# Reporte de avances

Dennis Masso Macías

Yuluka Gigante Muriel

# Contexto del problema

Actualmente existe una gran cantidad de asistentes virtuales que tienen, en su mayoría, las mismas funcionalidades entre sí. Sin embargo, aparte de las funcionalidades, tienen el factor común de no ser capaces de ofrecer conversaciones cuya naturalidad se asemeje a la que se podría tener con otra persona.

Además de estos asistentes convencionales, también existen otros como ChatGPT, BingChat, Bard, entre otros. Sin embargo, estos carecen de la comodidad que sí ofrecen los otros, pues su acceso tiene una mayor cantidad de pasos intermedios y, aunque ofrecen funcionalidades más avanzadas, estas no se encuentran integradas en el mismo lugar, por lo que su uso se hace más engorroso.

Es con base en esto, e inspirados por la Inteligencia Artificial de fantasía de los comics, Jarvis, que decidimos crear nuestro propio asistente virtual que combine lo mejor de ambos mundos. Por un lado, se busca la posibilidad de tener las conversaciones racionales y naturales que nos ofrecen las IA como ChatGPT, a la vez que la comodidad de la comunicación por voz y acceso rápido que podríamos tener con Syri o Google Assistant.

# Proceso de investigación

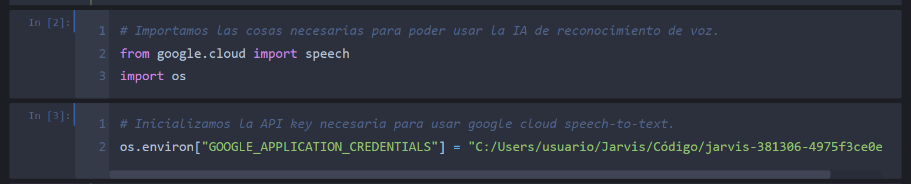
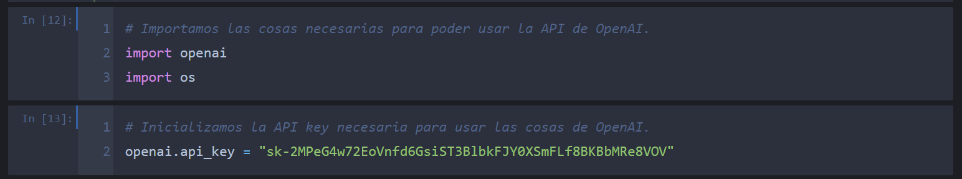
Para el desarrollo de nuestro proyecto, y puesto que perseguíamos un objetivo muy ambicioso, decidimos que la mejor opción era la de utilizar Interfaces de Programación de Aplicaciones (APIs).

Sabiendo esto, lo siguiente fue la búsqueda de APIs de reconocimiento de voz (Speech to text), conversión de texto a voz (Text to speech), y chat. Finalmente, las APIs seleccionadas fueron: Google Cloud Speech-To-Text, Google Cloud Text-To-Speech, y OpenAI API.

El paso siguiente fue documentarnos en la cuestión del funcionamiento de estas. Para esto, nuestras principales fuentes fueron las documentaciones ofrecidas por Google y OpenAI, además de foros de preguntas y respuestas como Stack Overflow. Los enlaces a las documentaciones son los siguientes:

* [Google Cloud Speech-To-Text](https://cloud.google.com/speech-to-text/docs/before-you-begin?hl=es-419): De esta, se desprenden un gran conjunto de otras guías en las que se explican distintas partes del uso de la API.
* [Google Cloud Text-To-Speech](https://cloud.google.com/text-to-speech/docs/before-you-begin?hl=es-419): Igual que con la anterior, esta guía contiene varias más guías que instruyen en el uso de otras partes de la API.
* [OpenAI API](https://platform.openai.com/docs/api-reference/introduction?lang=python): En esta documentación se incluyen las instrucciones del uso de la API para todas las funciones que ofrece OpenAI, tales como Chat, Completions, Image Creation, entre otras.

# Diseño del modelo

****Para hacer el desarrollo de nuestro proyecto, usamos Python y notebooks de Jupyter Notebook y configuramos las APIs para hacer solicitudes a las mismas.

****Además de esto, creamos distintas funciones con el objetivo de interactuar con etas APIs.

# Funcionamiento de los modelos

Para nuestro proyecto, usamos modelos para las siguientes funcionalidades: Conversión de voz a texto, Conversión de texto a voz, Generación de imágenes y Generación de texto.

El núcleo de la aplicación es el modelo de generación de texto. En nuestro caso, usamos el modelo GPT 3.5 turbo, que es una variación del modelo GPT (Generative Pre-trained Transformer) creado por OpenAI y que ofrece la posibilidad de procesar grandes volúmenes de texto en lenguaje natural.

El modelo GPT-3.5 Turbo se basa en la arquitectura Transformer, que utiliza capas de atención y mecanismos de autoregresión para procesar y generar texto. La arquitectura Transformer permite capturar relaciones a largo plazo en el texto y generar respuestas coherentes.

Es importante aclarar que, para el correcto desempeño de este modelo, es necesario su entrenamiento usando cantidades muy grandes de texto no etiquetado proveniente de libros, sitios web, foros, entre otros. Durante este proceso, el modelo aprende a predecir la siguiente palabra en una secuencia de texto, desarrollando una comprensión general del lenguaje y el conocimiento enciclopédico.

Después del pre-entrenamiento, el modelo GPT-3.5 Turbo se ajusta o adapta para tareas específicas a través del afinamiento (fine-tuning). Durante esta etapa, el modelo se entrena en conjuntos de datos específicos y se le proporciona retroalimentación para mejorar su rendimiento en tareas particulares, como la generación de texto, la traducción automática o la respuesta a preguntas.

Una vez que el modelo está entrenado y afinado, se puede utilizar para generar texto en respuesta a una entrada dada. El modelo GPT-3.5 Turbo procesa el texto de entrada y utiliza su conocimiento y patrones aprendidos para generar una respuesta coherente y relevante. El modelo tiene en cuenta el contexto previo y trata de generar texto que siga una lógica y coherencia temática.

Claro está que la facultad de tener en cuenta el contexto previo, para los modelos de este tipo, no es ilimitada, lo que explica el por qué, al hablar con ChatGPT, por ejemplo, este olvida la conversación después de un punto. Para el caso del modelo que nosotros usamos, el límite es de 4096 tokens, entendiéndose como un token una palabra, un carácter o incluso una parte de una palabra, dependiendo de cómo se haya tokenizado el texto.

En términos sencillos, el funcionamiento de este modelo se basa en la predicción de palabras subsecuentes con base las que ha predicho anteriormente para una misma oración.

En cuanto al modelo de conversión de voz a texto, el modelo comienza por procesar el audio de entrada. Si se envía un archivo de audio, se realizan etapas de preprocesamiento como el muestreo y la normalización del volumen. Si se trata de una transmisión en tiempo real, el audio se segmenta en fragmentos más pequeños para facilitar el procesamiento continuo.

A continuación, se extraen características acústicas del audio, como el espectrograma, que representa cómo varía la energía de las diferentes frecuencias a lo largo del tiempo. Lo que ayuda a capturar información importante sobre el habla, como los cambios en el tono y la intensidad.

Después, entra en juego el modelo acústico, que es una parte crucial del proceso. Este modelo usa redes neuronales recurrentes (RNN) o redes neuronales convolucionales (CNN) para analizar las características extraídas del audio y hacer predicciones sobre las unidades fonéticas subyacentes (como fonemas o subfonemas) presentes en el habla.

Tras realizar las predicciones del modelo acústico, se utiliza un algoritmo de decodificación, como el algoritmo de Viterbi, para determinar la secuencia de unidades fonéticas más probable en función de las predicciones y los modelos lingüísticos.

El modelo de lenguaje se encarga de mejorar la precisión de la transcripción al tener en cuenta las probabilidades de ocurrencia de las palabras y las relaciones entre ellas. Utiliza modelos de lenguaje basados en n-gramas o modelos más avanzados, como los modelos de lenguaje neuronales, para calcular las probabilidades de secuencias de palabras y guiar la decodificación del habla en función de la coherencia lingüística.

Finalmente, se aplican técnicas de postprocesamiento para mejorar la transcripción resultante. Estas técnicas pueden incluir la corrección de errores comunes, la eliminación de repeticiones o la capitalización adecuada de palabras. Los resultados se devuelven en forma de texto, con información adicional, como la confianza asociada a cada palabra.

Por último, en lo que concierne al modelo de conversión de texto a voz, se inicia el proceso realizando un procesamiento lingüístico del texto de entrada para comprender su estructura y significado. Esto implica analizar la gramática, la puntuación y las entidades en el texto para generar una representación interna que se utilizará para la síntesis de voz.

Seguido a esto, y utilizando técnicas de síntesis de voz, el modelo transforma el texto en habla. Google Cloud Text-to-Speech utiliza redes neuronales profundas y algoritmos de síntesis para generar audio que suena natural y coherente. Las redes neuronales están entrenadas en datos de voz grabados por hablantes humanos para aprender patrones y características de pronunciación.

Después, para mejorar la calidad de la voz generada, el modelo realiza optimizaciones adicionales. Estas optimizaciones incluyen ajustes en la pronunciación, el ritmo y la entonación del habla, así como la suavización de transiciones entre palabras para que el audio resultante sea lo más fluido y natural posible.

Por último, el modelo genera el audio en alguno de los diferentes formatos que se haya seleccionado al hacer la configuración previa a las llamadas usando la API. Ya teniendo este audio, se guarda en un archivo dentro del dispositivo y queda disponible para su uso según las necesidades.

Finalmente, en cuanto al modelo de creación de imágenes, el proceso inicia procesando una especificación de entrada (de lo que se quiere que sea la imagen), que puede ser en forma de texto o de boceto en imagen.

Después, OpenAI utiliza una arquitectura de red neuronal profunda llamada GAN (Generative Adversarial Network) para generar imágenes realistas. El modelo consta de dos partes principales: el generador y el discriminador:

* El **generador** es responsable de crear imágenes a partir de la especificación de entrada. Toma como entrada la descripción textual o el boceto proporcionado y genera una imagen correspondiente. Utiliza técnicas de aprendizaje automático, como redes neuronales convolucionales, para aprender patrones y características en los conjuntos de datos de imágenes utilizados durante el entrenamiento.
* El **discriminador** tiene la función de distinguir entre imágenes generadas por el generador e imágenes reales del conjunto de datos de entrenamiento. Este, aprende a discernir entre las imágenes generadas y las reales, proporcionando retroalimentación al generador para mejorar la calidad y la autenticidad de las imágenes generadas.

Para finalizar esta sección, la forma en que nuestra aplicación interactúa con estos modelos es mediante el uso de APIs. En nuestro caso, hacemos las configuraciones pertinentes para cada modelo, hacemos las solicitudes a estas, y usamos los outputs generados para interactuar con el usuario y mostrar los resultados.

# Desarrollo y validación

El proceso de desarrollo estuvo un poco interrumpido por algunas complicaciones con respecto al uso de las APIs y de las librerías de Python desconocidas hasta ese momento. Sin embargo, una vez comprendido el funcionamiento de todo, las cosas fluyeron de forma amena.

La última complicación que tuvimos fue con respecto a que OpenAI restiró la capa gratuita para el uso de su API, por lo que tuvimos que pagar para seguir con el proyecto.

Con respecto a la validación, todos los modelos usados responden de manera satisfactoria siempre. La transformación de voz a texto únicamente omite los signos de puntuación, pero el resto lo detecta a la perfección.

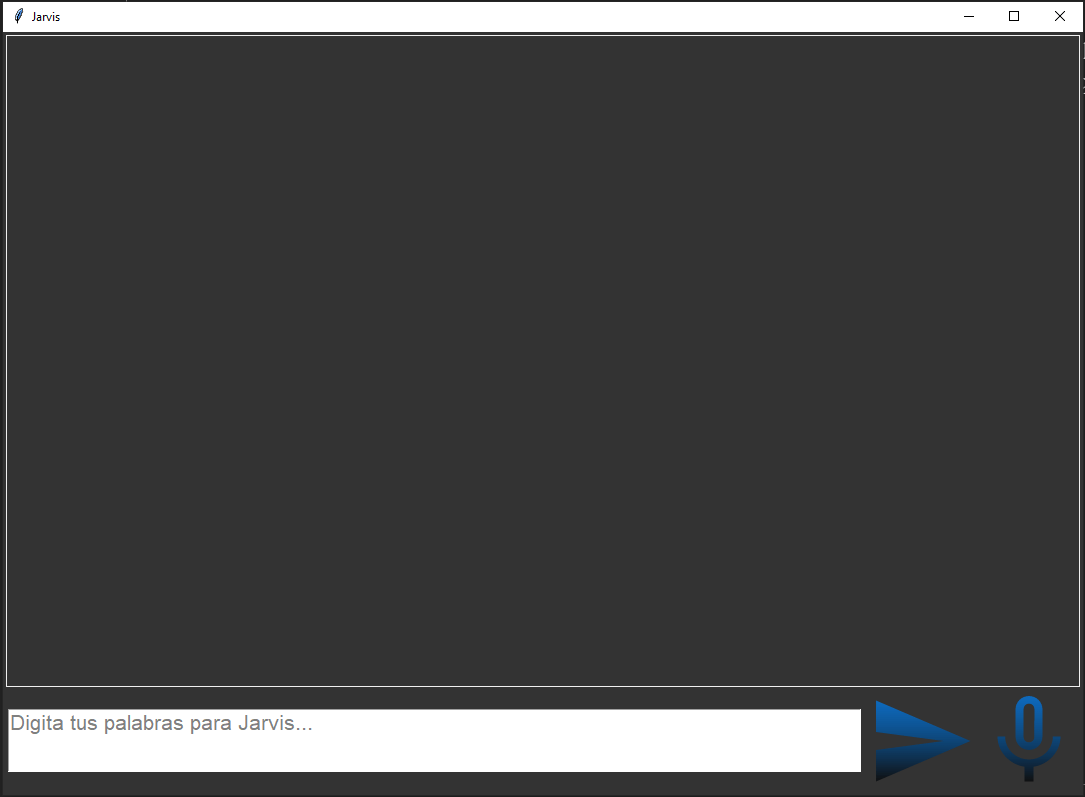
Los tiempos de espera de respuesta se incrementan un poco al usar el ejecutable en lugar de la compilación directa, sin embargo, no son nada del otro mundo y sigue siendo una aplicación por completo funcional.

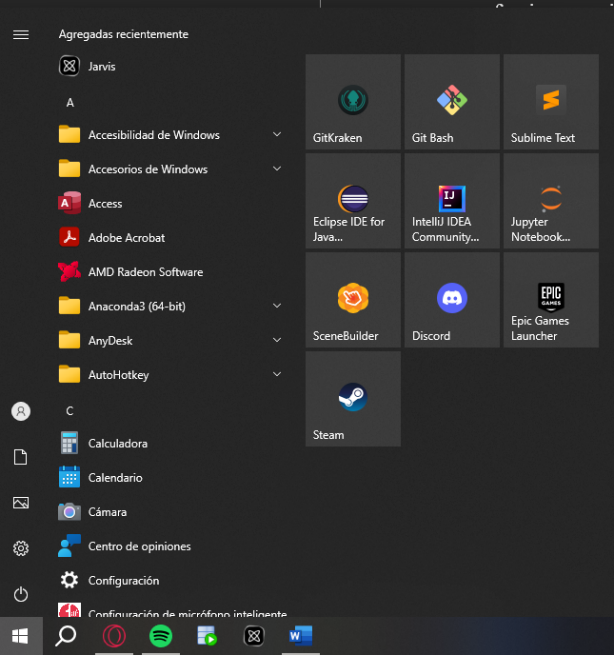
# Despliegue

Para el despliegue decidimos hacer, desde el principio, una aplicación de escritorio que ofreciese una interfaz gráfica óptima para la interacción del usuario con la aplicación.

La razón de hacer el despliegue mediante una aplicación de escritorio es la de eliminar todos los pasos intermedios necesarios para el acceso a los asistentes como ChatGPT.

Por nuestra parte, no solo dejamos la aplicación para ser ejecutada desde el compilador de Jupyter Notebook, sino que convertimos la aplicación en un ejecutable funcional que puede ser accedido como cualquier otra aplicación del sistema.





ww