Your allies for change

Investigación en Multimodalidad con Gemini

Enfoque práctico del modelo “Multimodal Gemini 2 Flash Exp”

# Análisis de Gemini (Google)

## Introducción a Gemini

**Gemini** es un modelo multimodal que destaca por su capacidad de procesar y generar contenidos en diferentes formatos (texto, audio y video) de manera simultánea y en tiempo real. Para habilitar estas interacciones, Google ha desarrollado la **API de Multimodal Live**, la cual permite:

* **Interacciones de voz y video bidireccionales de baja latencia**: Los usuarios pueden mantener conversaciones de voz naturales y fluidas con el modelo, interrumpirlo en cualquier momento e, incluso, combinar la comunicación de texto y video según la necesidad.
* **Conversaciones “human-like”**: Su arquitectura está diseñada para ofrecer respuestas que imitan la dinámica de las interacciones humanas, dando pie a experiencias más inmersivas.

Gracias a estas funcionalidades, Gemini abre la puerta a una variedad de usos prácticos que van desde asistentes virtuales conversacionales hasta plataformas de educación y entretenimiento en tiempo real.

## Características Multimodales de Gemini

La **API de Multimodal Live** incorpora una serie de funciones clave que convierten a Gemini en un modelo sumamente versátil:

* **Multimodalidad**

El modelo puede “ver, escuchar y hablar”, lo que significa que integra de forma nativa texto, audio y video en sus procesos de entrada y salida.

* **Interacción en tiempo real de baja latencia**

Permite entregar respuestas rápidas, factor esencial en aplicaciones donde la inmediatez es prioritaria, como la atención al cliente o la formación en entornos virtuales.

* **Memoria de sesión**

Durante una sola sesión, Gemini retiene la información de interacciones previas, facilitando el seguimiento del contexto y evitando la repetición innecesaria de datos.

* **Compatibilidad con llamadas a funciones, ejecución de código y búsqueda externa**

Ofrece la capacidad de conectarse con APIs de terceros, ejecutar rutinas específicas o consultar fuentes externas, ampliando de forma significativa el espectro de posibilidades para el desarrollo de aplicaciones integradas.

* **Detección automática de actividad de voz (VAD)**

Reconoce de manera precisa el inicio y el final de la voz del usuario, fomentando conversaciones fluidas y permitiendo la interrupción del modelo cuando sea necesario.

# Prueba de Gemini 2.0 flash en aistudio.google.com

Actualmente, Google ofrece la posibilidad de explorar **Gemini** y sus capacidades multimodales a través de plataformas como [**aistudio.google.com**](https://aistudio.google.com/).

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Sin embargo, el enfoque de esta investigación no se limita a la prueba directa en la plataforma oficial, sino que se orienta a:

* **Integrar Gemini en aplicaciones existentes**, como **GPTup o AIsQ**, para potenciar la experiencia conversacional. Esta integración permitiría a GPTup aprovechar la multimodalidad de Gemini y ofrecer a sus usuarios una interacción más rica (voz, video y texto), junto con mejores capacidades de contextualización.
* **Implementar y probar el modelo usando código Python** de forma local o en entornos de desarrollo personalizados. Aunque aquí no profundizaremos en la codificación en sí, el objetivo es mostrar cómo Gemini puede combinarse con aplicaciones ya creadas para proporcionar respuestas rápidas y fluidas, ejecutar tareas específicas, consultar fuentes externas y adaptarse a distintas necesidades de negocio o investigación.

# Uso básico del modelo Gemini

En esta sección se aborda tanto la base teórica como la aplicación práctica de las tecnologías clave que permiten la integración de Gemini en entornos de desarrollo. Concretamente, se describe el uso de la **API de Multimodal Live**, la cual funciona sobre un modelo de comunicación con estado que se apoya en **WebSockets** para habilitar interacciones multimodales en tiempo real.

## Fundamentos de la API de Multimodal Live y WebSockets

La **API de Multimodal Live** proporciona un canal de comunicación bidireccional y en tiempo real con Gemini, permitiendo a la aplicación cliente enviar y recibir datos de texto, audio o video sin bloqueos ni demoras innecesarias.

* **API con estado**: La sesión mantiene un “contexto vivo” que registra las interacciones previas, de forma que el modelo puede recordar información escuchada o vista anteriormente.
* **Uso de WebSockets**:
  + **Comunicación bidireccional y full-duplex**: WebSocket ofrece un canal continuo entre el cliente y el servidor que permite el envío y recepción de datos de forma simultánea, esencial para conversaciones fluidas en aplicaciones que involucran voz y video.

Diagrama

El contenido generado por IA puede ser incorrecto.

* + **Protocolo estandarizado**: La API WebSocket está normalizada por el W3C, mientras que el protocolo fue definido por la IETF como RFC 6455. Esto brinda compatibilidad y estabilidad a largo plazo para proyectos que quieran integrar la API de Multimodal Live.
  + **Ventajas sobre conexiones HTTP tradicionales**: Dado que muchas redes bloquean conexiones TCP en puertos distintos al 80 o 443, la capacidad de WebSockets de operar sobre estos puertos y multiplexar múltiples servicios en un mismo canal TCP evita la necesidad de abrir puertos adicionales y simplifica la configuración de la infraestructura.

## Implementación en Python

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.Para la implementación en Python, es recomendable seguir las indicaciones proporcionadas en la documentación oficial de Google sobre la **API Multimodal Live**. A continuación, se muestra un ejemplo sencillo de cómo utilizar esta API para la generación de texto a texto en **Python 3.9** o versiones posteriores. [API Multimodal Live](https://ai.google.dev/gemini-api/docs/multimodal-live?hl=es-419)

Vamos a aplicar un ejemplo de cómo usar la API de Multimodal Live para la generación de texto a texto con Python 3.9 o versiones posteriores.

### Instala la biblioteca de la API de Gemini

Para instalar el paquete google para usar sus modelos.

pip install google-genai

Una vez instalado importamos las librerías utilizadas.

Texto

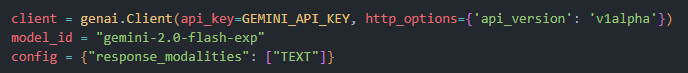
El contenido generado por IA puede ser incorrecto.

**Carga de Variables de Entorno**: Se extrae la clave de API (o cualquier otra configuración sensible) desde un archivo .env, evitando exponer datos críticos directamente en el código.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

**Creación del Cliente**: Se inicializa la clase genai.Client con la clave de API para autenticar las solicitudes al servicio de Gemini y se configura el modelo que se usará, en este caso "gemini-2.0-flash-exp".



**Sesión Asíncrona con WebSockets**:

Se utiliza async with para establecer y mantener la conexión en tiempo real con la API a través de WebSockets.



Durante la sesión, el usuario ingresa texto a través de la consola. Si escribe "exit", se cierra la sesión y el programa termina.

Texto

El contenido generado por IA puede ser incorrecto.

**Interacción con el Modelo**:

**Envío de Mensajes**: El texto introducido por el usuario se envía al modelo con session.send(), indicando con end\_of\_turn=True que se ha finalizado la interacción por ese turno y se espera la respuesta.



**Recepción de Respuestas**: Con session.receive(), se obtiene un flujo potencialmente continuo de datos (ideal si se requiere la respuesta en modo streaming). Si el modelo envía varios fragmentos, se irán imprimiendo en pantalla de forma progresiva.

Texto

El contenido generado por IA puede ser incorrecto.

Todo el código de ejemplo básico.



Este ejemplo, aun siendo básico, demuestra la sencillez y fluidez con las que se puede interactuar con Gemini gracias a las conexiones WebSocket. La baja latencia y la comunicación bidireccional en tiempo real brindan una experiencia óptima de conversación, lo que hace que la integración del modelo en diversas plataformas (web, móviles o incluso gafas inteligentes como las Meta Ray-Ban) sea altamente factible y eficiente.

Texto

El contenido generado por IA puede ser incorrecto.

## Tiene memoria

Un aspecto fundamental de los modelos de lenguaje de última generación es su capacidad para conservar y utilizar el contexto de la conversación a lo largo de múltiples intercambios. A partir del ejemplo mencionado, se observa que el modelo puede recordar y hacer referencia a la información proporcionada en preguntas y respuestas anteriores, lo que se traduce en una interacción más coherente y fluida, y mejora exponencialmente la relación con el usuario.



Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

# Implementación de Text to Voice

En esta sección damos un paso más allá al explorar el enfoque para habilitar interacciones en las que el usuario introduce texto y recibe una respuesta hablada generada por el modelo Gemini. Esta arquitectura, conocida como Text to Speech (TTS), permite una comunicación más natural y accesible entre humanos y sistemas de inteligencia artificial. El objetivo de esta implementación es demostrar cómo se puede transformar la entrada textual en audio en tiempo real, utilizando la API experimental de Gemini Multimodal Live, lo cual abre la puerta a aplicaciones innovadoras en asistencia virtual, accesibilidad y interfaces conversacionales.

## Arquitectura del sistema

La solución se basa en el uso del modelo **gemini-2.0-flash-exp** de la API Gemini, que permite la generación de respuestas en formato de audio. La arquitectura del sistema se compone de los siguientes elementos clave:

* **Cliente de la API:** Configurado mediante una clave de API y parámetros específicos (como la versión y el modelo), se establece la conexión asíncrona con la API de Gemini.
* **Configuración del Prompt y Parámetros de Voz:** Se define un mensaje de instrucciones del sistema para establecer el rol del modelo (por ejemplo, "Eres un asistente experto en tecnología que responde en Español") y se especifica la voz deseada (ej. "Charon") a través del parámetro generation\_config.
* **Gestión de Archivos de Audio:** Se utiliza un *context manager* para crear y configurar archivos WAV en los que se almacenan los datos de audio recibidos.
* **Interacción Asíncrona:** Se implementa un bucle asíncrono que envía la entrada del usuario y recibe en fragmentos la respuesta en audio, permitiendo su reproducción al finalizar cada interacción.

## Configuración y Conexión a la API

Se define un objeto de configuración que incluye:

* **Instrucción del Sistema:** Establece el rol del modelo, orientándolo para responder como un asistente experto en tecnología en Español.
* **Parámetros de Generación:** Especifica que la respuesta debe incluir salida en audio y se selecciona la voz (por ejemplo, "Charon").

La configuración se estructura de la siguiente manera

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

## Gestión y Almacenamiento del Audio

Aunque el objetivo de la investigación es que el modelo hable directamente en esta fase de las pruebas va a ser guardado en disco las respuestas del modelo, posteriormente cambaremos este funcionamiento.

Texto

El contenido generado por IA puede ser incorrecto.

## Manejo de sesión y comunicación asíncrona

**Envío de la entrada:** Se solicita al usuario que ingrese una pregunta o comando, que se envía a la API utilizando el método session.send con parámetros por palabra clave.



**Recepción de la respuesta:** La respuesta en audio se recibe en fragmentos a través de un iterable asíncrono. La función async\_enumerate facilita la iteración y se escriben los datos en el archivo WAV configurado.

Texto

El contenido generado por IA puede ser incorrecto.

## Conclusión de Text to Voice

La implementación de Text to Voice mediante la API de Gemini demuestra cómo es posible integrar respuestas en audio en aplicaciones interactivas. Los puntos destacados son:

* **Interacción en tiempo Real:** La arquitectura asíncrona permite enviar y recibir datos de forma fluida, logrando una comunicación inmediata.
* **Flexibilidad en la configuración:** La posibilidad de definir el rol del modelo y seleccionar diferentes voces (mediante speech\_config) ofrece una personalización que puede adaptarse a diversos contextos y aplicaciones.
* **Aplicaciones prácticas:** Este enfoque puede extenderse a asistentes virtuales, sistemas de accesibilidad y otras soluciones donde la interacción hablada mejora la experiencia del usuario.
* **Diferentes Voces:** El uso de distintas voces, aunque todas con características similares, añade un plus a la interactividad, permitiendo ajustar la experiencia de usuario según preferencias y necesidades específicas.



# Implementación de voice to voice

Después de comprobar el funcionamiento del sistema Text to Voice y confirmar que la conversión de texto en audio (TTS) opera correctamente, se ha decidido profundizar en la siguiente etapa: la implementación de un sistema Voice to Voice. Esta solución integra dos tecnologías esenciales: la conversión de voz a texto (STT) para capturar la entrada del usuario y la conversión de texto a voz (TTS) para generar la respuesta hablada. El objetivo es lograr una comunicación completamente natural, donde tanto la entrada como la salida se realicen mediante voz, eliminando la necesidad de interacción escrita y operando en tiempo real. Además, se ha incluido como requisito fundamental que el sistema permita interrumpir al modelo mientras éste está emitiendo la respuesta, facilitando una interacción más dinámica y adaptativa.

## Arquitectura del sistema

El código se estructura en torno a la clase AudioLoop, la cual centraliza el manejo de las siguientes funcionalidades:

* **Captura de audio:** Se usa PyAudio para abrir una secuencia de entrada y leer bloques de audio (chunks) de forma asíncrona.
* **Envío en tiempo real:** Los datos de audio se colocan en una cola (out\_queue) y se envían a la sesión establecida con la API Gemini.
* **Recepción de respuesta:** Se reciben mensajes de la API y se distinguen dos tipos de respuesta:
  + Audio: Se coloca en una cola (audio\_in\_queue) para su posterior reproducción.
  + Texto: Se imprime en la salida estándar.
* **Reproducción de audio:** Se abre una salida de audio mediante PyAudio y se reproducen los datos en cola de manera asíncrona.
* **Coordinación de tareas:** Se utiliza un asyncio.TaskGroup para correr las tareas de envío, captura, recepción y reproducción de manera simultánea.

## Estructura del Código y Funcionalidades

* asyncio: Para gestionar la concurrencia asíncrona.
* PyAudio: Para el manejo de audio (captura y reproducción).
* dotenv: Para la carga de variables de entorno, en este caso la API key.
* google.genai: Cliente para conectarse a la API Gemini.

## Clase AudioLoop

La clase AudioLoop encapsula la lógica del bucle de procesamiento de audio y su comunicación con la API **Gemini**. Gestiona la captura, almacenamiento y transmisión de datos de audio de manera asíncrona, permitiendo una integración eficiente con la API.

**Atributos principales**

**audio\_in\_queue**: Cola asíncrona que almacena los datos de audio recibidos desde la API **Gemini** para su posterior procesamiento.

**out\_queue**: Cola asíncrona que almacena los datos de audio capturados y listos para ser enviados a la API.

**session**: Objeto de sesión que gestiona la conexión activa con la API **Gemini**, permitiendo una comunicación estable y eficiente.

Texto

El contenido generado por IA puede ser incorrecto.

## Método listen\_audio

Capturar audio en tiempo real desde el micrófono y encolar cada bloque (chunk) en out\_queue para su posterior envío.

**Implementación**

1. Se obtiene el dispositivo de entrada predeterminado utilizando pya.get\_default\_input\_device\_info().
2. Se abre un flujo de audio en modo lectura con los parámetros especificados (formato, número de canales, tasa de muestreo, etc.).
3. Se utiliza asyncio.to\_thread para ejecutar la operación de lectura bloqueante sin interrumpir el bucle de eventos asíncrono.
4. Cada bloque de audio capturado se encapsula en un diccionario con la clave "data" y el **MIME type** "audio/pcm", y luego se encola en out\_queue.

**Observaciones**

* Se emplea la opción exception\_on\_overflow para evitar excepciones en modo **debug**, asegurando una captura de audio estable incluso en condiciones de carga alta.

Texto

El contenido generado por IA puede ser incorrecto.

## Método send\_realtime

Transmitir en tiempo real los datos de audio capturados (almacenados en out\_queue) a la API Gemini para su procesamiento.

Implementación

1. Se ejecuta en un bucle infinito que extrae cada mensaje de la cola out\_queue.
2. Cada mensaje se envía a la API utilizando el método send de la sesión activa.

Observaciones

* Este método funciona como un puente entre la captura local de audio y el procesamiento remoto en la API Gemini, garantizando una transmisión fluida y en tiempo real.

Texto

El contenido generado por IA puede ser incorrecto.

## Método receive\_audio

Recibir y procesar las respuestas de la API Gemini, diferenciando entre audio y texto para su correcta gestión.

Implementación

1. Se obtiene un "turn" de respuesta mediante session.receive().
2. Para cada respuesta recibida, se determina su tipo:
   * Si contiene datos de audio, se encola en audio\_in\_queue.
   * Si contiene texto, se imprime en la salida estándar sin salto de línea, permitiendo la continuidad de la conversación.
3. Al finalizar el turno, se limpia la cola de audio para eliminar posibles datos residuales en caso de interrupción.

Observaciones

* Este método permite manejar respuestas multimodales, adaptándose dinámicamente a los diferentes formatos de salida proporcionados por la API Gemini.

Texto

El contenido generado por IA puede ser incorrecto.

## Método play\_audio

Reproducir en tiempo real el audio recibido desde la API Gemini, garantizando una experiencia fluida y sin interrupciones.

**Implementación**

1. Se abre un flujo de salida de audio con los parámetros adecuados (formato, número de canales, tasa de muestreo, etc.).
2. Se ejecuta un bucle infinito que extrae los datos de audio\_in\_queue y los escribe en el stream de salida.
3. Se utiliza asyncio.to\_thread para ejecutar la reproducción en un hilo secundario, evitando el bloqueo del bucle de eventos principal.

**Observaciones**

* Asegura una reproducción fluida y sin cortes, optimizando el manejo asíncrono del audio.

Texto

El contenido generado por IA puede ser incorrecto.

## Método run

Coordinar y ejecutar todas las tareas asíncronas necesarias para el funcionamiento del sistema de procesamiento y transmisión de audio en tiempo real.

**Implementación**

1. Se establece una conexión asíncrona con la API Gemini utilizando un contexto async with.
2. Se crea un asyncio.TaskGroup para ejecutar de forma concurrente las siguientes tareas:
   * send\_realtime: Envío de audio capturado a la API.
   * listen\_audio: Captura de audio desde el micrófono.
   * receive\_audio: Recepción y procesamiento de respuestas de la API.
   * play\_audio: Reproducción del audio recibido.
3. Se inicializan las colas audio\_in\_queue y out\_queue para la gestión de los datos de audio.
4. Se mantiene la ejecución indefinida mediante await asyncio.Event().wait(), asegurando que el sistema permanezca activo hasta una cancelación externa.
5. Se captura y gestiona cualquier excepción, garantizando el cierre adecuado del flujo de audio y mostrando el traceback para depuración.

**Observaciones**

* Permite la ejecución simultánea y sincronizada de todas las funciones esenciales del sistema.
* Implementa un mecanismo de manejo de errores para asegurar una finalización segura en caso de fallo.

Texto

El contenido generado por IA puede ser incorrecto.

# Conclusiones finales

## Conclusiones

Tras la implementación del modelo **voice-to-voice**, se han identificado diversos aspectos a mejorar. Actualmente, el modelo sigue en **fase experimental**, por lo que aún no es capaz de generar texto y voz de manera simultánea. Sin embargo, según la documentación de **Google**, esta funcionalidad estará disponible próximamente.

## Capacidad de comprensión

El modelo demuestra una **precisión excepcional** en la comprensión del habla. En todas las pruebas realizadas, ha entendido a la perfección las instrucciones proporcionadas, no he entrado a valorar la calidad de la respuestas, me he centrado en el entendimiento de mi voz y de la calidad de la voz generada.

## Inferencia y respuesta

* La **velocidad de inferencia** es adecuada, con respuestas prácticamente instantáneas en múltiples idiomas.
* Se ha detectado cierta **pérdida de datos**, sobre todo cuando la calidad de internet es regular

## Funcionamiento multimodal

El sistema **multimodal** funciona de manera correcta, ofreciendo una gran oportunidad para su integración en diversos dispositivos, como **móviles, gafas inteligentes y otros sistemas embebidos**.

## Tiempos de inferencia y latencia

Los **tiempos de inferencia** son muy buenos, proporcionando una experiencia conversacional que se siente **natural y fluida**. La **baja latencia** contribuye significativamente a una interacción en tiempo real, mejorando la **satisfacción del usuario**. Esta eficiencia es comparable a soluciones avanzadas en el mercado, como la API en tiempo real de OpenAI, que permite respuestas instantáneas gracias a la comunicación directa de audio a audio .

## Inconvenientes

## Destacar las limitaciones actuales del modelo Gemini en su fase experimental. Aunque Gemini está diseñado para procesar y generar múltiples tipos de datos, como texto, imágenes y audio, de manera simultánea , se han observado restricciones en su funcionalidad multimodal. Específicamente, cuando el modelo genera una respuesta en formato de audio, no es capaz de proporcionar simultáneamente una respuesta en texto, y viceversa. Esta limitación se atribuye a su estado experimental y a la complejidad inherente de integrar múltiples modalidades de salida de manera coherente.

## Próximos pasos

1. Implementar un sistema RAG (Retrieval-Augmented Generation) para limitar las respuestas del modelo a un dominio específico.
2. Explorar la visión artificial, probando su capacidad para entender imágenes y combinarlas con la interacción por voz.
3. Integración de todos los modos posibles, texto, voz e imagen