Asistente virtual escalable orientado a voz: innovación en la interacción hombre-máquina

Ignacio Martín Citate Gómez[1,2], Leonardo Martín Esnaola[2], Hugo Dionisio Ramón[2],

[1] Becario de la Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CIC)
[2] Instituto de Investigación y Transferencia en Tecnología (ITT) - UNNOBA-CIC
{ignacio.citate, leonardo.esnaola, hugo.ramon}@itt.unnoba.edu.ar

Abstract. Este trabajo describe el diseño, desarrollo e implementación de un asistente virtual escalable, con una interfaz de operación orientada a comandos de voz, y capaz de entregar sus respuestas en formato de texto o audio. El aporte principal consiste en la construcción de una interfaz, que emplea distintos avances de inteligencia artificial, para exponer los servicios de cualquier API que respete el contrato de interfaz del asistente. Así, se consigue acceder a los mismos de una forma mucho más natural y cercana a la comunicación humana. Se considera de propósito general, porque la aplicación concreta está dada por el tipo de servicios ofrecidos por la API que utiliza el asistente como su interfaz de usuario.

Keywords: Asistente virtual. Comandos vocales. Inteligencia artificial.

1 Introducción

Una interfaz de usuario (UI, por sus siglas en inglés) constituye el medio a través del cual los usuarios interactúan con un dispositivo, una aplicación o un sistema informático. Cualquiera de estos puede ser muy potente en cuanto a sus capacidades, diversidad de funciones y utilidades potenciales. Sin embargo, pueden ofrecer una interfaz de usuario pobre o inadecuada, haciendo que fracasen o caigan en desuso. Por esta razón, se debe poner especial énfasis en desarrollar interfaces accesibles, es decir, amigables para personas con discapacidades; usables, implica que sean fáciles de entender, permitiendo interacciones de forma eficiente y sin frustraciones; adaptables, pudiendo adecuarse a las diversas necesidades y preferencias de los usuarios; y eficientes, permitiendo optimizar el flujo de trabajo y la simplificación de procesos complejos.

Recientemente, avances en el campo de la inteligencia artificial, como los grandes modelos de lenguaje (LLM, por sus siglas en inglés de *Large Language Model*), los modelos de reconocimiento del habla (ASR, por sus siglas en inglés de *Automatic Speech Recognition*) y otros modelos generativos, pueden ser combinados y adaptados para construir nuevas interfaces, mucho más cercanas a los usuarios, gracias a la utilización del lenguaje natural. De esta manera, la experiencia de

comunicarse con una aplicación o sistema informático no debería diferir sustancialmente de comunicarse con otro ser humano.
Los recientes y considerables avances en el campo de la inteligencia artificial abren la puerta a una amplia gama de posibilidades en cuanto a sus posibles aplicaciones. En este sentido, el más notable de ellos se dio con la aparición de los modelos de lenguaje GPT-3 [1] y GPT-4 [2], impulsados por la empresa OpenAI, que de cierta forma forzó a que otros competidores del sector, como Google y Meta, mostraran resultados y avances en este campo, presentando sus propios LLMs. En principio, se comercializaron servicios pagos para utilizar dichos modelos, pero con el paso del tiempo emergieron alternativas gratuitas y de código abierto como el modelo LLaMA [3] de Meta, que fomentaron aún más su utilización.

Por otro lado, en el campo del reconocimiento del habla, OpenAI lanzó, y publicó como código abierto, un modelo de redes neuronales llamado Whisper [4], que se encarga del reconocimiento y transcripción de audio. Además, tiene la capacidad de traducir desde diferentes idiomas hablados, lo cual también marca un progreso notable respecto de anteriores modelos que intentan realizar la misma tarea.

Con estas nuevas tecnologías a disposición, este trabajo describe el diseño, desarrollo e implementación de un asistente virtual escalable accionado por comandos vocales. Esto es, que toma el audio del usuario como medio de entrada y proporciona como salida el resultado de la ejecución del comando pretendido.

2 Estado del arte

En lo que respecta a asistentes virtuales accionados por voz, se pueden señalar a Siri, Alexa, Google Assistant y Cortana. Cada uno de ellos son soluciones comerciales provistas por grandes empresas tecnológicas como Apple, Amazon, Google y Microsoft. Evidentemente, al tratarse de software propietario y de fuente cerrada, no es posible evaluar las técnicas y/o tecnologías específicas utilizadas internamente. Sin embargo, pueden inferirse algunas cuestiones generales al diseccionar cada una de las tareas que realizan.

Reconocimiento automático del habla. El pilar del funcionamiento de estos asistentes es el reconocimiento automático del habla. Se trata del reconocimiento y transcripción del habla humana a través del procesamiento de audio.

Hasta la fecha, los últimos y más exitosos enfoques utilizan *Deep Learning*. Desde redes neuronales recurrentes, como las expuestas por *Deep Speech* [5]; redes neuronales convolucionales, como en Wav2Letter [6], Jasper [7], Wav2Vec [8]; y, finalmente, el uso de *transformers*, es el caso de Conformer [9] y Whisper. Siendo el uso de *transformers*, con el modelo Whisper, y entrenado con una masiva cantidad de datos, el que muestra el mayor rendimiento de los mencionados [4].

Procesamiento de lenguaje natural. Es la habilidad de interpretar, manipular y comprender el lenguaje humano computacionalmente y de forma automática. En este caso, con el fin de ejecutar la acción solicitada por el usuario a través de su voz.

En este tópico, el avance es el más notable, utilizándose LLMs que se apoyan en el uso de la arquitectura *transformer* y los mecanismos de atención. El ejemplo por excelencia es GPT-4, que logra un acercamiento al procesamiento del lenguaje de propósito general, en lugar de realizar una tarea específica. También cabe mencionar modelos como Gemini [10] y LLaMA, que son lanzados en respuesta a GPT-3 y GPT-4.

3 LLMs

Un LLM es un tipo de modelo de inteligencia artificial que utiliza técnicas de *deep learning* y una masiva cantidad de datos (de aquí proviene el término "*Large*") para realizar tareas de comprensión, síntesis, generación y predicción de nuevo contenido. El término inteligencia artificial generativa está muy conectado con el concepto de LLM, de hecho, un LLM es un tipo de inteligencia artificial generativa específicamente diseñada para generar contenido basado en texto, aunque ya están emergiendo LLMs de tipo multimodal, capaces de no solo generar texto, sino imágenes u otro tipo de contenido multimedia, y de interpretar estos formatos como entrada.

Están basados en la arquitectura *transformer* [11], la cual se fundamenta en los llamados mecanismos de atención. Así, se intenta replicar la atención cognitiva, calculando pesos "livianos" para cada palabra, más precisamente para el *embedding* (encaje léxico), en la ventana de contexto. Estos pesos pueden ser computados tanto de forma paralela como secuencial. Los pesos "livianos" pueden cambiar durante cada tiempo de ejecución, en contraste con los pesos "pesados", que son pre-entrenados y afinados (*fine-tuning*) para luego mantenerse fijos.

Este concepto fue desarrollado para solucionar las debilidades de los *outputs* ocultos de las redes neuronales recurrentes. Las redes neuronales recurrentes favorecen la información más reciente contenida en las palabras al final de la oración, mientras que la información al principio de la oración se atenúa. En cambio, los mecanismos de atención permiten calcular la representación oculta de un *token* en formas iguales desde cualquier parte de la oración, en lugar de tomarla del estado oculto anterior.

Los *transformers*, a diferencia de las redes neuronales recurrentes, utilizan los mecanismos previamente mencionados de forma paralela, obteniendo una mayor eficiencia respecto de la alternativa secuencial, y aprovechando el potencial de procesamiento paralelo de las GPUs.

Como fue mencionado anteriormente, estos grandes modelos de lenguaje comenzaron a estar disponibles en forma de código abierto y software libre, además de las opciones pagas. En este trabajo, se explora el uso de estas alternativas para el desarrollo de un asistente virtual.

3.1 LLaMA

LLaMA (*Large Language Model Meta AI*) es una colección de modelos que van desde los 7B a los 65B de parámetros (B: *billion* estadounidense, o mil millones). Estos modelos son entrenados con una cantidad de tokens en el orden de 10^12. Estos datos provienen de fuentes disponibles públicamente, sin el uso de conjuntos de datos propietarios. En particular, LLaMA-13B supera a GPT-3 (175B) en la mayoría de los *benchmarks*, y LLaMA-65B es competitivo contra los mejores modelos como Chinchilla-70B [12] y PaLM-540B [13].

3.2 Mistral

Mistral [14] (7B), es un LLM lanzado bajo la licencia Apache 2.0 que supera el rendimiento del mejor modelo de 13B de parámetros (LLaMA 2) en cada uno de los *benchmarks*, y a su vez al mejor modelo de 34B de parámetros (LLaMA 1) en las tareas de razonamiento, matemáticas y generación de código. Este modelo utiliza *grouped-query attention* (GQA) para aumentar la velocidad de la inferencia, en conjunto con *sliding-window attention* (SWA) para manejar de forma efectiva secuencias de tamaño arbitrario con un costo de inferencia reducido.

Para el desarrollo del asistente virtual, y luego de exhaustivas pruebas, se decidió adoptar el modelo Mistral-7B, que mostró mejores resultados en cuanto a la tarea de clasificación de texto a uno de los comandos disponibles en el asistente.

4 Whisper

El progreso en materia de reconocimiento del habla fue potenciado por el desarrollo de técnicas de pre-entrenamiento no supervisadas, como las ejemplificadas por Wav2Vec 2.0. Como estos métodos aprenden directamente desde el audio “crudo”, sin la necesidad de etiquetas, pueden utilizar grandes conjuntos de datos de habla sin etiquetar y rápidamente han escalado al millón de horas de datos de entrenamiento, considerablemente mayor que las típicas 1000 horas de un *dataset* académico supervisado. Al aplicar *fine-tuning*, este enfoque ha mejorado el estado del arte, especialmente en un escenario de pocos datos. Sin embargo, hay evidencia de que los métodos supervisados suelen otorgar mejores resultados. En esta disyuntiva, Whisper adopta un punto de vista intermedio, donde se utiliza un método llamado “supervisión débil”.

Whisper (v2) está entrenado en 680000 horas de datos supervisados multitarea y multilenguaje recolectados en la web. El uso de un conjunto de datos tan grande y diverso produce una mejora en la robustez del sistema, a pesar de los acentos, ruido de fondo y lenguaje técnico. Además, permite la transcripción en múltiples lenguajes, así como la traducción desde los mismos al inglés.

La arquitectura de Whisper se describe como *end-to-end*, implementado como un *transformer encoder-decoder*. El audio de entrada se divide en trozos de 30 segundos, para posteriormente convertirse en un espectrograma log-Mel [15] y luego pasado a un *encoder*. Se entrena un *decoder* para predecir el correspondiente texto,

entremezclado con *tokens* especiales que dirigen el modelo, para realizar tareas como identificación de lenguaje, *timestamps* por frase, transcripción multilenguaje y traducción al inglés.

Otras alternativas existentes utilizan conjuntos de datos más pequeños y acotados, o conjuntos de datos grandes, pero no supervisados. Como Whisper fue entrenado en un gran y diverso conjunto de datos que no fue afinado (*fine-tuned*) a ninguno en específico, no supera a modelos que se especializan en la famosa métrica LibriSpeech [16]. Sin embargo, cuando se mide el rendimiento de Whisper en varios y diversos conjuntos de datos, se encuentra que es mucho más robusto y comete 50% menos errores que aquellos modelos.

OpenAI explícitamente lanza el modelo Whisper como código abierto y alientan a que utilicen el mismo para crear nuevas interfaces por voz, con este enfoque más robusto que los vistos hasta el momento. Por este motivo se decidió emplearlo en este trabajo, además de ser una alternativa libre.

5 Asistente virtual

El asistente virtual presentado, se trata de una aplicación que ofrece un conjunto de comandos disponibles a los que responde por medio de entradas en formato de audio. Este audio es procesado por el asistente para reconocer cual es el comando invocado, junto con sus respectivos parámetros, según corresponda. Una vez hecho esto, el asistente procede a la ejecución particular del comando para mostrar el resultado en formato texto. A su vez, puede reproducir este texto en formato audio utilizando TTS (*Text-To-Speech*).

Nótese que el procesamiento se da en forma de *pipeline*, es decir, una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo. Esto se refleja en el diseño del asistente y la implementación realizada (Véase Fig. 1).

5.1 Arquitectura

Remarcando la idea de pipeline, se dividen las tareas del asistente virtual en distintos procesos, y se desarrolla cada uno de los mismos como módulos independientes que toman como entrada la salida del proceso anterior.

Los datos se transforman de la siguiente manera: en primer lugar, se cuenta con la entrada de audio que el usuario envía al asistente, ésta se procesa y se convierte a texto, para luego ser clasificada y analizada gramaticalmente, para la ejecución de un comando, lo que a su vez genera un resultado, que se convierte primero a texto y luego a audio, para finalmente volver al usuario. El flujo consta de: audio → texto → comando → resultado (texto y audio).

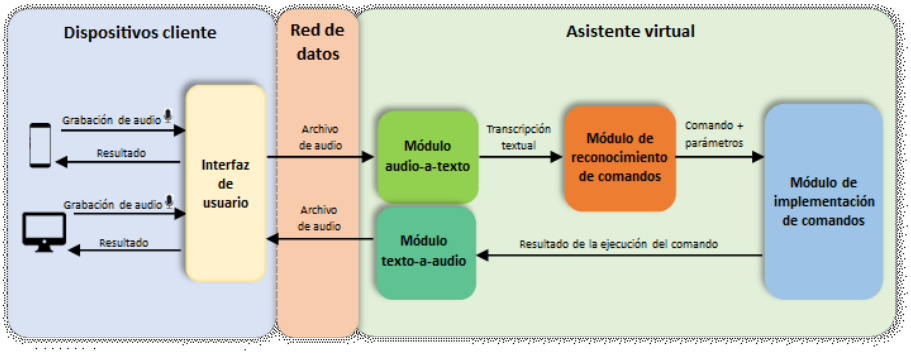

Fig. 1. Gráfico de la arquitectura y flujo de procesos del asistente virtual.

Interfaz de usuario. Representa la capa más externa del asistente, y como tal, se encarga de recibir las entradas de audio del usuario y mostrar los resultados de la ejecución de los comandos. Esta puede ser implementada como una aplicación web, móvil o de escritorio de forma indiferente. Lo importante es que se comunique con el servidor que está ejecutando la lógica propiamente dicha del asistente. Esta comunicación se da típicamente por una red de datos utilizando el protocolo HTTP (*Hyper-Text Transfer Protocol*).

Módulo audio-a-texto. Es la pieza fundamental que se encarga de transformar el audio ingresado por el usuario en texto. Como se anticipó, se utiliza el modelo de *deep learning* Whisper. Nótese que hay diferentes variantes del modelo Whisper, cada una con diferentes requerimientos. En el desarrollo de este trabajo, se utilizó el modelo "*base*" y el modelo "*medium*". Finalmente, optando por el modelo *medium*, dado que "*base"* mostró resultados erróneos con ciertas frases del español.

Se utilizó la herramienta de código abierto "whisper.cpp", una implementación en lenguaje C++ del modelo Whisper de OpenAI, más rápido y eficiente que el modelo de referencia. A su vez, whisper.cpp ofrece ser ejecutado como un servidor HTTP, lo cual facilita un desarrollo más desacoplado de este módulo, en caso de que en un futuro se desee integrar con diferentes alternativas o implementaciones para realizar esta tarea. Cabe destacar que hay que compilar whisper.cpp para la arquitectura de GPU específica que se esté utilizando, en este caso es CUDA de NVIDIA.

Whisper solo procesa audios en formato WAV, con lo cual, es necesario transformar cada archivo de audio a este formato, previo al procesamiento.

Módulo de reconocimiento de comandos. Una vez obtenida la representación textual del comando, debe procesarse para clasificar cuál de los comandos disponibles coincide con dicha representación (si es que lo hace), y cuáles son los argumentos involucrados en la ejecución del mismo que ya se encuentran dados (o no).

Para ello, se utiliza un LLM, al que se le da un *prompt* inicial describiendo la tarea de clasificación a realizar y, junto con esto, una serie de pautas que debe seguir en cuanto al formato de salida y las consideraciones a tener en cuenta para el análisis. Esto se conoce como "*prompt engineering*", y se fundamenta en instruir al modelo de

forma que interprete lo que se busca que genere como resultado, además de que lo haga de la forma más eficiente posible. La dificultad recae en que distintos modelos responden de distinta manera a la misma instrucción y, por ende, es mediante la prueba y error de diferentes instrucciones en diferentes modelos que puede obtenerse un resultado satisfactorio.

En este caso particular, se observaron mejores resultados cuando los *prompts* estaban escritos en idioma inglés. En el mismo, se le explica a la IA su propósito, es decir, que debe encargarse de clasificar comandos en base a una entrada textual aportada por usuarios, junto con un listado que contiene a cada uno de los comandos, cada uno de ellos con un identificador, una descripción, y los parámetros que debe recibir para su ejecución. Otra particularidad, es que se le pidió que su respuesta sea en formato CSV. En principio, se realizaron pruebas con el formato JSON, pero en este caso tiende a entregar respuestas malformadas o con parámetros que no existen para un comando. Esto puede deberse a que el formato CSV, implica una menor cantidad de *tokens* de respuesta para la misma cantidad de información.

Otra de las instrucciones dadas a Mistral-7B fue la solicitud de no explicar cómo llegó al resultado final, dado que, de otro modo, tiende a explicar detalladamente cómo llegó a la conclusión de que debía clasificar un comando de esa manera, y como reconoció cada uno de los argumentos del mismo (aun pidiendo que no explique su respuesta, en ocasiones tiende a hacerlo, probablemente porque fue entrenado con datos que estaban respondidos en este estilo). También, se observa que, al aportar ejemplos de la respuesta que debe dar, se reflejan mejores resultados.

Otra situación que se considera relevante ocurre cuando al usuario le falta especificar un argumento para la ejecución de un comando. En este caso, esto es detectado y posteriormente se le hace saber al usuario que debe especificar dicho argumento para continuar la ejecución. Se consideró la posibilidad de "humanizar" la interacción entre el usuario y el asistente, pidiendo a otro LLM que genere cada una de las interacciones con el usuario como último proceso previo a la muestra del resultado. Sin embargo, no se ha profundizado en esta idea, dado que no se cuenta con recursos suficientes para ejecutar tantos modelos de IA al mismo tiempo.

Se ha comentado que el desarrollo de este módulo puede realizarse de forma independiente del modelo específico de LLM utilizado. Siempre y cuando este modelo sea compatible con llama.cpp, se podrá cambiar y probar distintos modelos. Sin embargo, en el presente trabajo se ha utilizado como referencia el modelo Mistral-7B, dado que luego de realizar pruebas con otros modelos, como LLaMA2-7B, se obtuvieron mejores resultados, destacando que los modelos más grandes están fuera del alcance de los recursos computacionales disponibles para el desarrollo de este proyecto particular.

Módulo de implementación de comandos. Al recibir el resultado de clasificación del comando, junto con los argumentos detectados en formato CSV, se debe procesar para realizar la respectiva ejecución del mismo. En esta primera versión, el asistente ofrece comandos básicos, con la posibilidad de su posterior expansión. Esta primera versión tiene como objetivo principal: verificar que todo el resto del circuito funcione de forma adecuada, para generalizar la implementación de nuevos comandos para tareas específicas en cualquier tipo de rubro.

En la definición de cada comando, deben especificarse: una descripción del mismo; un número identificador; y cada uno de los parámetros que recibe, con su respectivo nombre, descripción y tipo de dato (número o *string*). Esto es importante, dado que es lo que se utiliza para generar el *prompt*, que es recibido por el módulo de clasificación de comandos. A continuación, se describen una serie de comandos incluidos en la primera versión del asistente, aunque cabe mencionar que los comandos en sí no son significativos, y constituyen tan solo un ejemplo. Cualquiera que siga las pautas descritas podría ser invocado a través del asistente:

- Saludar: no recibe ningún parámetro y su respuesta es un saludo hacia el usuario.
- Registro de lluvia: recibe como parámetro la cantidad de lluvia a registrar en milímetros, y toma datos contextuales de la ubicación para realizar la carga.
- Clima: no recibe ningún parámetro, toma datos contextuales de la ubicación para otorgar como respuesta el clima actual (temperatura, humedad, presión).
- Conversión de moneda: recibe como parámetro la moneda origen, la moneda destino y la cantidad a convertir. Por ejemplo, 20 USD a ARS.

Módulo texto-a-audio. Finalmente, al obtener el resultado de la ejecución de un comando en formato de texto, se procede a transformarlo en audio. Este proceso, que se conoce como TTS, puede lograrse de diferentes maneras. En el presente caso, se optó por utilizar una biblioteca de código Python, llamada Coqui-TTS [17]. En la misma, se puede elegir dentro de una variedad de modelos, cada uno de ellos con diferentes idiomas soportados, entre los que se encuentra el español.

Este es el último paso de la cadena de procesos del asistente virtual, una vez realizada esta transformación se envía a la interfaz gráfica el resultado en formato WAV, de modo que el usuario pueda reproducirla y seguir interactuando con el asistente de forma subsecuente.

5.2 Detalles de implementación

Los módulos que son independientes se separan en servidores HTTP distintos, dado que esto permite desacoplarse de modelos o implementaciones específicas, permitiendo en un futuro reemplazar o experimentar con otras alternativas. En este sentido, cada servidor tiene un *endpoint* destinado a su función básica. El módulo de audio-a-texto tiene un *endpoint* (*/transcribe*) que recibe un audio y lo transforma en texto. Asimismo, el módulo de texto-a-audio tiene un *endpoint* (*/tts*) que recibe texto y lo transforma a audio. Similarmente, el módulo de reconocimiento de comandos, es decir el LLM, tiene un *endpoint* (*/inference*), que da respuesta a los *prompts* de texto recibidos, además, tiene la posibilidad de almacenar en caché parte de la respuesta

(que se va a repetir, dado que la primera parte es la descripción de la tarea a realizar y solo varía el comando final, transcrito de la solicitud del usuario).

Por su parte, el módulo de implementación de comandos, se encuentra integrado con el "*core*", el cual se encarga, de recibir las solicitudes de la interfaz gráfica por medio de HTTP, para luego internamente orquestar cada uno de los pasos a seguir entre procesos, es decir, comunicarse por HTTP con el resto de módulos para luego obtener el resultado del comando y, finalmente, luego de que sea transformado en audio por el módulo texto-a-audio, responderle a la interfaz con el archivo de audio generado como respuesta.

Es razonable pensar que utilizar HTTP como medio de comunicación, entre cada uno de los módulos, puede resultar en latencia agregada, dado que involucra una solicitud de red. Sin embargo, en ciertos casos, no existe una alternativa dado que los diferentes modelos/bibliotecas están disponibles para distintos lenguajes de programación. Además, esta desventaja se reduce considerablemente, dado que cada uno de estos servidores HTTP se ejecutan en la misma máquina local, lo que reduce dramáticamente dicha latencia. A efectos prácticos, es sencillo observar que la mayor parte del tiempo de respuesta reside en el procesamiento mismo que realizan los modelos de IA, y solo una parte despreciable en las solicitudes HTTP.

5.3 Caso de uso y potencialidad

Un caso de uso destacado, y en que se ha focalizado este trabajo, es el de proveer una interfaz general de aplicación, que pueda ser extendida con nuevos comandos que incorporan funcionalidad adicional a la misma. En términos simples, el asistente proporciona una interfaz de comandos, que es extendida por desarrolladores. Los desarrolladores en cuestión, deberán crear una API HTTP que implemente un *endpoint* que será consultado por el asistente cuando se reconozca el comando en cuestión. Esta API deberá cumplir un contrato simple con el asistente. Este debe indicar:

- Una descripción textual de la funcionalidad provista (de modo que pueda ser clasificado y comparado con el resto de los comandos por el LLM).
- Los parámetros recibidos por la API (si los hubiere).

Además de ello, el formato de la respuesta de la API debe ser en un formato dado, por ejemplo, JSON.

De este modo, los desarrolladores pueden acceder a un "*frontend*" común y homogéneo, en el que se puede proveer a los usuarios de múltiples funcionalidades a través de una experiencia común. Es sencillo ver que esto es atractivo tanto desde el punto de vista del usuario, como del desarrollador.

6 Resultados y conclusiones

Los resultados expuestos a continuación se obtuvieron utilizando la placa gráfica NVIDIA GeForce GTX 1060 6GB y el modelo Mistral-7B.

Table 1. Ejemplos de comandos enviados a asistente y sus respuestas.

Prompt	Clasificación	Respuesta	Tiempo(s)
Quiero registrar cien milímetros de lluvia.	Comando: Registrar lluvia (milímetros=100.00)	Lluvia registrada: 100.00 mm	1.65
Cayeron doscientos veinte.	Comando: Registrar lluvia (milímetros=220.00)	Lluvia registrada: 220.0 mm	4.22
Mil cuatrocientos de lluvia.	Comando: Registrar lluvia (milímetros=1400.00)	Lluvia registrada: 1400.0 mm	2.16
Quiero saber cómo está afuera.	Comando: Clima	Clima actual	1.42
¿Hace calor?	Comando: Clima	Clima actual	1.37
¿Qué edad tenés?	Comando: No existe	No encontrado	1.42
Quiero convertir 200 dólares en pesos.	Comando: Conversión de moneda (de: USD, a: ARS, cant: 200.00)	280400 ARS*	2.57
Pasar 20 centavos de dólar a peso.	Comando: Conversión de moneda (de: USD, a: ARS, cant: 0.20)	280 ARS*	4.66
Euro a Dólar.	Comando: Conversión de moneda (de: USD, a: ARS, cant: nula)	Debe indicar la cantidad a convertir	3.88
1000 yuanes a dólares.	Comando: Conversión de moneda (de: CNY, a: USD, cant: 1000.00)	137.81 USD*	2.52
Mil.	Comando: Conversión de moneda (de: nulo, a: nulo, cant: 1000.00)	Debe indicar moneda origen y moneda destino.	4.07
Buenas tardes.	Comando: Saludar	Hola.	1.32
¿Cómo andás?	Comando: Saludar	Hola.	1.14

** Las conversiones son calculadas por el asistente en el momento de la ejecución al tipo de cambio correspondiente.*

En múltiples ocasiones ocurre que el modelo alucina en el formato de la respuesta y, por lo tanto, no se puede continuar con la clasificación. Esto es altamente dependiente del modelo que se utilice.

Sin embargo, en los resultados es posible observar que el asistente logra clasificar correctamente los comandos junto con sus respectivos parámetros, inclusive

detectando faltantes en los mismos con tiempos de respuesta muy razonables, considerando el hardware utilizado.

Es imperativo destacar que el desarrollo de un asistente de las características aquí expuestas (sin estar sujeto a software propietario) no sólo es posible de hacer, sino que también tiene un gran potencial de mejora a medida que el avance en el campo de IA continua a pasos agigantados.

7 Referencias

1. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.
2. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., ... & McGrew, B. (2023). Gpt-4 technical report. arXiv preprint arXiv:2303.08774.
3. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., ... & Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288.
4. Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2023, July). Robust speech recognition via large-scale weak supervision. In International Conference on Machine Learning (pp. 28492-28518). PMLR.
5. Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., ... & Ng, A. Y. (2014). Deep speech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567.
6. Collobert, R., Puhrsch, C., & Synnaeve, G. (2016). Wav2letter: an end-to-end convnet-based speech recognition system. arXiv preprint arXiv:1609.03193.
7. Li, J., Lavrukhin, V., Ginsburg, B., Leary, R., Kuchaiev, O., Cohen, J. M., ... & Gadde, R. T. (2019). Jasper: An end-to-end convolutional neural acoustic model. arXiv preprint arXiv:1904.03288.
8. Baevski, A., Zhou, Y., Mohamed, A., & Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. Advances in neural information processing systems, 33, 12449-12460.
9. Gulati, A., Qin, J., Chiu, C. C., Parmar, N., Zhang, Y., Yu, J., ... & Pang, R. (2020). Conformer: Convolution-augmented transformer for speech recognition. arXiv preprint arXiv:2005.08100.
10. Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J. B., Yu, J., ... & Ahn, J. (2023). Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805.
11. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
12. Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., ... & Sifre, L. (2022). Training compute-optimal large language models. arXiv preprint arXiv:2203.15556.
13. Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., ... & Fiedel, N. (2023). Palm: Scaling language modeling with pathways. Journal of Machine Learning Research, 24(240), 1-113.
14. Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. D. L., ... & Sayed, W. E. (2023). Mistral 7B. arXiv preprint arXiv:2310.06825.
15. Meghanani, A., Anoop, C. S., & Ramakrishnan, A. G. (2021, January). An exploration of log-mel spectrogram and MFCC features for Alzheimer's dementia recognition from

spontaneous speech. In 2021 IEEE spoken language technology workshop (SLT) (pp. 670-677). IEEE.
16. Panayotov, V., Chen, G., Povey, D., & Khudanpur, S. (2015, April). Librispeech: an asr corpus based on public domain audio books. In 2015 IEEE international conference on acoustics, speech and signal processing (ICASSP) (pp. 5206-5210). IEEE.
17. CoquiTTS 0.22.0 documentation. (n.d.). https://docs.coqui.ai/en/latest/models/xtts.html.