diversos lenguajes que puedan aceptar código C++ de alguna forma, como es el caso de C# [22]. Algunos de estos modelos son Llama [21], Vicuña [23] y Koala [24].

2.3.2 Uso de APIs web

Por otro lado, existen versiones comerciales de modelos de lenguaje que son accesibles a través de API web, accesibles de forma remota a través de solicitudes HTTP. Estos modelos, alojados en la nube, ofrecen una solución conveniente y escalable para aplicaciones que requieren capacidades de procesamiento del lenguaje natural. Algunos ejemplos populares incluyen la API de OpenAI, como los modelos GPT-4 y Chat-GPT, así como otras soluciones comerciales como Google Cloud Natural Language API y Microsoft Azure Cognitive Services. Estos modelos basados en API web brindan acceso instantáneo a potentes capacidades de procesamiento y generación de lenguaje, lo que facilita su integración en aplicaciones web y móviles sin la necesidad de implementar y mantener la infraestructura localmente.

Las desventajas más relevantes en nuestro caso de uso son, no solo un costo económico asociado

al uso, sino también la dependencia respecto de una conexión de red estable y de la carga que esté sufriendo la infraestructura de servidores que empleemos. No sería aceptable que, durante una conversación, en vez de recibir una respuesta encontráramos un error que nos indicara que el servidor está bajo una gran carga y debemos volver a intentarlo más adelante (Fig. 2.3). Además, hay que tener en cuenta que la velocidad de respuesta varía en gran medida según la conexión de la que se disponga, pudiendo llegar a aumentar la latencia total de la interacción por encima de los tiempos propios de una comunicación interpersonal.



RateLimitError: That model is currently overloaded with other requests. You can retry your request, or contact us through our help center at help.openai.com ⁹ if the error persists. (Please include the request ID 55f781b768b2b5c44611fa0995748014 in your message.)

Figura 2.3: Mensaje de error sobrecarga API OpenAI

Como ventaja, hay que destacar la facilidad de integración puesto que hoy en día resulta muy sencillo realizar peticiones web a una API de forma programática; casi cualquier lenguaje dispone de una librería nativa que soporte esta funcionalidad.

Capítulo 3

Desarrollo e implementación

Durante el transcurso del proyecto, se realizaron varios desarrollos con diferentes objetivos: entender el uso y funcionamiento de los modelos de lenguaje a bajo nivel, comenzar a ver el rendimiento que se puede obtener en hardware básico, adaptar estas tecnologías a nuestro caso de uso y finalmente implementar la interfaz previamente descrita.

3.1 Chat Bot por consola con GPT-2

3.1.1 Objetivos de este desarrollo

Centrándonos en el componente del modelo de lenguaje de la interfaz planteada (Fig. 2.1), se desarrolló un Chat Bot por línea de comandos que, usando la librería ONNX [17], permitía la carga e inferencia del modelo de lenguaje de código abierto más potente en aquel momento: GPT-2.

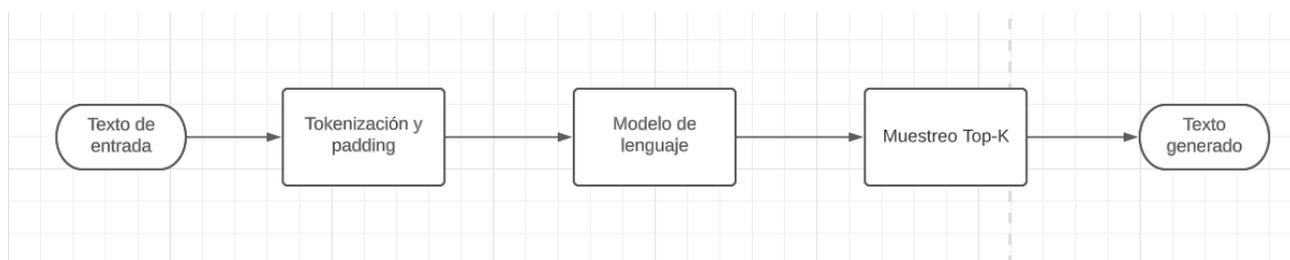


Figura 3.1 Flujo de Inferencia sobre un modelo de lenguaje

Gracias a este desarrollo pude entender que el flujo de ejecución de un módulo que integre el uso de un LLM es el descrito en la figura 3.1. A continuación, pasaré a desglosarlo en detalle:

3.1.2 Procesado del Input (Tokenización y *Padding*)

El programa comienza cargando el modelo usando la librería y recibiendo la entrada del usuario. Ese input debe ser procesado antes de darlo como entrada al modelo y ahí es donde entra el proceso de tokenización y padding, que es comúnmente utilizado en el procesamiento de lenguaje natural. Un tokenizador es una herramienta utilizada en este proceso para dividir el texto en unidades más pequeñas llamadas tokens. Los tokens pueden ser palabras individuales, subpalabras o incluso caracteres, dependiendo del nivel de tokenización requerido.

Uno de los métodos comunes de tokenización es el algoritmo BPE (Byte Pair Encoding). El algoritmo BPE se basa en la observación de que las secuencias de caracteres más comunes en un corpus de texto tienden a repetirse con frecuencia. Su objetivo es encontrar una representación eficiente de los datos reemplazando las secuencias repetidas con símbolos más cortos.

El proceso de tokenización (Fig. 3.2) con el algoritmo BPE implica los siguientes pasos: inicialización, construcción de frecuencias y fusión. En la inicialización, el corpus de texto empleado para el modelo de lenguaje se divide en caracteres o subpalabras iniciales. Luego, se construye un diccionario que mapea los pares de caracteres o subpalabras más frecuentes junto con su frecuencia en el corpus. A continuación, se fusionan los pares de caracteres o subpalabras más frecuentes y se agregan al diccionario. A todo el proceso se le conoce como entrenamiento del tokenizador y generalmente se realiza una sola vez en un corpus de entrenamiento antes de ser utilizado en tareas posteriores. Teniendo un tokenizador entrenado, al darle una cadena de entrada es capaz de devolverte la secuencia de tokens que representa a esa cadena.

Una vez que se ha realizado la tokenización, es común que los modelos de aprendizaje automático requieran que todas las secuencias de entrada tengan la misma longitud. Aquí es donde entra en juego el padding. El padding implica agregar tokens especiales llamados "tokens de relleno" al final de las secuencias más cortas para que todas tengan la misma longitud. Por ejemplo, si tenemos dos oraciones: ["Hola", "mundo"] y ["Hola"], podríamos agregar un token de relleno, como "<PAD>", al final de la segunda oración: ["Hola", "<PAD>"]. Esto permite que todas las secuencias tengan una longitud uniforme y se puedan procesar en lotes durante el entrenamiento del modelo.

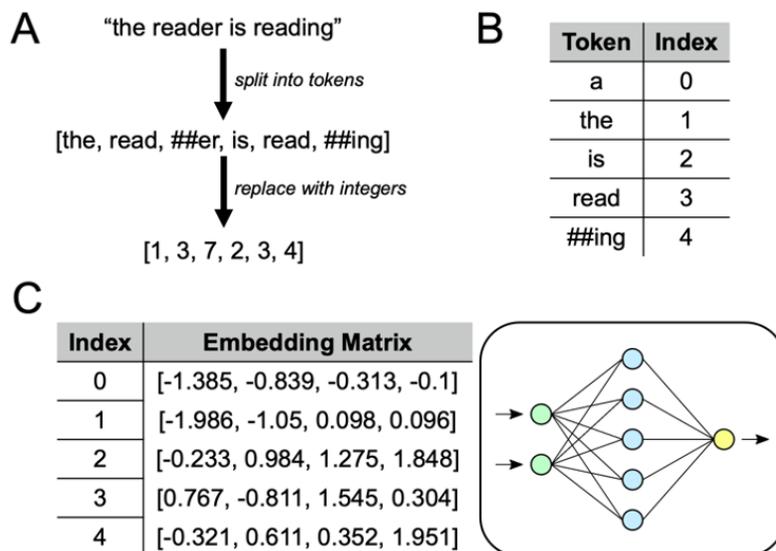


Figura 3.2: Esquema general del procesado de la entrada del modelo [25]

En nuestro caso, para realizar este proceso tuvimos que emplear un tokenizador externo a la librería ONNX puesto que esta no incluye ninguno. Se seleccionó uno implementado de forma nativa en C# por Microsoft llamado BlingFire [26]. Finalmente, los tokens se convierten en valores numéricos usando la codificación por *embeddings*, que consiste en la representación de cada token mediante un vector denso, que indica no solo a qué tipo de token hace referencia, si no

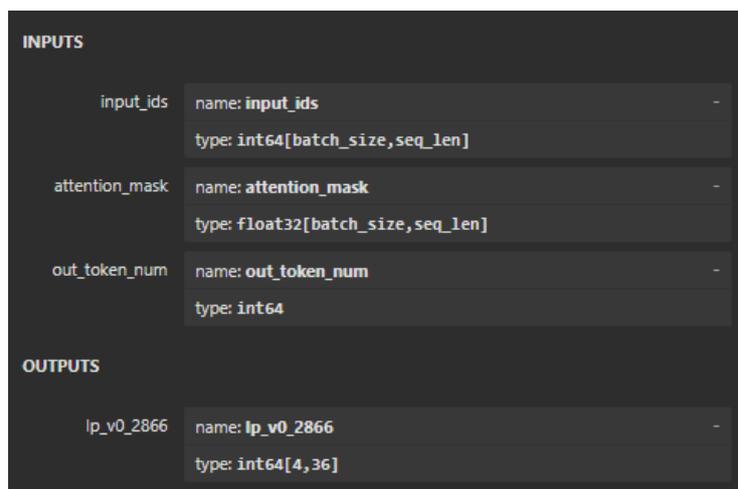
también algo de información extra, en este caso, un valor de id que es igual a la posición del token en el vocabulario.

3.1.3 Inferencia sobre el modelo de lenguaje

Una vez tenemos el input ya convertido en una representación numérica, debemos almacenarlo en la estructura de datos que emplean los modelos de aprendizaje profundo: el tensor.

Un tensor es una estructura de datos multidimensional que generaliza los conceptos de vectores y matrices a dimensiones superiores. Se utilizan para representar datos como entradas de modelo, salidas de modelo y parámetros aprendidos. En nuestro caso, cada posición almacena un token de la entrada.

Ese tensor se da como entrada al modelo y se recibe otro tensor de vuelta que almacena los ids de los tokens de la respuesta. Cada modelo posee diferentes entradas y tipos de dato de tensor diferentes. Para conocer estos valores y así poder instanciar de antemano correctamente las estructuras de datos, se suele emplear la herramienta NetRon [27].



INPUTS	
input_ids	name: input_ids type: int64[batch_size, seq_len]
attention_mask	name: attention_mask type: float32[batch_size, seq_len]
out_token_num	name: out_token_num type: int64
OUTPUTS	
lp_v0_2866	name: lp_v0_2866 type: int64[4, 36]

Figura 3.3: Visualización de las características del modelo en NetRon

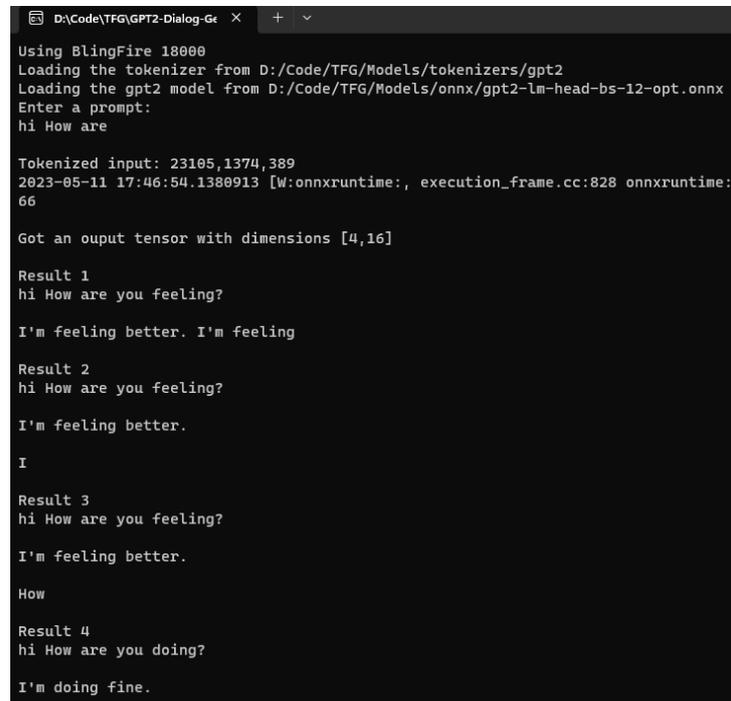
Una vez conocido el tamaño y tipo de dato del tensor de salida (Fig. 3.3), ejecutamos la inferencia del modelo usando la función disponible de la librería ONNX y obtenemos el tensor resultante.

3.1.4 Procesado de la salida (*Muestreo Top-K*)

Para procesar la salida, simplemente debemos extraer los ids de los tokens de la respuesta generada del tensor de salida y convertir cada id en el token asociado para construir la cadena resultante. En el caso del modelo empleado, está codificado de antemano para devolver un tensor que contiene 4 respuestas generadas. A este proceso de seleccionar los 4 mejores resultados se le denomina “Top-K sampling”, con “k” igual a 4 en este caso.

3.1.5 Resultados

La aplicación final generada se caracteriza por poseer una interfaz en línea de comandos (Fig. 3.4), que espera la entrada por de teclado del usuario y responde con las diferentes alternativas textuales generadas por el modelo GPT-2 ejecutado de forma local.



```
D:\Code\TFG\GPT2-Dialog-Ge x + v
Using BlingFire 18000
Loading the tokenizer from D:/Code/TFG/Models/tokenizers/gpt2
Loading the gpt2 model from D:/Code/TFG/Models/onnx/gpt2-lm-head-bs-12-opt.onnx
Enter a prompt:
hi How are
Tokenized input: 23105,1374,389
2023-05-11 17:46:54.1380913 [W:onnxruntime:, execution_frame.cc:828 onnxruntime::
66
Got an ouput tensor with dimensions [4,16]
Result 1
hi How are you feeling?
I'm feeling better. I'm feeling
Result 2
hi How are you feeling?
I'm feeling better.
I
Result 3
hi How are you feeling?
I'm feeling better.
How
Result 4
hi How are you doing?
I'm doing fine.
```

Figura 3.4: Traza de ejecución del Chat Bot por consola

La carga del modelo, que se realizaba en el arranque del programa, tardaba alrededor de 10 segundos y el proceso de inferencia, realizado en CPU por ausencia de soporte de ONNX para inferencia en GPU, tardaba unos 3 segundos en devolver unas respuestas de calidad relativa. A partir de cierto número de tokens, el modelo comenzaba a repetir palabras y sentencias en bucle. En los foros especializados no se encontró información respecto a este problema, así que se concluyó el desarrollo en ese punto y, habiendo evaluado el rendimiento del sistema, se consideró analizar otras vías para una interacción natural con los NPCs.

3.2 Demo Interfaz Multimodal para Google Cardboard

3.2.1 Objetivos de este desarrollo

Con la intención de dotar de vida a unos personajes ubicados en una reconstrucción a escala de la ciudad de La Laguna en el siglo XVI, se planteó el desarrollo de una interfaz multimodal que permitiera a un usuario interactuar de forma natural con dichos personajes, teniendo en cuenta el contexto histórico y geográfico de la experiencia virtual.

3.2.2 Especificación general del desarrollo

Como vemos en la figura 3.5, el sistema global se compone de hasta 10 clases que se pueden agrupar en 3 conjuntos diferentes:

- Interfaz de usuario
- Contextualización e interacción de los NPC
- Generación de diálogos

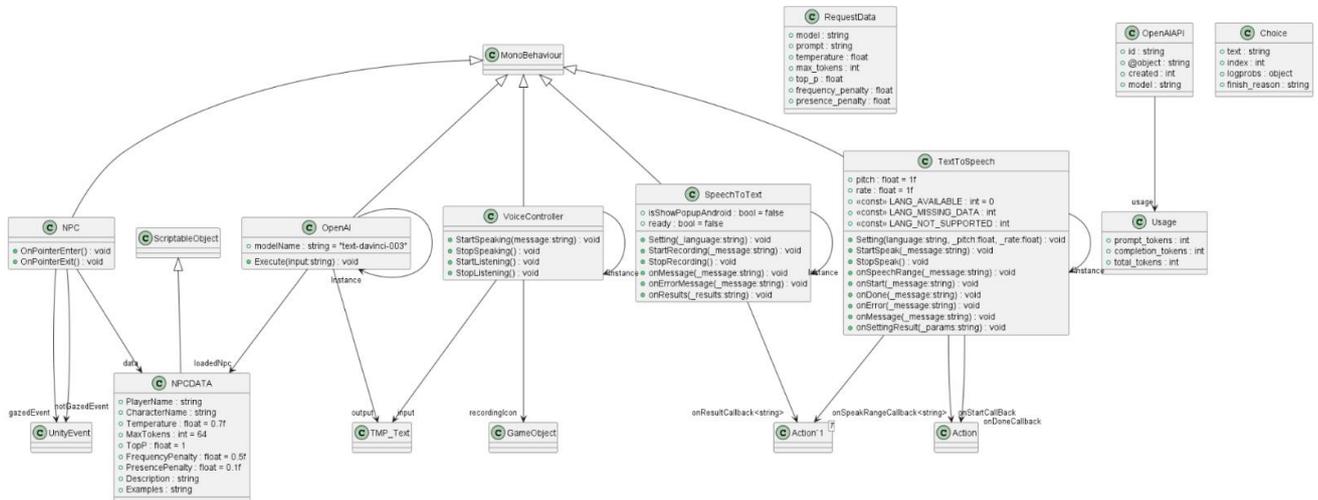


Figura 3.5: Diagrama de clases del proyecto Unity con la interfaz multimodal

3.2.3 Clases Text-to-Speech, Speech-to-Text y Voice Controller: interfaz de usuario

La interfaz multimodal comenzó su desarrollo pensado para la infraestructura de realidad virtual de Google CardBoard. Esto fue así por la facilidad que otorgaba el poder ejecutar los proyectos generados en un dispositivo móvil. Para ello, lo primero que hicimos fue configurar el proyecto de Unity e instalar el SDK de Google CardBoard.

Una vez hecho esto y teniendo claro que el proyecto se ejecutaría en dispositivos móviles, se buscó la manera de emplear las librerías nativas disponibles para tanto el Text-to-Speech como el Speech-to-Text.

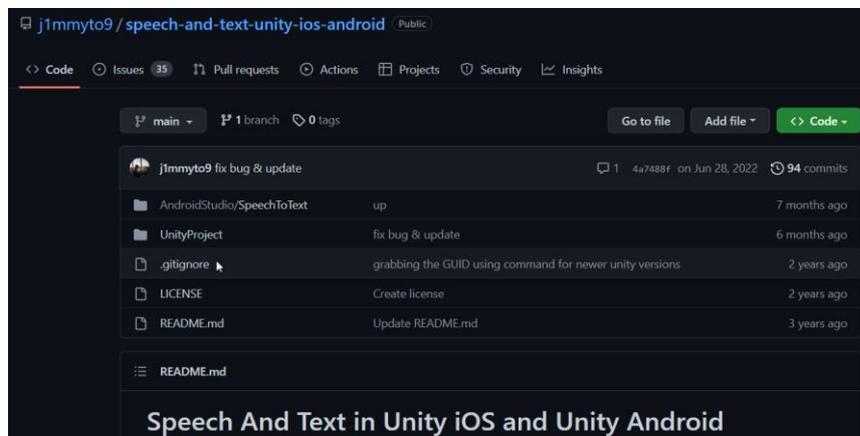


Figura 3.6: Repositorio del paquete de TTS y STT para Unity Android

Tras una búsqueda se encontró un repositorio (Fig. 3.6) que implementa unos Wrappers de la interfaz de java de las librerías nativas de Android para ser accedidas desde C#. Como además disponía de código de ejemplo, fue muy sencillo implementar un sistema que, al toque de un botón, grabara audio, mostrara en pantalla el texto sintetizado y reprodujera en el altavoz una voz leyendo ese texto. El código que maneja estos dos sistemas se engloba en dos clases, "SpeechToText" y "TextToSpeech", además de en una tercera clase, "VoiceController", que sirve de interfaz de alto nivel de ambos elementos para coordinar correctamente su uso, puesto que el código contenido dentro de las dos clases individuales contiene llamadas al plugin de Java que emplea la librería nativa de Android [28].

Como forma de otorgarle al usuario retroalimentación respecto al proceso de grabación y procesado del audio, se crearon dos mallas de texto flotantes, en medio de la escena, que se actualizaban indicando el estado de dos elementos, el procesado del audio de entrada en texto y la generación de respuesta por parte del modelo de lenguaje.

3.2.4 Clases NPCData y NPC: contextualización e interacción

Una vez integradas y probadas estas funcionalidades de la interfaz, se decidió implementar el sistema de eventos que activaría todo este proceso. Para ello, se estimó adecuado que la grabación se iniciara cuando el jugador mirara directamente al NPC, así que se montó un escenario de prueba con varios personajes (Fig. 3.7), se les añadió una malla y se definieron dos eventos: "Gazed Event" que se activa cuando la cámara apunta a un NPC, lo que realiza la llamada pertinente de iniciar la grabación y "Not Gazed Event" que se activa cuando el usuario deja de mirar al NPC, lo que activa la detención de la grabación, iniciando el procesado del audio.

Estos eventos se integraron en una clase NPC que también posee una referencia a la clase "NPCDATA"; esta se trata de un "Prefab" de un "Scriptable Object" (ambos son tipos de objetos dentro del motor de desarrollo) que es capaz de contener toda la información que define un NPC (descripción, ejemplos de conversación, ...). Esta información se carga en una variable global, durante la ejecución del primer evento, para hacerla accesible a la clase que ejecute el modelo de lenguaje. Con los detalles del NPC y algunos ejemplos de cómo debe responder, se compone un prompt que no solo contenga la entrada del usuario, sino también la información del personaje con el que está interactuando. La idea es que la respuesta generada tenga en cuenta todo el contexto de la interacción.



Figura 3.7: imagen de la escena de prueba inicial

Cualquier modelo 3D de un personaje puede valerse de la tecnología implementada para la interacción por voz. La conversión a un NPC parlante se logra a través del script NPC que requiere la instanciación del prefab que registra toda su información contextual: NPCDATA. Como consecuencia, la configuración de esta modalidad de interacción de un modelo 3D en la escena se logra adjuntándole el script NPC, creando una instancia del prefab NPCDATA con la información inicializada y enlazando en el script la referencia a NPCDATA (Fig. 3.8)

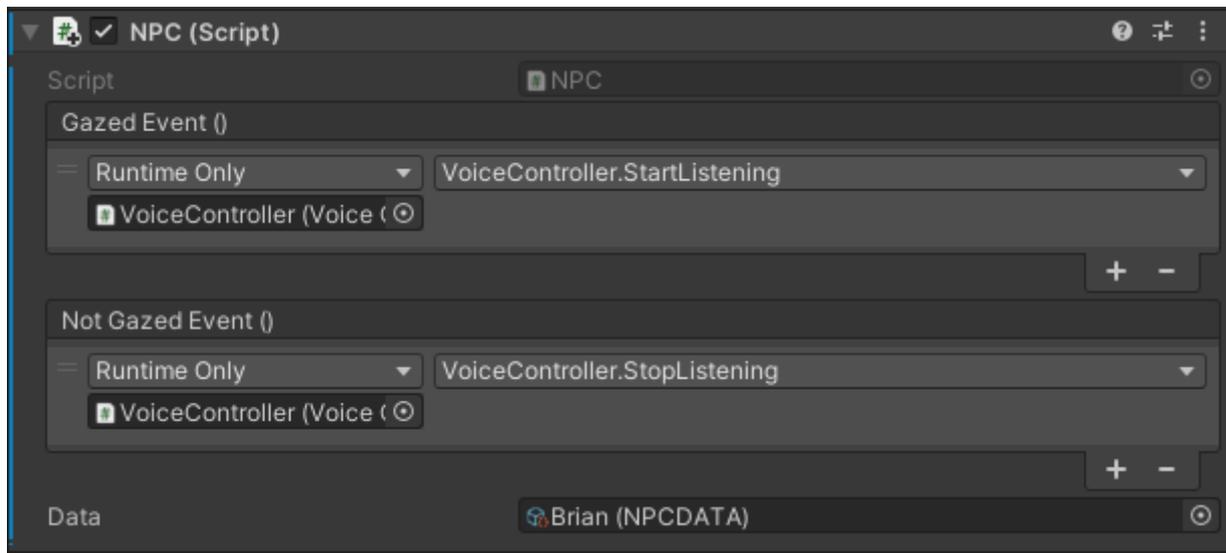


Figura 3.8: Vista del Script NPC desde el editor de Unity

3.2.5 Clases OpenAI, OpenAI API, RequestData, Usage y Choice: generación de diálogos

Respecto a la generación de texto, como esta demo tenía el objetivo de ejecutarse en dispositivos móviles, al final se optó por implementar el uso de un modelo de lenguaje a través de API web, en este caso el modelo GPT-3 de OpenAI. Para implementar su uso, se desarrollaron unas estructuras de datos siguiendo la documentación de la API [29], que almacenaban toda la información relativa a las consultas, tanto enviadas como recibidas. “RequestData” para el envío de información, “OpenAI API” para la recepción de la respuesta y las clases “Choice” y “Usage” para contener la respuesta generada en formato textual y datos de consumo de la API, respectivamente.

La clase “OpenAI” contiene los métodos necesarios para, dada una cadena de entrada, generar una petición web en el formato empleado por la API con todos los parámetros de configuración asociados al uso del modelo. En respuesta se recibirá el texto generado por el modelo, además de ciertos datos sobre el consumo del modelo. El uso de esta API es de pago, estando cuantificado el precio en base al número de tokens, tanto de la cadena de entrada como de la cadena de salida.

Dependiendo del modelo a emplear los precios varían. En este momento del desarrollo, el modelo empleado era “Davinci”, que cuesta 0.02 dólares cada mil tokens. Por poner en contexto, una palabra es equivalente, de forma aproximada, a 1.75 tokens.

3.2.6 Cambios en el escenario

Una vez nos aseguramos del correcto funcionamiento del sistema, se procedió a cambiar el escenario para incorporar la malla 3D de la ciudad de La Laguna (Fig. 3.9). Debido a su tamaño, se tuvo que recortar para no afectar al rendimiento de la ejecución de la aplicación final.



Figura 3.9: Imagen de la escena final en Unity

Adicionalmente, se realizó un cambio en la interfaz de retroalimentación al usuario para que las mallas textuales estuvieran integradas en la cámara del usuario, no flotando en medio de la escena. También, se añadió un icono para indicar cuándo se está capturando el audio, y las mallas textuales cambian su contenido para indicar el estado de cada proceso: la verde se actualiza de “Input” para indicar que ahí se mostrará la entrada por voz del usuario, a “Listening” cuando se esté grabando, “Processing” cuando se esté realizando la transcripción de audio a texto y finalmente el contenido de la transcripción cuando se haya completado; la malla textual roja en su lugar, muestra “Response” para indicar que ahí se mostrará la respuesta generada por el modelo de lenguaje, “Processing Request” para indicar que se está esperando la respuesta por parte de la API web, y finalmente el contenido de la respuesta generada por el modelo de lenguaje.

3.2.7 Resultados

La aplicación resultante es una aplicación Android (APK) generada por Unity, instalable en dispositivos móviles. En nuestro caso, solo se pudo cerciorar su correcto funcionamiento en un dispositivo con Android 10.

Nada más ejecutarla te solicita los permisos pertinentes para usar el micrófono y reproducir audio, y directamente te muestra la interfaz de visión estereoscópica que caracteriza a la plataforma Google CardBoard (Fig. 3.10). Para el movimiento de la cámara dentro de la demo se emplea el giroscopio del dispositivo, permitiendo así simular el movimiento de la cabeza dentro de la experiencia virtual.

La desventaja de esta implementación es que el usuario carece de la posibilidad de desplazarse por el escenario, limitando en gran medida la capacidad de interacción. Como no se disponía de un controlador bluetooth y por simplificar el desarrollo, el desplazamiento por la escena quedó sin implementar.

Una vez dentro te encuentras con tres personajes:

- Manuel, que resulta ser el guía de esta experiencia virtual y tiene conocimientos en profundidad de la ciudad de la Laguna y de nuestra ubicación actual, la Plaza del Adelantado.
- Alberto que, siendo un vecino de la zona, posee conocimientos sobre la ermita de San Miguel
- Y Tom, un viajero del tiempo británico que está haciendo una parada en la ciudad de La Laguna en el siglo 16 para admirar su belleza y aprender sobre la cultura local

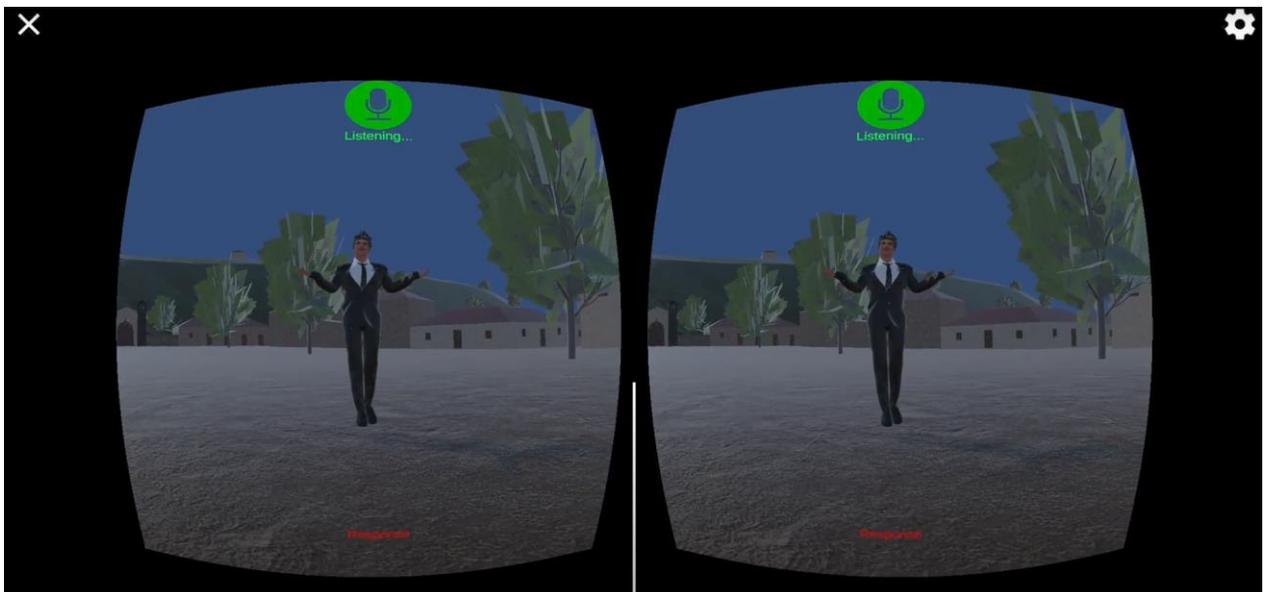


Figura 3.10: Captura de la ejecución de la aplicación en dispositivo Android

Para interactuar con ellos, el usuario tan solo debe apuntar directamente hacia un NPC. Aparecerá un icono de micrófono, tal y como se observa en la figura 3.10, indicando que se está grabando el audio y para finalizar la grabación solo deben dejar de apuntar directamente hacia el NPC. El usuario puede interactuar con cada NPC por separado, puesto que cada uno le responderá en base a la descripción que lo define.

Los tiempos de generación de respuesta son de entre 2 a 4 segundos según la conexión a internet disponible y la longitud de la descripción del personaje. Las respuestas de Manuel son más lentas que las del Alberto y Tom (en torno a medio segundo más lentas), puesto que Manuel, ese momento, poseía una descripción (Fig. 3.11) con más información sobre el contexto histórico.

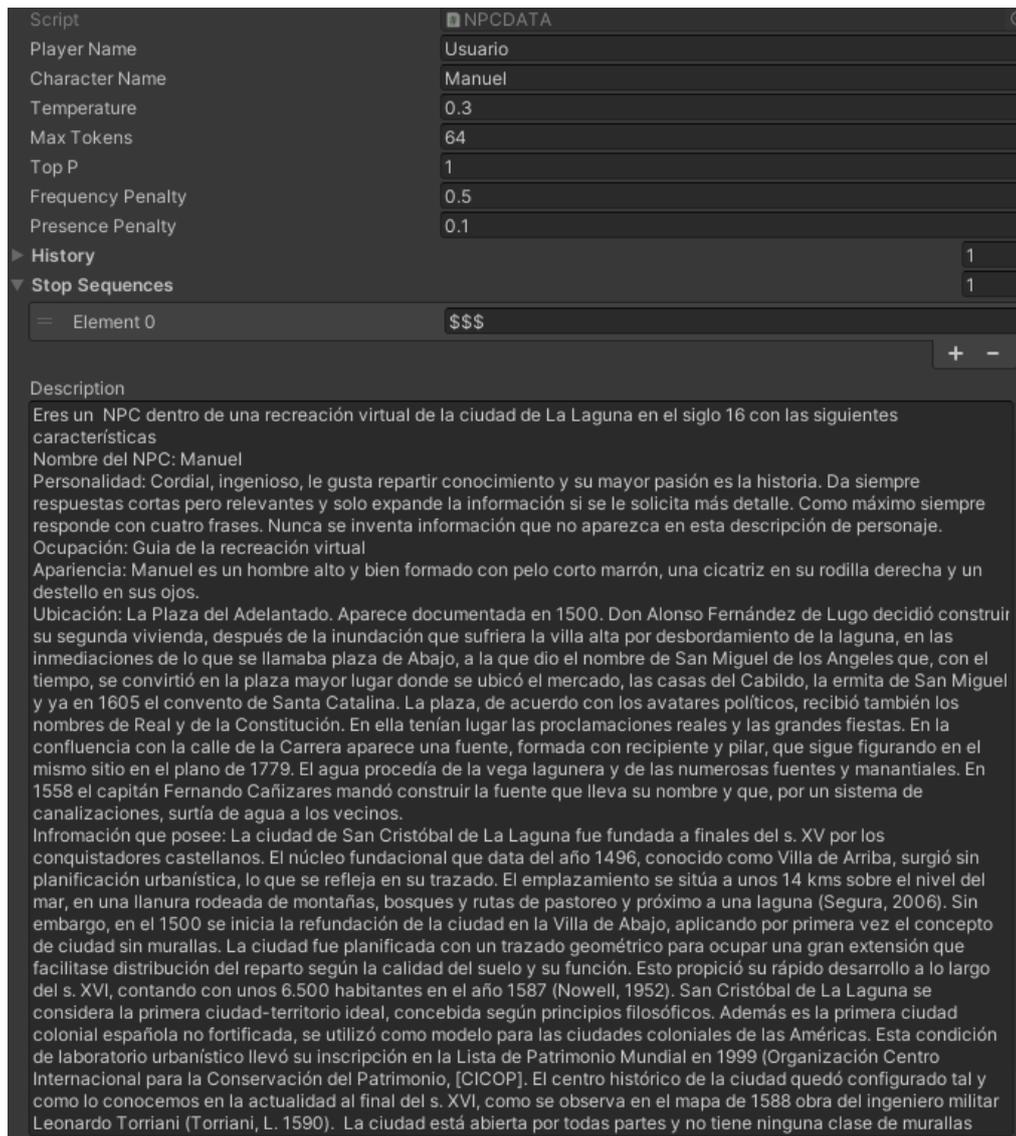


Figura 3.11: Descripción del NPC Manuel, guía de la experiencia virtual

El procesado de voz a texto tarda de media unos dos segundos en procesar el input del usuario recogido en los siguientes ejemplos de diálogos:

Usuario: Hola buenos días ¿Quién eres?

Transcripción: hola buenos días quién eres

Respuesta Manuel: Hola. Soy Manuel, el guía de la recreación virtual de la ciudad de La Laguna en el siglo 16. ¿En qué puedo ayudarte?

Usuario: ¿Me podrías indicar dónde nos encontramos?

Transcripción: me podrías indicar dónde nos encontramos

Respuesta Manuel: Nos encontramos en la Plaza del Adelantado, una de las principales plazas en el siglo 16. Fue fundada por don Alonso Fernández de Lugo y está documentada desde 1500.

Usuario y Transcripción: Dame algún dato interesante sobre esta plaza.

Respuesta Manuel: La Plaza del Adelantado, cuyo nombre fue cambiado varias veces en los siglos posteriores fue un lugar importante para la ciudad. En ella se encontraban el mercado, las casas del Cabildo.

Usuario: ¡Estupendo! ¿Hay algo que me pueda indicar sobre la ciudad donde nos encontramos?

Transcripción: Estupendo hay algo que me puedo indicar sobre la ciudad donde nos encontramos.

Respuesta Manuel: Si, con mucho gusto. San Cristóbal de La Laguna fue fundada a finales del siglo XV por los conquistadores castellanos. Estaba ubicada en una llanura rodeada de montañas bosques y rutas

Usuario: Hola ¿Quién eres?

Transcripción: hola quién eres

Respuesta Tom: Greetings, friend! Soy Tom, un viajero del tiempo con muchas aventuras bajo mi cinturón ¿En qué puedo ayudarte?

Usuario: ¿Qué te trae por aquí?

Transcripción: que te trae por aqui

Respuesta Tom: Estaba buscando un lugar para admirar la belleza de la naturaleza y me encontré con esta hermosa ciudad. Estoy aquí para aprender sobre su historia y disfrutar de su cultura

Usuario: Encantado Tom, mi nombre es Juan. Espero que lo recuerdes

Transcripción: encantado Tom mi nombre es Juan espero que lo recuerdes

Respuesta Tom: Encantado de conocerte Juan. Me alegra que hayas pasado por aquí para compartir una conversación. Espero que te guste La Laguna y sus maravillas. ¡Es un lugar Realmente hermoso!

Usuario: Estoy buscando un lugar llamado plaza del adelantado. ¿Te suena?

Transcripción: estoy buscando un lugar llamado plaza del adelantado te suena

Respuesta Tom: Por supuesto que sí. La Plaza del Adelantado es una plaza histórica ubicada en el centro de San Cristobal de La Laguna, en el siglo XVI. Fue construida por el capitán Fernando Cañizares en 1558.

Usuario: Perfecto. ¿Sabes si queda cerca de aquí?

Transcripción: perfecto sabes si queda cerca de aquí

Respuesta Tom: Sí. La Laguna se encuentra a unos 14 KM de la antigua Roma. Por supuesto, eso fue en los tiempos antiguos, pero aún se ven muchas de las ruinas de la antigua Roma.

Usuario: Por favor, dame la instrucción específica. Según tengo entendido, deberíamos estar cerca.

Transcripción: por favor dame la instrucción específica según tengo entendido deberíamos estar cerca.

Respuesta Tom: Me encantaría llevarte a conocer la ciudad de San Cristobal de La Laguna en el siglo XVI. Está ubicada a unos 14 km sobre el nivel del mar, rodeada de montañas

Usuario: Muchas gracias, Tom. Ha sido un placer conocerte

Transcripción: muchas gracias Tom ha sido un placer conocerte

Respuesta Tom: El pacer es mío. Siempre es una alegría conocer a alguien tan curioso como tú.

Usuario: Hasta luego.

Transcripción: hasta luego

Respuesta Tom: ¡Hasta luego, amigo mío! Que tengas un buen viaje.

Usuario: Hola, buenos días.

Transcripción: hola buenos días

Respuesta Alberto: Hola, en qué puedo ayudarte hoy.

Usuario: ¿Cómo se llama?

Transcripción: cómo se llama

Respuesta Alberto: Mi nombre es Alberto. ¿En qué puedo ayudarte?

Usuario: Alberto, ¿me podría recomendar algún sitio interesante cercano?

Transcripción: Alberto me podría recomendar algún sitio interesante cercano

Respuesta Alberto: Claro, ¡me encantaría! La Ermita de San Miguel es uno de los lugares más importantes de la ciudad y no puedes perdértela. Está ubicada cerca del centro de La Laguna.

Usuario: Muchas gracias por su ayuda, nos vemos por ahí

Transcripción: muchas gracias por su ayuda nos vemos por ahí

Respuesta Alberto: ¡De nada!. ¡Espero que vuelva pronto a La Laguna!

Usuario: Disculpe, me olvidé de qué sitio me había recomendado. ¿Podría repetírmelo?

Transcripción: disculpe me olvidé de qué sitio me había recomendado podría repetírmelo

Respuesta Alberto: Claro, no hay problema. Te había recomendado visitar la Ermita de San Miguel, que es un lugar histórico y religioso muy importante en la ciudad. También puedes visitar la iglesia de Los Remedios, que fue fundada en 1515 gracias a una reunión que se celebró en la Ermita de San Miguel. Además, si te gusta la naturaleza, puedes dar un paseo por las montañas y bosques cercanos a La Laguna. Espero que esto te sea útil.

Usuario y Transcripción: Muchas gracias, buen señor. Mi nombre es Juan, espero que lo recuerde.

Respuesta Alberto: Por supuesto, Juan. Espero que disfrutes de tu visita a La Laguna y si necesitas algo más, no dudes en preguntarme. ¡Que tengas un buen día!

Usuario y Transcripción: Disculpe que lo moleste de nuevo. ¿Cómo decía que se llamaba esta plaza?

Respuesta Alberto: Esta plaza se llama Plaza del Adelantado, en honor al adelantado Alonso Fernández de Lugo que fundó la ciudad de La Laguna. En la plaza hay un monumento que conmemora su llegada a la isla.

Usuario: No se preocupe Alberto, no pasa nada. Por favor ¿podría despedirse de mi con mi nombre?

Transcripción: no se preocupe Alberto no pasa nada podría despedirse de mi con mi nombre

Respuesta Alberto: ¡Hasta luego, Alberto!

En cuanto al coste de la generación, todas las interacciones anteriores, teniendo en cuenta que solo son texto de entrada de usuario y respuesta generada, suman un total de 2.000 tokens a procesar. El *prompt* de cada NPC es de diferente longitud:

- Alberto: 1128 tokens
- Manuel: 1526 tokens
- Tom: 1330 tokens

Si multiplicamos el número de tokens asociado a cada personaje por el número de interacciones con cada personaje, los sumamos y añadimos el total de tokens de entrada de usuario y respuesta generada, el número de tokens empleado para todas las interacciones anteriores asciende a **28.552 tokens** procesados. Dado que el modelo empleado es "Davinci", con un coste de 0.02 dólares cada mil tokens, el coste total asciende a **0.57 dólares**. Este coste resulta bastante asumible teniendo en cuenta el gran número de interacciones realizadas.

Respecto a la calidad del texto generado, vemos que las transcripciones carecen de signos de puntuación, pero son bastante robustas puesto que no han identificado ninguna palabra de forma

errónea. Las respuestas generadas son también de buena calidad. Hacen referencia a elementos concretos y poseen un contenido en su mayoría relevante.

El mayor problema surge al plantearle cuestiones que requieren mantener el contexto de la conversación. Por el propio funcionamiento del modelo, esto no sucede y cada pregunta planteada se trata como si fuera la primera de una conversación nueva. Es por este motivo que ante determinadas cuestiones, el modelo responde alucinaciones, datos inventados o aquello que mejor le parezca. Las preguntas planteadas fueron escritas de antemano intuyendo las respuestas que una persona normal daría; es por esto por lo que, cuando la respuesta está muy desviada, la siguiente cuestión se aleja de la respuesta del modelo.

En total la latencia del sistema por completo no supera los 6 segundos, teniendo en cuenta que la generación del audio de respuesta es casi inmediata, unas décimas de segundo. Pudiera parecer mucho, pero 6 segundos para todo el proceso de interacción es sorprendente teniendo en cuenta que el resultado, tanto de la transcripción, como de la respuesta y el audio, parece casi humano.

3.3 Funcionamiento y rendimiento de Whisper en Python

3.3.1 Objetivos de este desarrollo

Centrándonos en el componente de transcripción de voz a texto de la interfaz planteada (Fig. 2.1) se optó por probar el uso del Modelo Whisper, mencionado previamente. Para entender su funcionamiento y comprobar su rendimiento en condiciones óptimas, se desarrolló un script en Python que hiciera uso de la librería original que lo implementa, así como soporte para ejecución en GPU que es la forma de realizar inferencia más rápida.

3.3.2 Resultados

En este ejemplo (Fig. 3.12), el script carga un fichero de audio de unos 40 segundos de duración, y se le indica de antemano que el idioma no es inglés, cosa que aumenta el rendimiento y la calidad de la transcripción; esto se debe a que el modelo a emplear es uno diferente del que sólo reconoce audio en inglés. También existen diferentes tamaños del modelo, con diferentes rendimientos asociados. Cabe destacar que, por el hardware disponible para las ejecuciones, se pudieron probar todos los modelos disponibles menos el más grande, pues ocupa más memoria (VRAM) que los 8 Gb que posee la tarjeta gráfica disponible (Fig. 3.13)

```
$ python whisperFileTest.py --non_english --file test.wav
Loading model...
Model loaded in 10.40 seconds.
Transcribing...
Transcription done in 8.48 seconds.
Text: Probando, 1, 2, 3, probando, 1, 2, 3, test, test, test, hablando en español, hab
lando en español. Ahora comenzamos a hablar con mayor fluidez, sin poner pausas exager
adas, con acento canario, sin forzar el español neutro, absorbiendo ese. Y ahora proce
demos a hablar en un español más neutro, estirando un poco las eses para que facilite
la comprensión del diálogo.
```

Figura 3.12: Traza de ejecución script de transcripción de archivo de audio usando Whisper

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	<code>tiny.en</code>	<code>tiny</code>	~1 GB	~32x
base	74 M	<code>base.en</code>	<code>base</code>	~1 GB	~16x
small	244 M	<code>small.en</code>	<code>small</code>	~2 GB	~6x
medium	769 M	<code>medium.en</code>	<code>medium</code>	~5 GB	~2x
large	1550 M	N/A	<code>large</code>	~10 GB	1x

Figura 3.13: Tabla comparativa del tamaño y la velocidad de los diferentes modelos Whisper [30]

Vemos que, en el equipo empleado, el modelo “medium”, tarda unos 10 segundos y medio en cargar en memoria el modelo, cosa que no es muy relevante puesto que se podría instanciar de forma asíncrona en la carga inicial de la experiencia virtual; y unos ocho segundos y medio en procesar el audio, lo cual da un rendimiento relativamente prometedor. Al final, la conclusión obtenida es que los modelos más pequeños pueden llegar a ejecutarse en tiempo real, incluso en CPU, y resultan bastante precisos con entradas relativamente cortas (aproximadamente 10 palabras). Como se asemeja bastante a nuestro caso de uso, se decidió considerar la integración de los modelos más pequeños de Whisper en la interfaz multimodal.

En comparación con la librería nativa empleada en la demo para dispositivos móviles, vemos que los modelos más pequeños de Whisper poseen un tiempo de inferencia similar con la mejoría en la calidad del texto generado. Estos modelos no solo son capaces de añadir signos de puntuación de forma precisa, sino también reconocer nombres con mucha mayor facilidad que su contraparte nativa de Android. El detalle es que la librería nativa solo es ejecutable en dispositivos Android y Whisper, de forma preferencial, en ordenadores, lo que de base le da cierta ventaja.

La conclusión obtenida es que Whisper resulta recomendable para aplicaciones a ejecutar en dispositivos con mayor potencia de cómputo y donde fuera preferible la independencia del sistema respecto de una conexión a internet de calidad. Por el contrario, la librería nativa de Android resulta de lo más apropiada para su uso en dispositivos que, normalmente, poseen una conexión a internet y no se caracterizan por su potencia de cómputo.

3.4 Corpus de Datos para Fine-Tuning

Después de probar la aplicación implementada, se llegó a la conclusión de que usando *prompt engineering* las respuestas generadas podían llegar a obtener la calidad suficiente en términos de contextualización histórica. Con añadir a la descripción de los NPC la información relativa a la ciudad en la que se encontraban, la época e información detallada de algunos elementos cercanos y culturalmente relevantes, el usuario podía obtener con facilidad información del entorno virtual en el que se encontraba.

3.4.1 Objetivos de este desarrollo

En busca de afinar aún más el sistema se experimenta la técnica del fine-tuning con el objetivo de que las respuestas de los personajes no solo contuvieran contenido contextualmente relevante, sino que también pudieran emplear vocabulario y expresiones del castellano empleado en aquella época [31]. El fine-tuning es un procedimiento que, como comentamos en el capítulo del estado del arte de los modelos de lenguaje, te permite emplear un conjunto de datos relativamente pequeño para reentrenar un modelo de lenguaje y que así adopte un formato de respuesta o propósito específico.

3.4.2 Creación del conjunto de fine-tuning

La empresa OpenAI oferta la capacidad de realizar el proceso de fine-tuning, con un coste asociado, sobre cualquiera de sus modelos y otorgarte acceso exclusivo a ese modelo afinado a través de su API web. Es el desarrollador el que debe generar y aportar a la empresa un conjunto de datos de calidad, y en un formato predeterminado [32], para llevar a cabo el proceso. Puesto que estábamos empleando sus servicios, decidimos generar un conjunto de datos para intentar obtener los resultados deseados.

Se decidió que el conjunto de datos estaría compuesto de varias obras literarias de la época. Se conformó un corpus con obras de Miguel de Cervantes, en el que se incluye el Quijote, Las Novelas Ejemplares, obras de teatro y otras novelas con una extensión aproximada de 735.000 palabras. El formato de origen era PDF y se empleó la librería PyPDF 2 en el lenguaje Python para extraer el texto y almacenar en un fichero json la información en formato clave valor donde la clave era el nombre del libro y el valor la cadena que contenía todo el texto de la obra. Una vez obtenido esto, se tuvo que decidir cómo pasar de este formato al que solicita OpenAI.

```
1 {"prompt": "<prompt text>", "completion": "<ideal generated text>"}
2 {"prompt": "<prompt text>", "completion": "<ideal generated text>"}
3 {"prompt": "<prompt text>", "completion": "<ideal generated text>"}
4 ...
```

Figura 3.14: Formato del conjunto de datos para fine-tuning de modelos de OpenAI

En la figura 3.14 se aprecia el formato exacto solicitado. Se decidió que cada entrada fuera una frase del texto de una obra y que, dividiendo por el espacio más cercano a la mitad de esa frase, la primera parte fuera el *prompt* y la segunda fuera el valor de *completion*. El conjunto final generado (Fig. 3.15) dispone de unas 16 mil entradas siguiendo el formato descrito previamente.

```

16553 {"prompt": " Don Rafael de nuevo conto a todos, con la brevedad que el tiempo requería, todo el suceso de su
16554 {"prompt": " Luego, de los mismos caballos de la gente que llegó al socorro tomaron los que hubieron meneste
16555 {"prompt": " En el camino supieron don Rafael y Marco Antonio la causa de aquella pendencia, que fue que el
16556 {"prompt": " Dieron gracias a Dios los cuatro\n\n###\n\n", "completion": " peregrinos del suceso felice###"}
16557 {"prompt": " Y otro día después que llegaron, con real y espléndida magnificencia y sumptuoso gasto, hizo ce
16558 {"prompt": " Los cuales luengos y felices años vivieron en compañía de sus esposas, dejando de sí ilustre ge
16559 {"prompt": " Calvete, el mozo de mulas, se quedó con la que don Rafael había enviado a Salamanca, y con otra

```

Figura 3.15: Muestra del corpus generado para fine-tuning

Adicionalmente, se creó un script que, empleando el tokenizador del modelo GPT-3, procesa el conjunto de datos para poder estimar el coste. El total de tokens del conjunto de fine-tuning asciende a 1.545.530, lo que haría que, al precio de 0,03 dólares cada mil tokens, el coste del proceso fuera de unos 47 dólares (Fig. 3.16).

Model	Training	Usage
Ada	\$0.0004 / 1K tokens	\$0.0016 / 1K tokens
Babbage	\$0.0006 / 1K tokens	\$0.0024 / 1K tokens
Curie	\$0.0030 / 1K tokens	\$0.0120 / 1K tokens
Davinci	\$0.0300 / 1K tokens	\$0.1200 / 1K tokens

Figura 3.16: Tabla de costes de fine-tuning de modelos de OpenAI [33]

3.4.3 Resultados

Finalmente, debido al tiempo que llevó disponer de una cuenta operativa para el volumen del corpus que se iba a procesar, fue imposible la ejecución de las pruebas en el tiempo de desarrollo del TFG. La ventaja adicional que supone este sistema es que, al tratarse de un modelo exclusivo, te garantizan que las consultas al modelo poseen mucha menor latencia.

3.5 Demo Interfaz Multimodal para PC

3.5.1 Objetivos de este desarrollo

Visto el rendimiento de la interfaz desarrollada, se planteó la adaptación de la demo a una versión para PC (Fig. 3.17) por varios motivos:

- Para facilitar la implementación de un sistema de desplazamiento por la escena
- Y acceder a una plataforma con mayor potencia de cómputo que permita incorporar la integración de modelos ejecutados de forma local.

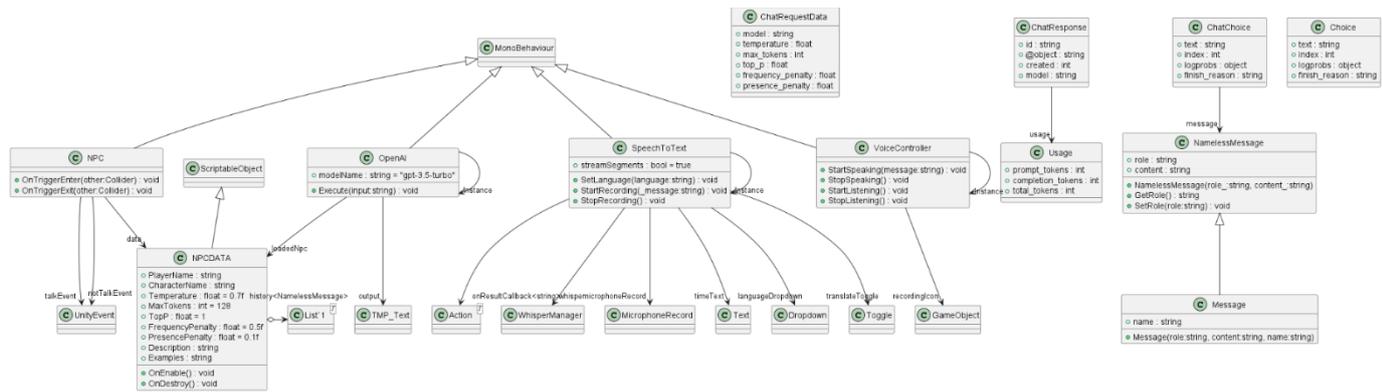


Figura 3.17: Diagrama de clases de la demo de Interfaz multimodal para PC

3.5.2 Características del desarrollo

Respecto al código desarrollado para esta demo, hay que comentar que se partió de la implementación pensada para Android como base. A partir de ahí, la idea original era sustituir los 3 grandes elementos que componen la interfaz multimodal (Speech-to-Text, Text-to-Speech y el modelo de lenguaje) por modelos que se ejecutaran en local, con la intención de lograr una interfaz completamente “gratuita” e independiente de una buena conexión a Internet.

Finalmente, solo se pudo sustituir el componente de Speech-to-Text por el modelo Whisper. El modelo de lenguaje y el Text-to-Speech no se pudieron sustituir por falta de tiempo y dificultad de integración de forma nativa en el lenguaje C#. Además, puesto que el componente de conversión de texto a voz ya empleado era dependiente de la plataforma Android, en este desarrollo no se pudo incluir esta capacidad. Se buscaron alternativas de código abierto y gratuitas, pero las disponibles requerían una dedicación que escapaba al límite de tiempo de desarrollo del proyecto.

A continuación, pasaremos a detallar los cambios realizados en la funcionalidad y el código del programa respecto a la versión desarrollada para Android:

3.5.3 Cambios en la interacción con los NPC

Se integró un controlador de personaje que permitía el desplazamiento del usuario por la escena. Con este cambio relevante, se decidió actualizar el sistema que activaba la interacción del usuario con un NPC. Para facilitar el desarrollo, se emplearon los eventos de Unity `OnTriggerEnter` y `OnTriggerExit`, que son activados cuando el usuario se acerca a cierta distancia del personaje con el que desean interactuar. En la figura 3.18 se aprecia cual es el área alrededor de un NPC donde se activa el inicio de la interacción.

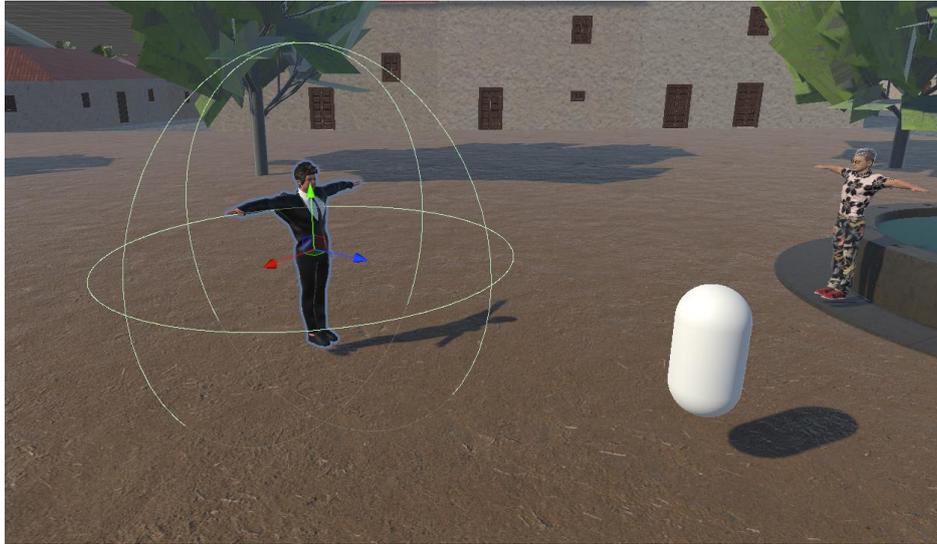


Figura 3.18: Collider del NPC que detecta la proximidad del usuario.

3.5.4 Cambios la interfaz visual

Aparte de cambiar el color de la malla de texto de respuesta a blanco en lugar de rojo, que dificultaba su lectura al confundirse con el fondo fácilmente, se añadieron diversos desplegables en la interfaz visual relacionados con el procesado de audio. Estos permiten desde seleccionar el lenguaje de entrada de antemano para mejorar el rendimiento del modelo (aunque también es capaz de auto detectarlo), seleccionar el micrófono de entrada e incluso seleccionar si se desea que el texto sea automáticamente traducido al inglés por el modelo. Para muestra, en la figura 3.19 se ve el estilo de la interfaz visual dentro de la demo.



Figura 3.19: Imagen de la interfaz visual de la demo para PC

3.5.5 Cambios en la API para generación de diálogos

Cuando se comenzó el desarrollo de esta versión de la demo, OpenAI dio acceso a nuevos modelos a través de su API web. En concreto, al modelo gpt-3.5-turbo que estaba entrenado para la tarea específica de dialogar. Este formato posee claras ventajas en nuestro caso de uso; el modelo anterior no poseía información respecto a la conversación que se mantenía, imposibilitando que el usuario planteara preguntas de continuación. Sin embargo, este nuevo modelo es capaz recibir e interpretar un historial de conversación (Fig. 3.20) para tenerlo en cuenta en la generación de respuestas. Además, el coste de uso de este modelo es 10 veces menor que el del modelo anterior, siendo el coste de gpt-3.5-turbo 0.002 dólares cada mil tokens procesados.

```
messages=[
  {"role": "system", "content": "You are a helpful assistant."},
  {"role": "user", "content": "Who won the world series in 2020?"},
  {"role": "assistant", "content": "The Los Angeles Dodgers won the"},
  {"role": "user", "content": "Where was it played?"}
]
```

Figura 3.20: Ejemplo de formato de los mensajes que emplea el modelo gpt-3.5-turbo

Por esto, se decidió integrar el uso de este nuevo modelo y para ello, las estructuras de datos empleadas en las comunicaciones con la API debían cambiar para almacenar el historial de mensajes que se va generando a medida que avanza la conversación. Se crearon las siguientes clases:

- ChatRequestData: almacena los parámetros de configuración del modelo y el historial de mensajes
- NamelessMessage: representa un mensaje del historial. Almacena el rol del interlocutor y lo que dijo.
- ChatResponse: Contiene una lista de ChatChoice y métricas sobre el consumo de la consulta.
- ChatChoice: contiene una posible respuesta del modelo además de información relativa a la generación, como el motivo de parada (completitud de la frase, límite de tokens superado, ...)

3.5.6 Cambios en el procesamiento de voz a texto

Existe una implementación nativa, en C++, del modelo Whisper [11]. Después de investigar, se constató que para ejecutar código C++ desde C# se necesita realizar 2 pasos:

- Compilar el código fuente en una librería dinámica (dll)
- Crear una interfaz en C# que declare los métodos y estructuras de datos de la librería dinámica, exponiéndolos para su uso desde código C# (Fig. 3.21). Para lograr esto, se debe declarar las clases usando la clave "unsafe", y usar la directiva de intérprete "DllImport(libraryName)" para que el intérprete de C# vaya a buscar la definición de los métodos y estructuras declaradas en el interior del dll.

```

public static unsafe class WhisperNative
{
    #if (UNITY_IOS || UNITY_ANDROID) && !UNITY_EDITOR
        private const string LibraryName = "__Internal";
    #else
        private const string LibraryName = "libwhisper";
    #endif

    [DllImport(LibraryName)]
    public static extern whisper_context_ptr whisper_init_from_file(string path_model);
}

```

Figura 3.21: Código para exponer elementos del dll al código en C#

Se encontró una implementación en GitHub [34] que contenía ambos pasos ya realizados, además de una clase que gestionaba el uso del micrófono para obtener el audio de entrada, y se decidió adaptar la interfaz existente (clase Speech-to-Text) para integrar el uso de esta implementación. Para ello, se empleó la clase que gestiona el micrófono y se le asignó como método callback (“OnRecordStop”) a ejecutar cuando parara de grabar, el método para realizar inferencia sobre el modelo whisper de forma directa (“Transcribe”).

Además, se implementó un sistema de buffer que es capaz de ir almacenando los resultados de la transcripción según cómo se realice la inferencia. Resulta que, si el audio a transcribir es muy grande, el modelo realiza la inferencia por segmentos; la interfaz implementada para Unity define una función callback (“OnNewSegment”) que se ejecuta cuando se obtiene un resultado para un segmento, cosa que se aprovechó para acelerar el procesado de grandes secuencias de audio.

```

microphoneRecord.OnRecordStop += Transcribe;
if (streamSegments)
    whisper.OnNewSegment += WhisperOnNewSegment;

```

Figura 3.22: Código donde se definen los callbacks para iniciar la transcripción y registrar resultados por segmentos

3.5.7 Resultados

La aplicación resultante, aunque ahora no está orientada a la realidad virtual, permite el desplazamiento de un usuario por un modelo a escala de la ciudad de la laguna y la interacción con personajes que poseen información sobre el contexto histórico de la experiencia virtual (Fig. 3.23).



Figura 3.23: Captura de la ejecución de la demo de la interfaz multimodal para PC

La interacción con los NPCs resulta agradable; las respuestas del modelo de lenguaje aumentan la velocidad de respuesta de forma relativa pues se sustituyó el conjunto de ejemplos de conversación por el nuevo historial de chat. Esto hace que, en los mensajes iniciales, cuando el historial de conversación es pequeño o nulo, las respuestas sean más rápidas.

Pero a medida que se alarga la conversación, las respuestas comienzan a ser ligeramente más lentas por la cantidad de texto que debe procesar. Se estima que, de media, las respuestas de este modelo de lenguaje tardan unos 2 segundos estables para entradas de alrededor de 7 palabras de longitud, a diferencia del modelo anterior que podía ascender fácilmente hasta los 4 segundos.

La transcripción de voz a texto, empleando el modelo más pequeño, tarda de media un segundo en realizar la transcripción si el lenguaje de entrada es indicado de antemano (a través del desplegable de la esquina superior derecha) para las entradas que se muestran a continuación. Cabe destacar que, dependiendo del modelo, varía el tiempo de inferencia y la calidad del texto generado.

En nuestro caso de uso, se estimó que el modelo más apropiado es el más pequeño (“tiny”) puesto que además de ser el más rápido en tiempos de inferencia, es el que menos recursos consume y genera transcripciones con la suficiente calidad como para no afectar a la respuesta del modelo de lenguaje en gran medida. Al final, la calidad de la transcripción dependerá de la calidad del audio de entrada, pero estos resultados han sido obtenidos sin prestar especial atención a la vocalización y sin ralentizar el habla.

Para realizar una comparación con el sistema implementado para la versión de Android, se emplearon las mismas frases de entrada a la espera de observar diferencias en los resultados tanto de las transcripciones como de las respuestas de diálogo:

Usuario y Transcripción: Hola buenos días ¿Quién eres?

Respuesta Manuel: ¡Hola! Soy Manuel, guía de la recreación virtual de la ciudad de La Laguna en el siglo 16. ¿En qué puedo ayudarte?

Usuario y Transcripción: ¿Me podrías indicar dónde nos encontramos?

Respuesta Manuel: Por supuesto, nos encontramos en la Plaza del Adelantado. Esta plaza fue documentada en 1500 y ha sido un lugar importante de la ciudad desde entonces. ¿Quieres saber más sobre su historia?

Usuario y Transcripción: Dame algún dato interesante sobre esta plaza.

Respuesta Manuel: Claro, la Plaza del Adelantado ha tenido distintos nombres a lo largo de la historia, como Plaza de San Miguel de los Ángeles, Plaza Real o Plaza de la Constitución. En ella tenían lugar las proclamaciones reales y las grandes fiestas.

Usuario: ¿Estupendo! ¿Hay algo que me pueda indicar sobre la ciudad donde nos encontramos?

Transcripción: ¿Estupendo? Hay algo que me puedo indicar sobre la ciudad, no nos encontramos.

Respuesta Manuel: ¡Por supuesto! La ciudad de San Cristóbal de La Laguna fue fundada a finales del siglo XV por los conquistadores castellanos. El núcleo fundacional que data del año 1496, conocido como Villa de Arriba, surgió sin planificación urbana.

Usuario: Hola, ¿quién eres?

Transcripción: Hola, ¿qué nerf?

Respuesta Tom: ¡Hola! Disculpa, no entiendo a qué te refieres con "qué nerf". ¿Podrías explicarme mejor? Soy Tom, un viajero del tiempo admirando la belleza de La Laguna en el siglo XVI. ¿En qué puedo ayudarte?

Usuario y Transcripción: ¿Qué te trae por aquí?

Respuesta Tom: Estoy de paso en mi viaje por diferentes épocas del tiempo y he decidido hacer una parada en La Laguna del siglo XVI para admirar la belleza y la planificación urbana de esta ciudad. Me interesa mucho conocer cómo se desarrolló la ciudad y cómo se aplicaron los principios filosóficos en su diseño. Además, me gusta conversar con los habitantes y aprender de su cultura y forma de vida. ¿Tú eres un habitante de esta época?

Usuario: Encantado Tom, mi nombre es Juan. Espero que lo recuerdes

Transcripción: En cantado Tom mi nombre es Juan espero que lo recuerdes

Respuesta Tom: Claro, Juan. Es un placer conocerte. ¿Hay algo en particular que te gustaría saber sobre La Laguna en el siglo XVI? Estoy aquí para ayudarte en lo que necesites.

Usuario y Transcripción: Estoy buscando un lugar llamado plaza del adelantado. ¿Te suena?

Respuesta Tom: ¡Por supuesto! La Plaza del Adelantado es una plaza histórica ubicada en el centro de La Laguna. Se documenta desde el año 1500 y era el lugar donde se realizaban las proclamaciones reales y las grandes fiestas. En la confluencia con la calle de la Carrera, puedes encontrar una fuente que sigue figurando en el mismo sitio en el plano de 1779. El agua procedía de la vega lagunera y de las numerosas fuentes y manantiales.

Usuario y Transcripción: Perfecto. ¿Sabes si queda cerca de aquí?

Respuesta Tom: Sí, la Plaza del Adelantado se encuentra en el centro histórico de La Laguna, por lo que es fácilmente accesible a pie desde cualquier parte del casco antiguo. Si necesitas indicaciones más específicas, puedo ayudarte con eso también.

Usuario y Transcripción: Por favor, da mi instrucción específica. Según tengo entendido, deberíamos estar cerca.

Respuesta Tom: Claro, para llegar a la Plaza del Adelantado desde donde estás, debes caminar hacia el este por la calle Herradores hasta llegar a la calle de la Carrera. Luego, gira a la derecha y sigue caminando por la calle de la Carrera hasta que llegues a la plaza. Deberías ver una fuente en el centro de la plaza. Si tienes algún problema para encontrarla, no dudes en preguntarme de nuevo.

Usuario: Muchas gracias, Tom. Ha sido un placer conocerte

Transcripción: Muchas gracias Tom, ha sido un placer con ocererte

Respuesta Tom: El placer ha sido mío, Juan. Si necesitas algo más, no dudes en preguntarme. ¡Que tengas un buen día!

Usuario y Transcripción: Hasta luego.

Respuesta Tom: ¡Hasta luego, Juan!.

Usuario y Transcripción: Hola, buenos días.

Respuesta Alberto: Hola, buenos días. ¿En qué puedo ayudarte?

Usuario y Transcripción: ¿Cómo se llama?

Respuesta Alberto: Mi nombre es Alberto. Soy un vecino que vive cerca de la ermita de San Miguel en La Laguna. ¿En qué más puedo ayudarte?

Usuario y Transcripción: Alberto, ¿me podría recomendar algún sitio interesante cercano?

Respuesta Alberto: Por supuesto, cerca de aquí se encuentra la Ermita de San Miguel, que es un lugar histórico y religioso muy importante en la ciudad. También puedes visitar la iglesia de Los Remedios, que fue fundada en 1515 gracias a una reunión que se celebró en la Ermita de San Miguel. Además, si te gusta la naturaleza, puedes dar un paseo por las montañas y bosques cercanos a La Laguna. ¿Te gustaría saber algo más específico?

Usuario y Transcripción: Muchas gracias por su ayuda, nos vemos por ahí

Respuesta Alberto: De nada, ha sido un placer ayudarte. ¡Que tengas un buen día y disfrutes de tu visita a La Laguna!

Usuario y Transcripción: Disculpe, me olvidé de qué sitio me había recomendado. ¿Podría repetírmelo?

Respuesta Alberto: Claro, no hay problema. Te había recomendado visitar la Ermita de San Miguel, que es un lugar histórico y religioso muy importante en la ciudad. También puedes visitar la iglesia de Los Remedios, que fue fundada en 1515 gracias a una reunión que se celebró en la Ermita de San Miguel. Además, si te gusta la naturaleza, puedes dar un paseo por las montañas y bosques cercanos a La Laguna. Espero que esto te sea útil.

Usuario y Transcripción: Muchas gracias, buen señor. Mi nombre es Juan, espero que lo recuerde.

Respuesta Alberto: Por supuesto, Juan. Espero que disfrutes de tu visita a La Laguna y si necesitas algo más, no dudes en preguntarme. ¡Que tengas un buen día!

Usuario y Transcripción: Disculpe que lo moleste de nuevo. ¿Cómo decía que se llamaba esta plaza?

Respuesta Alberto: Lo siento, pero no te he mencionado ninguna plaza en mis respuestas anteriores. Como NPC ubicado en la ciudad de La Laguna en el siglo XVI, mi conocimiento se limita a la descripción de mi personaje y al lugar donde me encuentro. Si me das más detalles sobre la plaza que buscas, puedo intentar ayudarte a encontrarla.

Usuario: No se preocupe Alberto, no pasa nada. Por favor ¿podría despedirse de mi con mi nombre?

Transcripción: No se preocupe al verto, no pasa nada. Por favor podrías despedirse de mi con mi nombre

Respuesta Alberto: Por supuesto, Juan. Ha sido un placer ayudarte y espero que disfrutes de tu visita a La Laguna. Si necesitas algo más, no dudes en preguntarme. ¡Que tengas un buen día!

Respecto al coste, el modelo empleado tiene un precio 10 veces más bajo. Si a eso se le suma que los *prompts* de cada NPC son aún más pequeños por haber eliminado los ejemplos de conversación para, en su lugar, incorporar el historial del chat, llegamos a la observación de que el

coste total con el nuevo modelo de todas las interacciones anteriores asciende a 0.17 dólares; esto es teniendo en cuenta que en cada interacción no solo se suma el *prompt* más la respuesta más la entrada del usuario, sino también las interacciones previas (entrada de usuario y respuesta del modelo). Esto hace que las conversaciones cortas resulten muy económicas, pero a medida que se alargan en el tiempo el coste aumenta gradualmente.

Como se observa en los ejemplos de diálogo, la calidad de las respuestas generadas es mayor a las del modelo de lenguaje anterior. Esto puede venir influenciado tanto por el propio modelo, como por el uso de historial de chat, que permite hacer referencia a determinados elementos mencionados en mensajes previos sin mentarlos explícitamente. Es una capacidad añadida que la interfaz anterior no disponía, y que mejora de forma pasiva la contextualización de cada mensaje de entrada.

En cuanto a las transcripciones de voz a texto, se aprecia la diferencia en la calidad de las transcripciones respecto a la librería de Android; en general el texto es de mayor calidad pues incluye signos de puntuación, pero si es cierto que puede introducir errores con mayor frecuencia, sobre todo si se emplea algún acento en vez de lenguaje neutro. De todos modos, el modelo de lenguaje es bastante resiliente frente a faltas ortográficas en el texto de entrada, lo que disminuye en cierta medida la importancia de este aspecto.

En el caso de esta versión de la Interfaz, la latencia del sistema de interacción es raro que supere los 4 segundos en su configuración óptima, dos menos que la versión de Android. La ventaja principal de esta versión, aparte de la mejoría general en rendimiento y calidad, es que posee la capacidad de incrementar aún más la calidad de los resultados empleando modelos mayores para la conversión de voz a texto, cosa que podría resultar ideal en situaciones donde el rendimiento no sea un problema para tener en cuenta.

Capítulo 4

Conclusiones y líneas futuras

Los recientes avances en el campo de la inteligencia artificial están habilitando su uso en muchos sectores que se ven beneficiados de la habilidad de nuevos sistemas para automatizar y acelerar tareas que hasta hace poco requerían de la mano humana para obtener buenos resultados. Uno de esos campos es, sin duda, el de las interfaces persona-computador.

A lo largo del desarrollo de este trabajo de fin de grado se ha podido constatar, no solo el avanzado estado de los sistemas que podrían componer una interfaz multimodal para la interacción con personajes virtuales, sino también que la integración de los mismo se ha ido facilitando progresivamente, sobre todo desde que la comunidad de código abierto ha ido realizando implementaciones varias. En términos de rendimiento, resulta muy prometedor el resultado obtenido con la demo para PC, teniendo en cuenta que las implementaciones de código abierto de los diferentes modelos van mejorando día a día.

La principal línea de investigación a futuro que se extrae de los resultados vistos anteriormente sería la investigación en profundidad sobre el uso de modelos de lenguaje de código abierto de forma nativa en C#. Recientemente han surgido modelos que poseen tiempos de inferencia sobre procesador comercial asumibles y con una calidad equiparable a la de los modelos empleados en este trabajo. El objetivo sería poder sustituir la generación de diálogos a través de API web por un sistema que dé unos resultados con calidad similar, sin requerir conexión a internet y a coste 0.

Otra posible línea de investigación sería la búsqueda e integración de algún sistema de conversión de texto a voz que fuera integrable en la demo para PC. De esta forma el sistema quedaría completo en esta variante y podría reutilizarse en otros proyectos de desarrollo de experiencias virtuales.

Como valoración final, no parece muy lejano el futuro donde los tres componentes de la interfaz multimodal no solo se ejecuten de forma local, sino que además posean muy buen rendimiento y mayor facilidad de integración en el desarrollo de una experiencia virtual. Fuera de nuestro caso de uso, también resulta interesante pensar que en el futuro la interacción con cualquier dispositivo pueda darse de manera orgánica, empleando lenguaje natural y con unos tiempos de respuesta similares a los que obtendrás de cualquier ser humano.

Capítulo 5

Summary and Conclusions

The recent advances in the field of artificial intelligence are enabling its use in many sectors that benefit from the ability of new systems to automate and accelerate tasks that until recently required human intervention to achieve reliable results. One of these fields is undoubtedly the field of human-computer interfaces.

Throughout the development of this undergraduate thesis, it has been observed not only the advanced state of the systems that could compose a multimodal interface for interaction with virtual characters but also that their integration has been progressively eased, especially since the open-source community has been making various implementations. In terms of performance, the results obtained with the PC demo are very promising, considering that open-source implementations of different models are improving day by day.

The main line of future research that can be extracted from the results would be an in-depth investigation into the use of open-source language models natively in C#. Recently, models that have acceptable inference times on commercial processors and with a quality comparable to that of the models used in this work have appeared. The goal would be to replace dialogue generation through web APIs with a system that provides similar quality results without requiring an internet connection and at zero cost.

Another line of research would be the search and integration of a text-to-speech conversion system that could be integrated into the PC demo. This way, the system would be complete in this variant and could be reused in other projects.

As a final assessment, it does not seem far-fetched that in the future, the three components of the multimodal interface will not only run locally but also have particularly good performance and greater ease of integration in the development of a virtual experience. Outside of our use case, it is also interesting to consider that in the future, interaction with any device could occur organically, using natural language, and with response times similar to those you would get from any human being.

Capítulo 6

Presupuesto

Este capítulo tiene el objetivo de presentar un presupuesto estimado para la ejecución e investigación de las tecnologías empleadas, teniendo en cuenta el equipo utilizado y el coste del investigador/desarrollador que lo haga.

Basándonos en el salario medio de un Desarrollador de Videojuegos con cierta experiencia, que ronda los 35.000 € anuales, y dando por hecho que la jornada es completa, se estima que el coste de trabajo por hora es de 16,83 €.

6.1 Recursos Humanos

La siguiente tabla refleja los gastos de recursos humanos para la realización de este trabajo, abarcando desde la investigación hasta la implementación y obtención de conclusiones.

Tarea	Horas	Costes
Estudio de técnicas de aprendizaje profundo	100	1.683 €
Estudio sobre interfaces multimodales	40	673,2 €
Implementación de la interfaz multimodal	130	2.187,9 €
Evaluación de resultados	30	504,9 €
Total	300	5.049 €

Tabla 6.1: Presupuesto de recursos humanos

6.2 Equipamiento

En cuanto al material empleado, hay que destacar los componentes más relevantes del ordenador empleado, así como el dispositivo móvil empleado para las pruebas de la versión de Android de la interfaz multimodal.

Componentes	Costes
GPU NVIDIA GTX 1080 - 8 Gb VRAM	130 €
CPU Intel Core i7-9700-k	299,95 €
32 GB memoria RAM DDR4 @ 3600 MHz	95 €
Samsung Galaxy Note 9	254,99 €
Coste total de uso API OpenAI	3,48 €
Total	783,42 €

Tabla 6.2: Presupuesto de equipamiento

6.3 Coste total del proyecto

Se concluye, por tanto, como coste total del proyecto, a 300 horas de trabajo con un puesto de Desarrollador de Videojuegos y con el equipo mencionado anteriormente:

Concepto	Costes
RRHH	5.049 €
Equipamiento	783,42 €
Total	5.832,42 €

Tabla 6.3: Presupuesto total del proyecto

Apéndice 1: Enlace al código y videos de ejemplo

Repositorio de GitHub

<https://github.com/alu0101325583/TFG>

Bibliografía

- [1] «Así era ELIZA, el primer bot conversacional de la historia». <https://www.xataka.com/historia-tecnologica/asi-era-eliza-el-primer-bot-conversacional-de-la-historia> (accedido 21 de mayo de 2023).
- [2] A. Vaswani *et al.*, «Attention is All you Need», en *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, y R. Garnett, Eds., Curran Associates, Inc., 2017. [En línea]. Disponible en: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
- [3] «Transformer model architecture». https://www.researchgate.net/figure/Transformer-model-architecture-this-figures-left-and-right-halves-sketch-how-the_fig1_357410305
- [4] F. Pérez Nava, I. Sánchez Berriel, J. Pérez Morera, N. Martín Dorta, C. Meier, y J. Hernández Rodríguez, «From Maps to 3D Models: Reconstructing the Urban Landscape of San Cristóbal de La Laguna in the 16th Century», *Appl. Sci.*, vol. 13, n.º 7, p. 4293, mar. 2023, doi: 10.3390/app13074293.
- [5] R. O. Ogundokun, O. C. Abikoye, A. A. Adegun, y J. B. Awotunde, «Speech Recognition System: Overview of the State-Of-The-Arts», *Int. J. Eng. Res. Technol.*, vol. 13, n.º 3, p. 384, mar. 2020, doi: 10.37624/IJERT/13.3.2020.384-392.
- [6] «Mozilla deep speech architecture». https://www.researchgate.net/figure/Mozilla-deep-speech-architecture-The-recurrent-neural-network-contains-five-layers-of_fig1_353076207
- [7] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, y I. Sutskever, «Robust Speech Recognition via Large-Scale Weak Supervision». arXiv, 6 de diciembre de 2022. Accedido: 24 de mayo de 2023. [En línea]. Disponible en: <http://arxiv.org/abs/2212.04356>
- [8] «Mozilla DeepSpeech Documentation». <https://deepspeech.readthedocs.io/en/r0.9/>
- [9] «Archive DeepSpeech repository». <https://github.com/mozilla/DeepSpeech/issues/3693>
- [10] «Coqui STT Repository». <https://github.com/coqui-ai/STT>
- [11] «[P] Pure C/C++ port of OpenAI's Whisper : MachineLearning». https://www.reddit.com/r/MachineLearning/comments/y0nvqu/p_pure_cc_port_of_openai_s_whisper/ (accedido 24 de noviembre de 2022).
- [12] «SpeechRecognizer Class Android». <https://developer.android.com/reference/android/speech/SpeechRecognizer> (accedido 22 de mayo de 2023).
- [13] «Text-To-Speech Synthesis State of the Art». <https://paperswithcode.com/task/text-to-speech-synthesis#papers-list> (accedido 22 de mayo de 2023).
- [14] «GPT-3 Wikipedia». <https://es.wikipedia.org/wiki/GPT-3>
- [15] «Fine-tune a non-English GPT-2 Model with Huggingface». <https://www.philschmid.de/fine-tune-a-non-english-gpt-2-model-with-huggingface> (accedido 1 de diciembre de 2022).
- [16] «¿Qué es el prompt engineering?» <https://www.calamoycran.com/blog/que-es-el-prompt-engineering/>
- [17] «ONNX | Home». <https://onnx.ai/> (accedido 7 de junio de 2022).

- [18] «TensorFlow.NET». <https://scisharp.github.io/tensorflow-net-docs/#/> (accedido 31 de diciembre de 2022).
- [19] «ChatGPT: Optimizing Language Models for Dialogue». <https://openai.com/blog/chatgpt/> (accedido 1 de diciembre de 2022).
- [20] «Palm 2». <https://ai.google/discover/palm2>
- [21] «Llama.cpp». <https://github.com/ggerganov/llama.cpp>
- [22] «LLamaSharp». <https://github.com/SciSharp/LLamaSharp>
- [23] «Vicuna c++». <https://huggingface.co/eachadea/ggml-vicuna-13b-1.1>
- [24] «Koala: A Dialogue Model for Academic Research». <https://bair.berkeley.edu/blog/2023/04/03/koala/>
- [25] «Bye-Bye BERT-ie: Why We Built Our Own Neural Network Language Model Instead of Using BERT or GPT». <https://www.proofpoint.com/us/blog/engineering-insights/using-neural-network-language-model-instead-of-bert-gpt>
- [26] «Microsoft BlingFire Tokenizer». <https://github.com/myl1ne/unity-geepetee>
- [27] «Netron». <https://netron.app/> (accedido 2 de diciembre de 2022).
- [28] «Speech and Text in Android and IOS Unity». <https://github.com/j1mmyto9/speech-and-text-unity-ios-android>
- [29] «Open AI API Reference». <https://platform.openai.com/docs/api-reference>
- [30] «Memory requirements? · openai/whisper · Discussion #5». <https://github.com/openai/whisper/discussions/5> (accedido 28 de marzo de 2023).
- [31] I. Samenko, A. Tikhonov, B. M. Kozlovskii, y I. P. Yamshchikov, «Fine-Tuning Transformers: Vocabulary Transfer», *Artif Intell*, vol. 317, p. 103860, 2021.
- [32] «OpenAI Fine Tuning: Preparing Data». <https://platform.openai.com/docs/guides/fine-tuning/prepare-training-data>
- [33] «OpenAI Pricing». <https://openai.com/pricing>
- [34] «Macoron/whisper.unity: Bindings of whisper.cpp speech to text model for Unity3d running on your local machine». <https://github.com/Macoron/whisper.unity> (accedido 7 de mayo de 2023).