UAB
Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

Aquilué Rubio, Raúl; Sánchez Albaladejo, Gemma, dir. Asistente de voz local. 2024. (Enginyeria Informàtica)

This version is available at https://ddd.uab.cat/record/298991

under the terms of the  license

Asistente de voz local

Raúl Aquilué Rubio

Resumen—Este proyecto tiene como objetivo desarrollar un asistente virtual[2] local que convierta consultas en lenguaje natural[3] a SQL y recupere respuestas de una base de datos, asegurando la privacidad y seguridad de los datos. Se ha creado una página web basada en Laravel, centrada en la funcionalidad del backend, que opera en un servidor Proxmox[9] con Cloudflare y MySQL. Las características clave incluyen pruebas de STT[4] y TTS[5] en línea y locales, consultas manuales a la base de datos, historial de audios de STT y TTS, y evaluación de consultas y respuestas. El asistente admite interacciones por voz y texto, utilizando WhisperAI[6] para la transcripción de voz a texto local y CoquiTTS[5] con el modelo XTTS v2 para texto a audio. La conversión de lenguaje natural a SQL se realiza mediante el modelo CodeQwen-7B a través de Ollama[17]. El informe detalla la creación del proyecto, la elección de componentes y evaluaciones basadas en más de 1000 consultas de prueba.

Palabras clave—Asistente de voz, privacidad de datos, Texto a Voz (TTS), Voz a Texto (STT), Whisper AI, Coqui TTS, Laravel, MySQL, Proxmox, lenguaje natural, consultas SQL, privacidad de datos, modelos de lenguaje grandes (LLM), procesamiento local, Cloudflare.

Summary — This project aims to develop a local virtual assistant[2] that converts natural language[3] queries to SQL and retrieves responses from a database, ensuring data privacy and security. A Laravel-based website was created, focusing on backend functionality, operating on a Proxmox[9] server with Cloudflare and MySQL. Key features include online and local STT[4] and TTS[14] tests, manual database queries, STT and TTS audio history, and query and response evaluation. The assistant supports voice and text interactions, using WhisperAI[6] for local voice-to-text transcription and CoquiTTS[14] with the XTTS v2 model for text-to-audio. Natural language to SQL conversion is done via the CodeQwen-7B model through Ollama[17]. The report details the project creation, component choices, and evaluations based on over 1000 test queries.

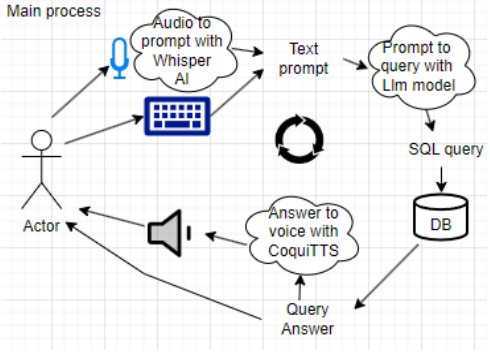
Keywords — Voice assistant, data privacy, Text-to-Speech (TTS), Speech-to-Text (STT), Whisper AI, Coqui TTS, Laravel, MySQL, Proxmox, natural language, SQL queries, data privacy, large language models (LLM), local processing, Cloudflare.

1 Introducción - Contexto del trabajo

En la actualidad, los asistentes de voz se han convertido en herramientas comunes para realizar una variedad de acciones cotidianas, desde controlar dispositivos del hogar hasta gestionar agendas personales. Estos dispositivos, como smartphones o tablets, operan mediante la transmisión de consultas de los usuarios a servidores remotos pertenecientes a grandes empresas como Google o Amazon, donde se procesan y generan las respuestas adecuadas. Sin embargo, este proceso de transmisión de datos plantea preocupaciones significativas en cuanto a la privacidad y seguridad de la información[1], ya que los datos enviados a los servidores de estas compañías pueden ser utilizados para diversos propósitos más allá de la mera asistencia al usuario.

Este proyecto se propone desarrollar un asistente virtual[2] que opere de manera completamente local, eliminando la necesidad de enviar datos a terceros. El asistente permitirá a los usuarios interactuar en lenguaje natural[3] para realizar consultas SQL a una base de datos, ofreciendo respuestas tanto por texto como por voz.
El esquema del asistente se muestra en la [Fig.1].
Al mantener todas las operaciones dentro del entorno local, se garantiza una mayor seguridad y control sobre la información.

En los apartados 2, 3 y 4 se describe detalladamente la gestión del proyecto y el desarrollo del asistente virtual.
La integración de las funcionalidades en una plataforma web accesible se describe en el apartado 5.1 y 5.2.
La implementación de las tecnologías de reconocimiento y síntesis de voz se detallan en los subapartados de 5.3 y 5.4 y la transformación de lenguaje natural a SQL en el apartado 5.5.
Por último, se muestran distintas pruebas realizadas, así como un análisis de los resultados en el apartado 6 y consideraciones futuras para el proyecto en el apartado 7.



[Fig.1] Diagrama de la interacción esperada con el asistente.

2 Objetivos del proyecto

El objetivo principal del proyecto es desarrollar un asistente virtual que pueda interpretar consultas en lenguaje natural y convertirlas en consultas SQL, ejecutándose sobre una base de datos local. Todo esto se quiere realizar en un entorno local para eliminar la necesidad de enviar información a terceros a través de internet. Se busca que el programa sea aplicable en entornos domésticos o con datos importantes, además de ser capaz de operar con recursos limitados.

2.1 Planteamiento

Los objetivos del proyecto, en orden de importancia son los siguientes:
1-Crear una plataforma web para acceder a los diferentes servicios desarrollados.
2-Desarrollar un modelo de transcripción de consultas en lenguaje natural escrito a consultas SQL.
3-Crear un sistema de interacción de consultas SQL del asistente con la base de datos.
4-Evaluar sistemas de transcripción de lenguaje natural oral a escrito.
5-Evaluar sistemas de transcripción de texto a audio.
6-Desarrollar una plataforma web para realizar pruebas
7-Realizar pruebas para validar la eficacia del sistema.
Al cumplir estos objetivos, el proyecto garantizará la creación de un asistente virtual eficiente y seguro, capaz de gestionar consultas en lenguaje natural y ofrecer respuestas precisas desde una base de datos local, sin comprometer la privacidad de los datos del usuario.

2.2 Ejecución

Para llevar a cabo este proyecto, se han desarrollado módulos de reconocimiento de voz, clasificación de texto y síntesis de texto, además de una interfaz web que permite la interacción con el usuario. Cada uno de estos módulos está diseñado para funcionar de manera eficiente incluso en ordenadores con pocos recursos, facilitando su implementación en diversos entornos sin requerir infraestructura avanzada. A continuación, se describen los módulos y su funcionamiento:

2.2.1 Transformación de lenguaje natural a SQL

Este módulo convierte las consultas en lenguaje natural a SQL. Utiliza el modelo CodeQwen-7B mediante el programa Ollama. Este modelo se ha seleccionado tras la evaluación de diversas alternativas, tanto locales como online, incluyendo LLaMA3, Phi 3 Mini, Mistral y ChatGPT.

2.2.2 Módulo de Speech-to-Text (STT) [4]

Este módulo convierte la voz del usuario en texto. Se ha optado por utilizar Whisper[6] en local para esta tarea debido a la posibilidad de usar diversos tamaños del modelo, permitiendo así un mayor manejo de los recursos. Además, se encontró posicionado como uno de los mejores modelos de STT en local.

2.2.3 Módulo de Text-to-Speech (TTS)[5]

Este módulo convierte las respuestas textuales del asistente en voz. Se han probado diversos modelos locales, pero ninguno ofrecía un rendimiento similar a modelos online como los de Google. Al final, se optó por usar CoquiTTS aunque su rendimiento para texto en español no fuese tan óptimo como en inglés.

2.2.4 Interfaz web

Proporciona un punto centralizado de interacción, permitiendo a los usuarios interactuar con el asistente, realizar pruebas de funcionalidades y consultar el historial de interacciones. Está desarrollada en Laravel y operativa en un servidor Proxmox con conexión mediante Cloudflare. Esta configuración permite una operación segura y eficiente, gestionando las consultas y respuestas del asistente sin comprometer la privacidad de los datos.

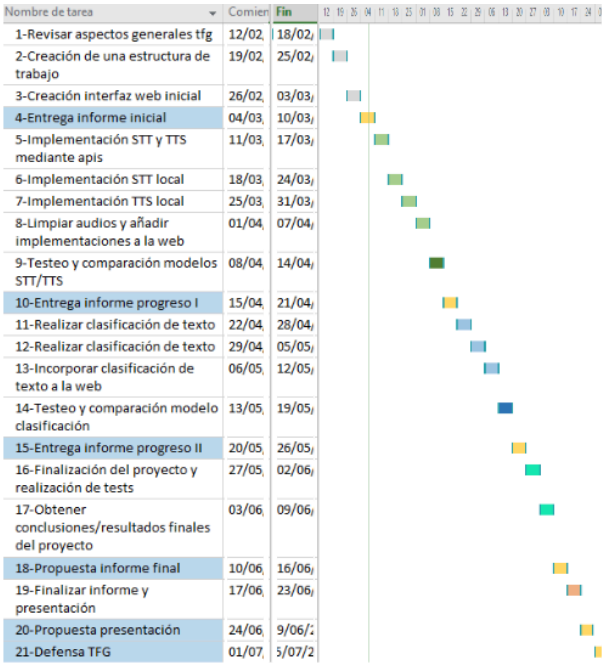
Además de estos módulos, se realizaron estudios comparativos de proyectos y algoritmos existentes, analizando sus diseños y prestaciones.

3 Metodología a seguir

Para el desarrollo de este proyecto, se ha adoptado una metodología ágil que facilita la iteración y mejora continua del producto[7]. Esta aproximación permite desarrollar y perfeccionar el proyecto progresivamente, obteniendo resultados preliminares desde las fases iniciales y generando métricas para comparar las sucesivas iteraciones. Así, se puede evaluar el avance y realizar los ajustes necesarios para asegurar la calidad y el correcto funcionamiento del asistente de voz.
La elección de la metodología ágil es estratégica, ya que posibilita centrar la atención en distintas áreas del proyecto de manera secuencial, lo que permite comprender cómo cada componente influye en el sistema global. Esto se puede hacer sin necesidad de tener un producto final completamente desarrollado, lo que agiliza el proceso de aprendizaje y adaptación.
Para la gestión del proyecto, se pretendía utilizar software especializado como Trello, una herramienta que facilita la organización del flujo de trabajo mediante la creación de tareas y la definición de sus estados (pendiente, en proceso, completada, etc.). Finalmente, se optó por realizar un documento en google drive semanalmente con las tareas a realizar para la semana. Se estableció un seguimiento semanal y se reevaluaron las tareas en cada reunión para garantizar que el proyecto se mantuviese en curso.
Además de aplicar la metodología ágil, se llevó a cabo una documentación detallada del progreso del proyecto. Esto incluye el registro de cambios, nuevas adiciones y conocimientos adquiridos durante el desarrollo. Toda la documentación se organizó y almacenó en un repositorio de GitHub y en Google Drive, asegurando un acceso y gestión eficientes de la información.

4 Planificación

El desarrollo de este proyecto se ha estructurado en torno a una serie de entregas programadas a lo largo de su ejecución. Para gestionar eficazmente estos hitos, se estableció una planificación semanal con el objetivo de alcanzar metas específicas en cada uno de los puntos de entrega.
Aunque la planificación se ha diseñado para ser flexible, se ha enfocado en la obtención de resultados concretos cada semana. Esta flexibilidad permite reevaluar y ajustar la planificación según sea necesario, en función del progreso del proyecto y de los resultados obtenidos. Este enfoque iterativo y adaptable es coherente con la metodología ágil adoptada y ha facilitado la gestión del tiempo y los recursos, asegurando que el proyecto se mantenga alineado con los objetivos establecidos.
La planificación general se ve en la [Fig.2]; en el apartado siguiente se desarrolla cada una de sus fases.



[Fig.2] Diagrama de Gantt del TFG con los puntos de entrega marcados en amarillo y las distintas fases del proyecto en diversos colores según la similitud entre ellas.

4.1 Seguimiento de la planificación

4.1.1 Tareas (1,2,3 y 4)

Estas tareas engloban el inicio de la planificación del TFG, junto con la redacción de los requisitos, objetivos y bases del proyecto. En estos apartados también se redactó la estructura del trabajo y se definieron las tareas a realizar. Por último, se hizo una web básica y se explicó todo en el informe inicial.

4.1.2 Tareas (5,6,7,8,9 y 10)

Estas tareas engloban la creación de TTS y STT. Se implementó un entorno de pruebas y se buscaron distintas formas de implementar el sistema de transcripción de audio, eligiendo varias opciones para su implementación y comparación. También se buscaron opciones para elegir e implementar un modelo en local. Todo esto se aplicó en un entorno de pruebas de la web ya funcional.

4.1.3 Tareas (10,11,12,13,14 y 15)

Se revisaron distintos métodos de clasificación de texto y se decidió implementar el uso de modelos de LLM en local. Estos, són modelos de aprendizaje con arquitecturas basadas en transformers[8]. Se reestructuró la web para adaptarla a la nueva complejidad que implicaba el manejo de los diferentes módulos, y se implementó un modelo de conversión de lenguaje natural a consulta SQL. También se rehizo la base de datos para incorporar muchos nuevos datos y permitir así realizar tests más complejos. Se incorporó el modelo a la web y se integró con el sistema de STT y TTS.

4.1.4 Tareas (16,17 y 18)

En estos apartados se rehizo toda la funcionalidad de tests para poder comparar diversos modelos de LLM.
Se crearon scripts para obtener métricas de rendimiento de los diferentes apartados de la web y se corrigieron los errores restantes del proyecto.

4.1.5 Tareas (19,20 y 21)

Se finalizó el informe del proyecto y se creó la presentación.

4.1.6 Consideraciones extra

Durante la creación del proyecto, hubo varios contratiempos como la pérdida de parte del trabajo o dificultades no consideradas en la planificación, como fallos en el rendimiento de la máquina o la creación de una base de datos de test mucho más compleja de lo esperado. Pese a esos retrasos, el proyecto se ha podido completar sin dejar tareas pendientes, aunque se retrasaron algunas entregas.

5 Desarrollo

5.1 Creación del servidor

Para el desarrollo del proyecto, se ha establecido un servidor en una máquina virtual utilizando Proxmox[9] como solución de virtualización. Proxmox se eligió por ser una plataforma robusta y flexible que permite gestionar múltiples máquinas virtuales (VMs) y contenedores, facilitando la separación de servicios y asegurando un entorno modular. Aunque se consideraron alternativas como VMware y VirtualBox, Proxmox ofreció ventajas significativas en términos de escalabilidad y gestión centralizada, cruciales para este proyecto.

Se configuraron varias máquinas virtuales para distintos servicios, tal y como se muestra en la [Fig.3]:

- **VM para la base de datos MySQL**: Gestiona todos los datos del proyecto, proporcionando un entorno seguro y optimizado para el

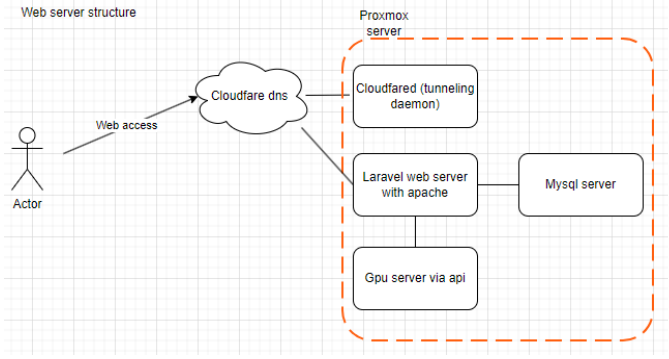
almacenamiento y consultas de datos.

- **VM para la conexión con Cloudflare**: Facilita el acceso al servidor desde un entorno local con IP dinámica, garantizando una conexión segura y estable.
- **VM para la ejecución de la web**: Aloja la interfaz web del proyecto, desarrollada en Laravel. Esta VM se configuró inicialmente solo con CPU, aunque se identificó que el uso de una GPU Nvidia con CUDA Cores [10] mejoraría significativamente los tiempos de procesamiento para las funcionalidades de transcripción de voz a texto (STT) y texto a voz (TTS), así como el uso del modelo LLM.

La separación de servicios en distintas VMs proporciona varios beneficios:

- **Aislamiento de servicios**: Mejora la seguridad y facilita el mantenimiento, ya que los problemas en una VM no afectan a las demás.
- **Escalabilidad**: Permite ajustar los recursos asignados a cada servicio de manera independiente.
- **Facilidad de gestión**: Simplifica la administración y el monitoreo de los recursos del servidor.

Además, se ha prestado especial atención a la gestión segura del acceso remoto al servidor. Para ello, se implementaron prácticas de seguridad informática, como el uso de conexiones cifradas a través de SSH (Secure Shell) para el acceso al servidor, la autenticación mediante llaves públicas y privadas para reforzar la seguridad, y la configuración de un firewall para controlar el tráfico de entrada y salida. Estas medidas aseguran que solo los usuarios autorizados puedan acceder al servidor y que la información se mantenga protegida.



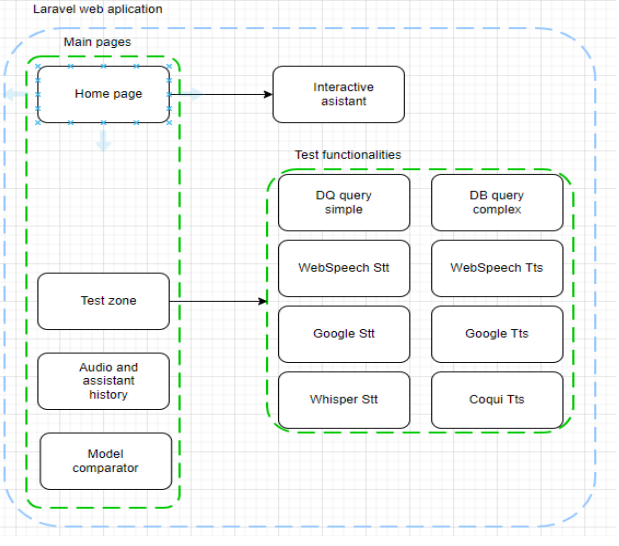
[Fig.3] Esquema de las máquinas del proyecto, así como la conexión de la web al usuario.

5.2 Implementación de la interfaz web y la base de datos

En el desarrollo del proyecto del asistente de voz local,se optó por crear una interfaz web robusta y funcional utilizando Laravel [11], un framework de PHP para el desarrollo de aplicaciones web. Laravel se eligió por su arquitectura expresiva y su capacidad para manejar operaciones complejas de backend de manera eficiente. La programación de la interfaz se realizó principalmente en PHP, aprovechando las características de Laravel que facilitan el trabajo con patrones de diseño MVC (Modelo-Vista-Controlador) y ofrecen una amplia gama de funcionalidades integradas.
Para la parte estética de la interfaz web, se utilizó Bootstrap, un framework de diseño de front-end que permite crear interfaces de usuario limpias y responsivas. Bootstrap proporciona un conjunto de componentes de diseño que se han utilizado para garantizar que la interfaz sea visualmente atractiva y fácil de navegar, manteniendo la coherencia en los diferentes elementos de la interfaz.
Además, se desarrollaron ejecutables en Python para manejar las funcionalidades específicas del asistente de voz, como el procesamiento de lenguaje natural y la integración con las APIs de Texto a Voz y Reconocimiento de Voz.

Para facilitar las pruebas y el desarrollo, se implementó una base de datos MySQL. Esta base de datos se utilizó para la creación de un entorno de pruebas con múltiples tablas y datos falsos, así como para la gestión de los resultados de los tests y pruebas de los módulos de TTS, STT y consultas SQL generadas por el sistema. MySQL es un sistema de gestión de bases de datos relacional que se integra bien con Laravel y proporciona las herramientas necesarias para gestionar los datos de manera eficiente.

Se puso especial énfasis en el backend, integrando funcionalidades críticas y asegurando que los scripts de STT, TTS y transcripción de consultas, desarrollados en Python, fueran independientes de la VM de la web. Esto significa que estos scripts podrían ejecutarse en otros servidores y ser accesibles mediante API, ofreciendo flexibilidad y escalabilidad al proyecto.
La [Fig.4] muestra el esquema de interacción en la web con diversos servicios.



[Fig.4] Esquema de la web del proyecto.

5.3 Funcionalidad de Reconocimiento de Voz (STT)

Además de la integración de las funcionalidades de Texto a Voz (TTS), el proyecto también ha incorporado capacidades de Reconocimiento de Voz (Speech-to-Text, STT) utilizando las APIs de Google y Mozilla, así como un modelo local. Esta implementación complementa la funcionalidad del asistente de voz, permitiendo no solo sintetizar voz a partir de texto, sino también convertir el habla de los usuarios en texto escrito, lo que facilita una interacción bidireccional más natural y efectiva.

Al elegir el modelo de STT, se tuvieron en cuenta factores como la facilidad de integración, la velocidad de procesamiento y la capacidad de manejar errores comunes en el reconocimiento de voz. Se eligió usar WhisperAI[6] por su facilidad de ejecución en entornos con bajos recursos y su buen funcionamiento.

5.3.1 Google API

Para tener una comparativa, se realizó una implementación del modelo en línea de pago de Google. Se utilizó Google Cloud Speech-to-Text, con una configuración similar a la API de Text-to-Speech. Se creó un entorno en Google Cloud Platform, se activó la API de Speech-to-Text y se gestionaron las credenciales necesarias para su uso. La implementación en el proyecto requirió la instalación de la biblioteca cliente correspondiente y la integración en el código fuente, permitiendo que el asistente de voz procese el audio capturado y lo convierta en texto.

La API de Google se utilizó como punto de referencia para comparar con otros modelos de STT. Google API es conocida por su alta precisión y rapidez, proporcionando un estándar de calidad para las pruebas iniciales del proyecto. Aunque es efectiva, su dependencia de una conexión a internet no cumple con el objetivo del proyecto de operar completamente en local.

5.3.2 Webspeech API

También se decidió integrar la API de Mozilla WebSpeech, aprovechando su compatibilidad con los navegadores modernos. Esta API permite el procesamiento del habla directamente en el navegador, lo que reduce la latencia y mejora la privacidad al no tener que enviar datos a servidores externos. La implementación consistió en escribir funciones JavaScript que capturan el audio del usuario y lo transforman en texto, ajustando las configuraciones para optimizar el rendimiento y la precisión según las necesidades del proyecto.

WebSpeech API, integrada en muchos navegadores modernos, también se usó como punto de control adicional. Ofrece resultados comparables a Google API, pero presenta las mismas limitaciones en cuanto a la necesidad de una conexión a internet y la dependencia de servicios externos.

5.3.3 STT local

Para el modelo de reconocimiento de voz local se decidió utilizar WhisperAI, un modelo de código abierto y preentrenado desarrollado por OpenAI [12].

La elección de WhisperAI para la implementación local del reconocimiento de voz se hizo para mantener la privacidad y la seguridad de los datos, al procesar la información directamente en el dispositivo del usuario sin necesidad de enviar datos a servidores externos.

La integración de WhisperAI en el proyecto se llevó a cabo mediante la configuración de un entorno de ejecución local en el servidor web, utilizando ejecutables en Python y aprovechando la compatibilidad de este lenguaje con las bibliotecas del modelo. Esto permite que el asistente de voz procese y convierta el habla capturada en texto de manera eficiente y precisa.

Para la funcionalidad de STT local, se consideraron varias alternativas como Vosk y DeepSpeech. Sin embargo, WhisperAI fue seleccionado debido a su superior precisión y capacidad para manejar diferentes acentos y variaciones del español u otros idiomas.

5.4 Funcionalidades de Voz a Texto (TTS)

En el desarrollo del módulo de Text-to-Speech (TTS), se optó por integrar dos APIs para convertir texto escrito en habla sintetizada [13]. Las APIs seleccionadas son la API de Google Text-to-Speech y la API de Mozilla WebSpeech. Estas ofrecían en su versión gratuita voces robotizadas, pero muy funcionales para la lectura de textos. Para el modelo en local se usó CoquiTTS[14], que aunque su transcripción en español no era ni rápida ni sonaba especialmente bien, fue la mejor opción disponible, sobre todo teniendo en cuenta que funciona excepcionalmente bien en inglés.

5.4.1 Google API

La API de Google Text-to-Speech se configuró a través de Google Cloud Platform. Similar al STT, la API de TTS de Google se utilizó para establecer un punto de referencia debido a su alta calidad de audio y soporte para múltiples idiomas y voces. Aunque es altamente efectiva, su dependencia de internet y los costos asociados fueron factores limitantes.

5.4.2 WebSpeech API

Por otro lado, la API de Mozilla WebSpeech, al ser una tecnología de estándar abierto, se integró directamente en la interfaz web del proyecto sin necesidad de configuraciones adicionales. La implementación se realizó mediante código JavaScript, permitiendo el manejo de la síntesis de voz en el lado del cliente y ofreciendo opciones de personalización de la experiencia de usuario. Se prestó especial atención a la compatibilidad entre navegadores durante las pruebas de la API.

WebSpeech API también se probó para TTS. Ofrece una

buena calidad de síntesis, pero al igual que en STT, requiere una conexión a internet y depende de servicios externos, lo que no es ideal para los objetivos del proyecto.

5.4.3 TTS local

Para la implementación local de la funcionalidad de Texto a Voz (Text-to-Speech, TTS) en nuestro proyecto de asistente de voz, se seleccionó Coqui TTS, un proyecto de código abierto desarrollado por ex trabajadores de Mozilla. Coqui TTS se destaca por su capacidad para clonar voces y por ofrecer soporte para múltiples idiomas, incluyendo un rendimiento notable en español, lo cual es crucial para este proyecto dado que muchos modelos de TTS no ofrecen un buen desempeño en este idioma.

La elección de Coqui TTS se basó en la disponibilidad de un modelo pre entrenado XTTS v2[15] que incluye soporte para el español, permitiendo una integración más fluida y efectiva sin la necesidad de dedicar recursos extensos al entrenamiento desde cero. Esta característica es especialmente valiosa en contextos donde se requiere una implementación rápida y eficiente, y donde la calidad del habla generada es un factor crítico. El modelo XTTS v2 encontrado en Hugging Face es una herramienta avanzada de generación de voz que permite clonar voces en diferentes idiomas utilizando solo un breve clip de audio de 6 segundos. Además, el modelo soporta 17 idiomas, incluido el español.

Coqui TTS permite una personalización avanzada y la posibilidad de clonar voces, lo que ofrece oportunidades para crear experiencias de usuario más naturales y agradables. Esta capacidad de clonación de voces es particularmente útil para aplicaciones como asistentes de voz, donde una voz consistente y personalizable puede mejorar significativamente la interacción del usuario.

La implementación de Coqui TTS en el proyecto se ha realizado de manera local, asegurando que toda la conversión de texto a voz se procese en el servidor sin necesidad de enviar datos a la nube. Esto no solo mejora la respuesta del asistente de voz, sino que también fortalece la privacidad y seguridad de los datos del usuario, alineándose con los objetivos de privacidad del proyecto.

Para la implementación de TTS local, se consideraron alternativas como Tacotron y WaveNet. Finalmente, se eligió Coqui TTS con el modelo pre entrenado para múltiples idiomas (XTTS v2) debido a su equilibrio entre calidad de audio y rendimiento. Coqui TTS proporciona una calidad de síntesis decente en español y puede ejecutarse en hardware local, cumpliendo con los requisitos del proyecto.

5.5 Lenguaje natural a consulta SQL

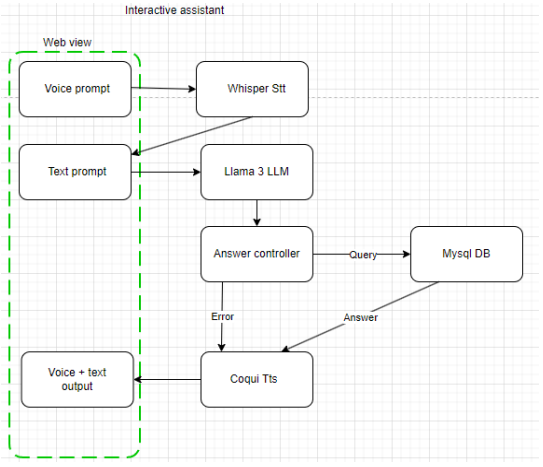
Para convertir lenguaje natural a consultas SQL, se utilizó el modelo LLM codeqwen7b a través de Ollama[16]. Ollama fue elegido debido a su facilidad de integración y eficiencia en la gestión de modelos de lenguaje, en comparación con otras alternativas como Langchain y Hugging Face Transformers que, aunque funcionales, presentan mayor complejidad en su configuración y mantenimiento.

Ollama también funciona como un servidor independiente con llamadas API, lo que permite una fácil integración con otros sistemas. El modelo se mantiene en memoria para evitar ralentizaciones, aunque su ejecución en un servidor con solo CPU resultó lenta. Para mejorar el rendimiento, se sugiere utilizar una GPU Nvidia con CUDA Cores, lo que aceleraría significativamente los tiempos de procesamiento.

Durante el desarrollo, se probaron varios modelos LLM y diferentes configuraciones. Entre las alternativas evaluadas se encontraban Llama3 y Phi3 mini, pero codeqwen7b se destacó por su precisión y capacidad de procesamiento, siendo el más adecuado para las necesidades del proyecto.

5.6 Implementación con el asistente

En la interfaz web se usaron los modelos de STT y TTS para facilitar la interacción con el asistente. Esto se hizo siguiendo el diagrama de la [Fig.5], donde se permite al usuario introducir un prompt para el asistente o narrarlo por voz y posteriormente se transcribe. Con la consulta en lenguaje natural escrito se procede a generar la consulta SQL mediante el modelo LLM elegido y se analiza el resultado. Si es apto para ejecutarse se realiza la consulta en la base de datos y se devuelve la respuesta a la interfaz. Por último, está la opción de narrar la respuesta por audio, pese a que con el uso de TTS local los tiempos de generación del audio sean bastante largos.



[Fig.5] Esquema de la funcionalidad de asistente web.

6 Test y resultados

6.1 Descripción de los Tests Realizados

6.1.1 Objetivo de los Tests

El objetivo principal de los tests es verificar la capacidad del asistente virtual para interpretar correctamente consultas en lenguaje natural, traducirlas a SQL y obtener respuestas precisas y relevantes de la base de datos. Además, se busca evaluar el rendimiento, precisión, robustez y usabilidad del sistema.

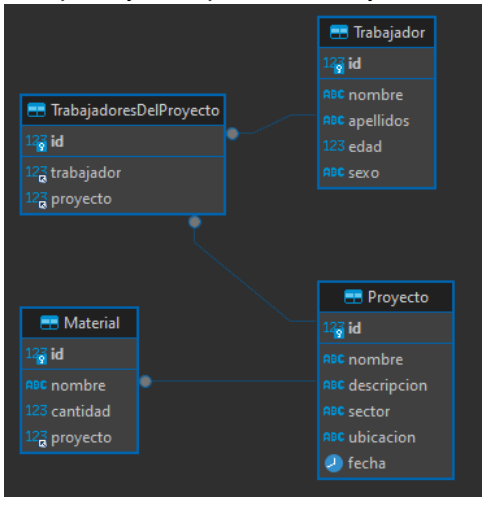
6.1.2 Tipos de Tests

· **Tests Funcionales**: Se enfocan en asegurar que el asistente pueda manejar adecuadamente una variedad de consultas en lenguaje natural y traducirlas correctamente a SQL.
· **Tests de Rendimiento**: Miden el tiempo de respuesta del sistema, desde la entrada de la consulta en lenguaje natural hasta la obtención de la respuesta.
· **Tests de Precisión**: Evalúan la exactitud de las respuestas proporcionadas por el asistente, comparándolas con las respuestas esperadas.
· **Tests de Robustez**: Verifican cómo el sistema maneja consultas ambiguas, incompletas o mal formuladas debido a la ambigüedad del lenguaje natural.

Se realizaron tests para comparar la eficacia de distintos modelos LLM, tanto online como offline.
Estos consistieron en introducir una frase indicando la información deseada de la base de datos, generar la consulta SQL adecuada y ejecutarla. Tras la ejecución, se evaluaban los datos obtenidos y la consulta SQL generada. También se midieron los tiempos de ejecución de los modelos locales.

6.2 Metodología de los Tests

Tal y como se muestra en la [Fig.6], se creó una base de datos con 4 tablas relacionadas: Material, Proyecto, Trabajador y TrabajadoresDelProyecto.



[Fig.6]Esquema de la base de datos a la que realizar las consultas generadas por distintos llm.

En estas tablas se añadieron manualmente más de 500 registros variados, representando distintas entidades y relaciones de proyectos y trabajadores, así como materiales del proyecto.

6.2.1 Conjunto de datos de prueba

Se generó una batería de más de 1140 consultas en lenguaje natural clasificadas en 19 categorías según el tipo de consulta SQL esperada. Cada consulta tiene tres versiones: formal, informal e informal en inglés; por lo que se generaron 20 consultas en lenguaje natural por cada categoría a evaluar (19 tipos * 3 versiones * 20 frases = 1140 consultas). Se utilizó un esquema predefinido para crear estas consultas, asegurando que cubren diversos escenarios relevantes a la base de datos.

Los 19 tipos distintos de consultas se pueden clasificar de la siguiente forma:
·Consultas básicas y selección de datos.
·Consultas con múltiples tablas y subconsultas.
·Consultas agregadas y de agrupamiento.
·Consultas de modificación de datos.
·Consultas con funciones y expresiones.
·Consultas con ordenamiento y límites.
·Consultas de eliminación de duplicados y combinación de condiciones.

6.2.2 Automatización de pruebas

Se desarrollaron scripts para automatizar la gestión de datos en la base de datos y la ejecución de las consultas, asegurando la integridad de la base de datos y facilitando la evaluación de la funcionalidad de las consultas.

El objetivo de esta metodología era obtener un conjunto diversificado de pruebas que permitieran evaluar la precisión y la robustez de los modelos en diferentes escenarios. Cada consulta se evaluó manualmente al menos una vez, y se creó un sistema para detectar respuestas correctas automáticamente, comparando con resultados previamente evaluados como correctos, reduciendo la necesidad de corrección manual constante.

6.3 Ejecución de los Tests

6.3.1 Automatización del proceso

Se crearon scripts en Python para insertar y eliminar datos en la base de datos y para ejecutar las consultas de manera automatizada. Las consultas SQL, después de ser generadas, se ejecutaban en la base de datos para comprobar sus resultados, pasando previamente por diversos filtros para verificar la seguridad de la base de datos.

6.3.2 Ejecución de las consultas

Cada consulta fue ejecutada por el asistente, verificando la correcta traducción a SQL y la precisión de la respuesta obtenida. Se evaluó la capacidad de los modelos para generar consultas SQL válidas y su exactitud en los resultados esperados.

6.3.3 Evaluación de los resultados

Se midió el tiempo de respuesta y se compararon las respuestas obtenidas con las esperadas para evaluar la precisión. Se analizó cómo el sistema manejaba consultas incorrectas o ambiguas, utilizando métricas específicas para evaluar la calidad de las respuestas generadas.

6.3.4 Modelos LLM evaluados

·**Llama38bToSql:** Basado en el modelo Llama3 8B con cuantización Q4_0.
·**Codeqwen7bToSql:** Basado en el modelo codeqwen 7B-chat-v1.5-q8_0, con la misma configuración que llama38bToSql.
GPT-3.5: Realizando las consultas online mediante el chat de ChatGPT con el modelo ChatGPT 3.5.
·**Mistral:** Basado en el modelo Codestral.
·**Phi3mini:** Basado en el modelo phi-3 3B (Mini) con cuantización Q4_K_M. Phi3mini fue el modelo más lento ejecutado en local, por lo que solo se testeó su generación de tokens (tiempo de ejecución) y no se realizaron todos los tests para comparar su precisión.

6.3.5 Ajustes en los prompts

Además de evaluar los modelos de lenguaje, se realizaron ajustes significativos en los prompts utilizados para interactuar con los asistentes de voz. Estos ajustes fueron cruciales para mejorar la precisión y la relevancia de las respuestas generadas por los modelos. El refinamiento de los prompts permitió optimizar la capacidad de los modelos para interpretar adecuadamente las intenciones del usuario, mejorando así la experiencia general del usuario con el asistente de voz.

6.4 Análisis y resultados de los tests

Los resultados de los tests realizados han demostrado ser positivos, validando la funcionalidad y rendimiento de los modelos de lenguaje natural utilizados en el proyecto del asistente de voz. Durante las pruebas, se evaluaron principalmente dos modelos locales destacados: llm_3b_q_4.0 y codeqwen:7b-chat-v1.5-q8_0.
El modelo phi-3 mini no fue considerado apto para el proyecto debido a sus tiempos de respuesta prolongados.

6.4.1 Tiempos de respuesta en local

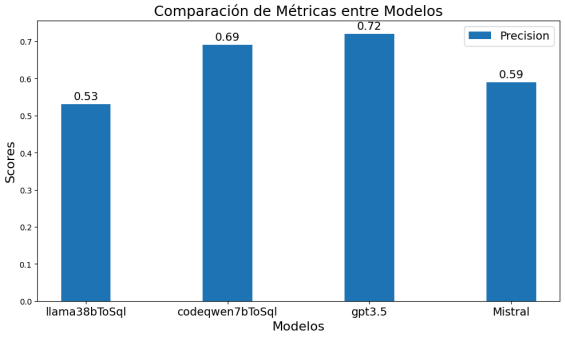
Se evaluaron los tiempos de generación de consultas SQL para tres modelos en local con diferentes tamaños. Los resultados fueron los siguientes:

Modelo	**Tamaño**	**Tiempo de respuesta**
phi3 mini	2.4GB Q_4_K_M	62.71 ms/palabra
llama3 8b	4.7GB Q4_0	31.81 ms/palabra
codeqwen chat 1.5	7.7 GB Q8_0	17.85 ms/palabra

Todos los modelos LLM cabían en la memoria RAM del servidor local. Cuanto mayor era el tamaño del modelo, más rápidos eran los tiempos de generación, posiblemente debido a la mejor optimización de memoria y procesamiento..

6.4.2 Modelos evaluados en precisión

Como se puede observar en la [Fig.7], el modelo con mejores resultados fue ChatGPT 3.5 muy seguido de cerca por codeqwen 7b. Esto sugiere la posibilidad de utilizar un modelo local con resultados similares a algunas implementaciones online.



[Fig.7] Métrica de precisión entre modelos clasificados

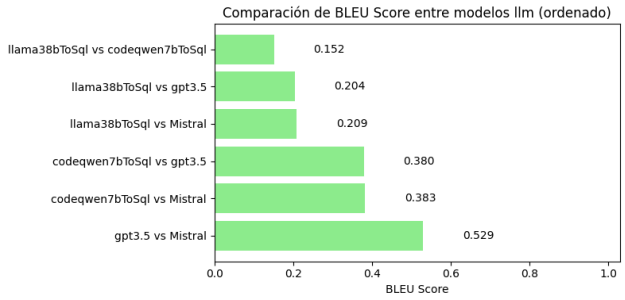
A pesar de la similitud de resultados entre codeqwen y GPT-3.5, aproximadamente un 30% de las consultas fueron respondidas incorrectamente.
Esto se debió principalmente a casos en los que la consulta SQL no se generaba correctamente, imposibilitando su ejecución. La mayoría de estos fallos se debieron a una mala gestión de espacios al crear la consulta SQL, creando palabras muy largas que, si se separaran, podrían funcionar.
Otro fallo común fue el uso de lenguaje ambiguo, como al pedir proyectos con ubicación en Europa sin considerar ubicaciones específicas como España. Estos casos podrían solventarse aumentando los detalles en la redacción de la consulta en lenguaje natural.

6.4.3 Métricas Beu, Meteor, Nist y Rouge

También se calcularon métricas BLEU, METEOR, NIST y ROUGE para comparar las consultas SQL generadas[17], permitiendo evaluar la similitud entre las salidas de los diferentes modelos. Estas métricas suelen estar muy relacionadas con la evaluación de traducción automática, pero aquí se adaptaron para evaluar las consultas SQL generadas. Para el uso adaptado al proyecto se usaron comparando la similitud de las consultas sql generadas por los modelos, dando una mayor puntuación cuanto más parecidas fuesen las respuestas.

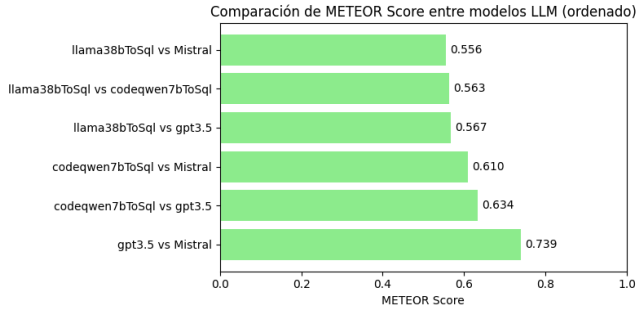
Como ya teníamos una clasificación de precisión para evaluar el funcionamiento de las consultas y por no tener un “ground truth” para las consultas sql, se decidió usar estas métricas para comparar y encontrar similitudes entre distintos modelos.

BLEU: Evalúa la similitud entre dos secuencias de texto generando n-gramas (fragmentos de palabras) y penaliza traducciones demasiado cortas [18]. Los resultados se muestran en la [Fig.8].



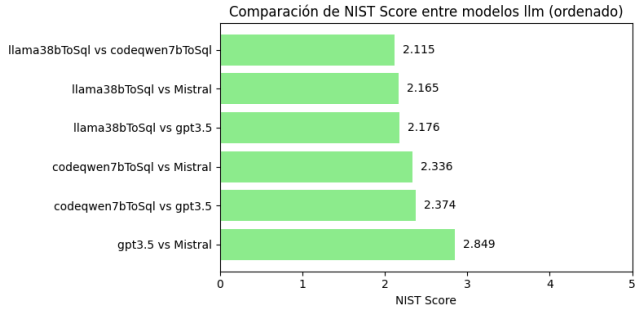
[Fig.8] Comparativa de modelos LLM según la métrica BLUE

METEOR: Evalúa la calidad de las traducciones considerando sinónimos, variaciones morfológicas y orden de palabras, combinando precisión y recall en una sola puntuación.. Los resultados se muestran en la [Fig.9].



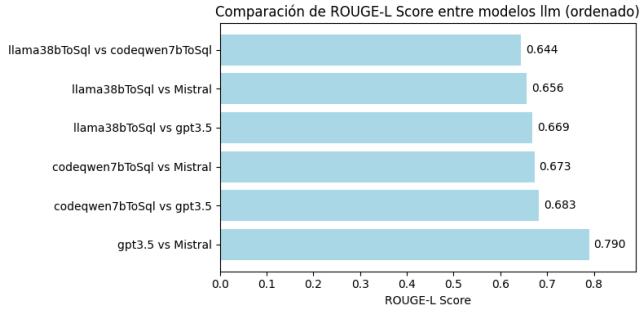
[Fig.9] Comparativa de modelos LLM según la métrica METEOR

NIST: Similar a BLEU, pero pondera más los n-gramas menos frecuentes y considera la importancia de diferentes n-gramas. Los resultados se muestran en la [Fig.10].



[Fig.10] Comparativa de modelos LLM según la métrica NIST

ROUGE: Tiene diversas variaciones según el cálculo de n-gramas que se elija. En este proyecto se usaron ROUGE-1, ROUGE-2 y ROUGE-L (longest common subsequence). Los resultados se muestran en la [Fig.11].



[Fig.11] Métrica Rouge-L. Los resultados para rouge-1 y 2 se muestran en el anexo por su similitud con Rogue-L

Con los resultados obtenidos en las comparaciones de las consultas SQL entre modelos, se puede observar que los modelos online GPT-3.5 y Codestral son los que mayor similitud tienen entre todos. Esto puede deberse a la gran cantidad de parámetros (175B y 22B respectivamente) en comparación con los otros modelos de 7B y 8B parámetros. De los modelos en local, gracias a BLEU, se puede observar que codeqwen es el modelo en local que genera consultas más similares a los modelos online.

6.5 Conclusiones

Durante el análisis comparativo de los resultados, se observó una notable correlación entre los modelos ChatGPT 3.5 y Codeqwen [19], que fueron consistentemente identificados como los más precisos en la generación de consultas SQL a partir de lenguaje natural. Esta correlación sugiere una capacidad similar entre ambos modelos para interpretar y responder efectivamente a solicitudes complejas.
Se observó que la cuantización del modelo Llama3 (Q4 en comparación con Q8 de Codeqwen) podría influir en la precisión de las consultas generadas. Esta diferencia cuantitativa posiblemente contribuyó a las variaciones en los resultados, con Codeqwen mostrando una mayor precisión y eficiencia general en la generación de consultas SQL comparado con Llama3.

Además, se destacó la notable rapidez de respuesta de Codeqwen durante las pruebas. Esta característica es crucial en aplicaciones donde la eficiencia y el tiempo de respuesta son críticos, proporcionando una ventaja competitiva sobre modelos con tiempos de respuesta más lentos.

Finalmente, mencionar que Codeqwen es un modelo especialmente diseñado para redactar código y superó las capacidades de Llama3 8b y, a pesar de que Codestral de Mistral es un modelo con 22B parámetros, obtuvo peores resultados. Esto se debió principalmente a que Codestral generó muchas consultas donde el nombre de las tablas o datos a buscar se mostraban en inglés, pese a que en ningún momento se le mencionó el nombre de las tablas en inglés.

6.6 Dificultades encontradas

Durante el desarrollo del proyecto del asistente de voz local, se encontraron varias dificultades significativas que requirieron atención especial y soluciones creativas
·Funcionamiento Subóptimo de los Modelos TTS para Español: A pesar de la existencia de varios modelos de TTS, muchos no lograron producir una voz sintetizada de alta calidad en español, resultando en una experiencia de usuario menos natural y fluida.

·Compatibilidad de Modelos en Hardware sin GPU: Muchos de los modelos de TTS, STT y LLM están diseñados para beneficiarse de las herramientas como

CUDA Cores de las GPU de Nvidia, lo que ralentiza considerablemente las ejecuciones en hardware sin GPU.

·Evaluación Manual de Consultas SQL: La premisa de generar consultas SQL para cualquier base de datos presentó una gran dificultad para evaluar los modelos, ya que no había una forma fácil de evaluar las consultas, lo que se tuvo que hacer manualmente. Además, la generación de datos para la base de datos de prueba y las consultas de test fue un trabajo manual considerable. Esto se podría solventar con consultas pre-generadas con anterioridad a la implementación de los modelos y del testeo.

7 Recomendaciones y mejoras futuras

1. **Uso de GPUs:** Implementar GPUs Nvidia con CUDA Cores para mejorar significativamente los tiempos de procesamiento y la eficiencia general del sistema.
2. **Fine-tuning de Modelos LLM[20]:** Realizar ajustes en los modelos con datos específicos del dominio para mejorar la precisión de las consultas generadas.
3. **Creación de un Frontend para un usuario inexperto:** Crear una interfaz de usuario más amigable y funcional para personas que desconozcan el funcionamiento del proyecto; Facilitando así la interacción del usuario con el asistente.
4. **Ampliación de Pruebas:** Incluir más categorías de consultas y escenarios de uso para mejorar la robustez del sistema y asegurar que el asistente pueda manejar una amplia variedad de solicitudes.
5. **Mejora de la Base de Datos:** Usar datos más complejos y variados para pruebas más complejas, asegurando que el sistema pueda manejar situaciones del mundo real con precisión.
6. **Mecanismos de Autoaprendizaje:** Desarrollar mecanismos de autoaprendizaje para que el asistente pueda mejorar continuamente su precisión y adaptarse a nuevas consultas y contextos.
7. **Fine-tuning de Whisper STT y Coqui TTS en Español:** Ajustar finamente los modelos de reconocimiento de voz y síntesis de voz para mejorar su desempeño específico en español.

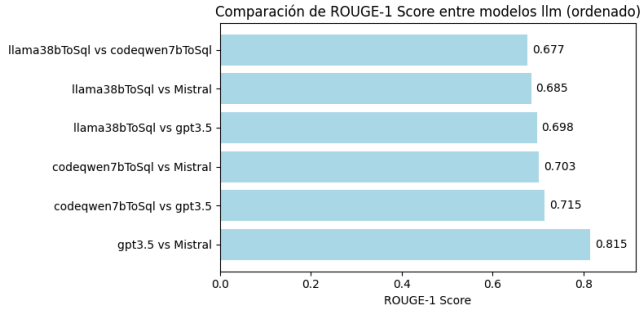
En resumen, este proyecto ha logrado crear un asistente virtual que permite la interacción mediante voz o texto para realizar consultas SQL en lenguaje natural, utilizando modelos LLM en un entorno local. Codeqwen7b, implementado con Ollama, resultó ser el modelo más efectivo para las necesidades del proyecto, combinando precisión y eficiencia. Las recomendaciones y mejoras futuras proporcionan una hoja de ruta clara para seguir mejorando y ampliando las capacidades del asistente virtual.

Bibliografia

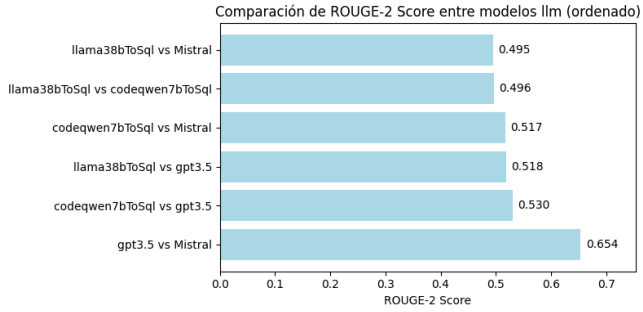
Enlaces revisados a fecha de 02/Julio/2024.

[1] Paper Weaver, R. L. (2017). Privacy in the Digital Age. Cambridge University Press.
[2] Explicación Asistente virtual
[3] Explicación Lenguaje natural
[4] Explicación Speech to text
[5] Explicación Text to speech
[6] Proyecto Whisper, Automatic speech recognition
[7] Libro Stellman, A., & Greene, J. (2014). Learning Agile: Understanding Scrum, XP, Lean, and Kanban. O'Reilly Media.
[8] Paper Vaswani, A., et al. (2017). Attention Is All You Need.
[9] Web Proxmox Virtual Environment
[10] Explicación CUDA
[11] Web Laravel framework
[12] Paper Radford, A., et al. (2022). Robust Speech Recognition via Large-Scale Weak Supervision.
[13] Libro Taylor, P. (2009). Text-to-Speech Synthesis. Cambridge University Press
[14] Proyecto CoquiTTS
[15] Proyecto XTTS-v2
[16] Web Ollama
[17] Libro Koehn, P. (2010). Statistical Machine Translation. Cambridge University Press.
[18] Paper Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002). BLEU: a method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting of the Association for Computational Linguistics
[19] Paper Brown, T. B., et al. (2020). Language Models are Few-Shot Learners.
[20] Explicación Fine-tuning llms

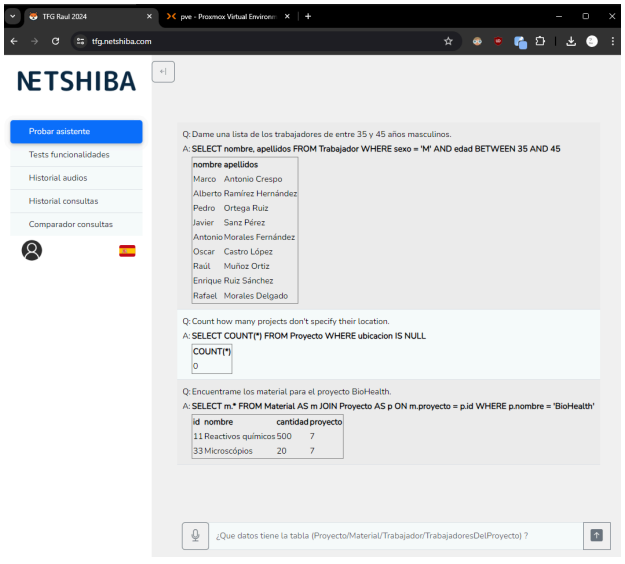
Anexo



[Fig.12] Comparativa de modelos LLM según la métrica Rouge-1

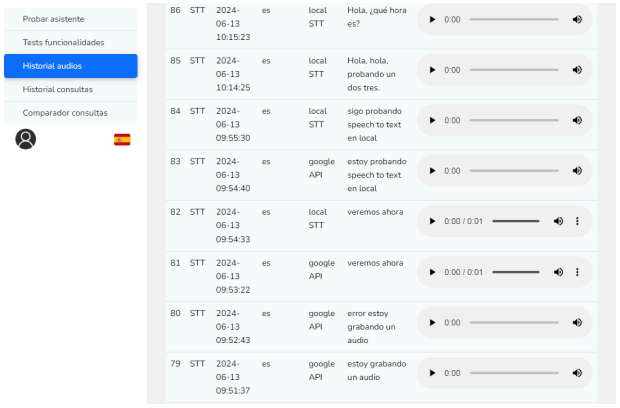


[Fig.13] Comparativa de modelos LLM según la métrica Rouge-2

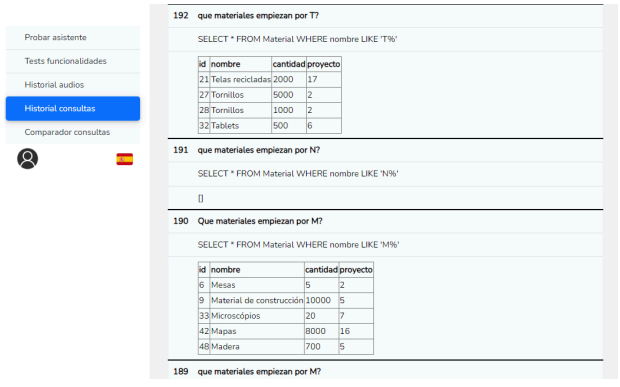


[Fig.14] Vista principal de la web. Interacción con el asistente.

[Fig.15] Vista del apartado tests de la web.

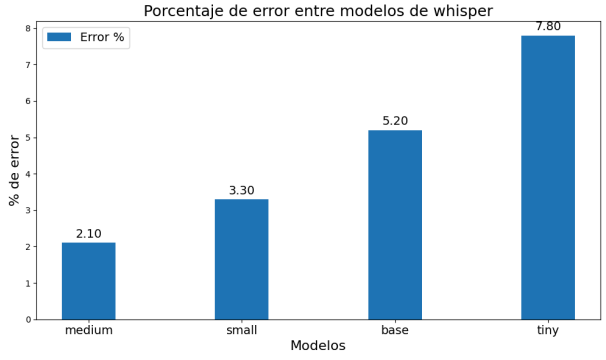


[Fig.16] Vista del historial de transcripción de audio.

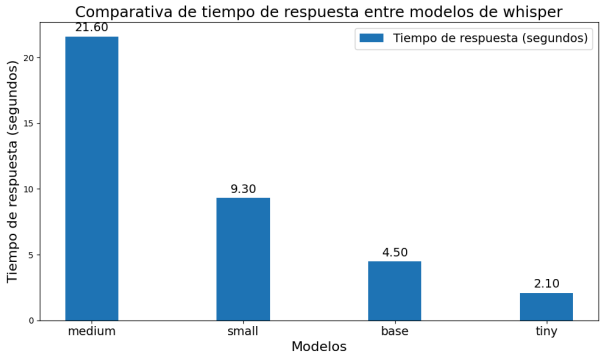


[Fig.17] Vista del historial de consultas sql generadas por el asistente.

[Fig.18] Vista del clasificador de los prompts del asistente.



[Fig.19] Comparativa del porcentaje de palabras erróneas al procesar las consultas naturales de test a sql, entre los modelos de diferente tamaño de whisper



[Fig.20] Comparativa de tiempo de respuesta entre los modelos de diferente tamaño de whisper