

PROYECTO FIN DE GRADO

TÍTULO: Mejora de flujos de trabajo en producción musical con Inteligencia Artificial

AUTOR/A: Luis Barroso Díaz

TITULACIÓN: Ingeniería Telemática

TUTOR/A: José Fernán Martínez Ortega

DEPARTAMENTO: Departamento de Ingeniería Telemática y Electrónica

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE/A: José Manuel Pardo Martín

TUTOR/A: José Fernán Martínez Ortega

SECRETARIO/A: Néstor Lucas Martínez

Fecha de lectura: 30 de septiembre de 2025

Calificación:

El Secretario/La Secretaria,

Resumen

La producción musical digital ha experimentado un crecimiento notable en los últimos años, impulsada por la accesibilidad de herramientas informáticas y la proliferación de recursos en línea. Cada vez son más los usuarios que, sin formación musical avanzada ni experiencia técnica previa, se inician en la creación musical desde sus hogares. Esta tendencia, que ha democratizado el acceso a la composición y edición musical, también ha puesto de manifiesto nuevas barreras: la complejidad de las estaciones de trabajo de audio digital, la terminología especializada y la dificultad para plasmar ideas musicales de forma rápida y sencilla pueden suponer un gran obstáculo para los principiantes.

En este contexto, surge el proyecto del desarrollo de un asistente musical inteligente, concebido para facilitar la creación y edición de música digital mediante comandos de voz y tarareos. El objetivo es proporcionar una herramienta accesible y adaptada a usuarios sin conocimientos técnicos profundos, permitiendo materializar ideas musicales de manera intuitiva y eficiente. La solución propuesta se integra en el entorno de producción musical Reaper, y se apoya en tecnologías de reconocimiento de voz, procesamiento de lenguaje natural, conversión de audio a MIDI y automatización de procesos, eliminando así la necesidad de dominar complejos flujos de trabajo o interfaces tradicionales. Además, el asistente utiliza un modelo de inteligencia artificial a través del servicio OpenRouter, capaz de ofrecer soluciones inteligentes y asistencia personalizada durante el proceso creativo, guiando al usuario y resolviendo dudas en tiempo real.

El proyecto se centra en la implementación de un sistema capaz de interpretar instrucciones orales y tarareos, transformándolos en material musical editable y automatizando tareas habituales dentro de la estación de trabajo de audio digital. Para ello, se han empleado lenguajes y herramientas como Python para desarrollar gran parte del proyecto, Lua para la automatización en Reaper y el protocolo Open Sound Control para la comunicación entre módulos, priorizando soluciones de código abierto.

El desarrollo se ha realizado siguiendo una metodología iterativa y modular, con validaciones progresivas de cada componente y pruebas de usuario orientadas a asegurar la usabilidad y la calidad del sistema. Los resultados obtenidos muestran que el asistente reduce de forma significativa la complejidad técnica del proceso creativo, permitiendo a los usuarios principiantes experimentar y producir música de manera más accesible y natural.

Este proyecto aporta una herramienta innovadora que amplía las posibilidades creativas de los usuarios y contribuye a la democratización de la producción musical digital. La arquitectura modular y la integración de inteligencia artificial sientan las bases para futuras mejoras, como la incorporación de nuevas funcionalidades, la adaptación a otros entornos de producción musical o la integración de algoritmos aún más avanzados. Así, la solución desarrollada no solo responde a una necesidad actual, sino que abre la puerta a nuevas formas de interacción y creación musical, consolidando el papel de la tecnología como facilitadora de la creatividad y la accesibilidad en la música digital.

Abstract

Digital music production has experienced remarkable growth in recent years, driven by the accessibility of software tools and the proliferation of online resources. Increasingly, users without advanced musical training or technical experience are beginning to create music from their homes. While this trend has democratized access to music composition and editing, it has also revealed new barriers: the complexity of digital audio workstations, specialized terminology, and the difficulty of quickly and easily capturing musical ideas can pose significant obstacles for beginners.

In this context, this project presents the development of an intelligent musical assistant, designed to facilitate the creation and editing of digital music through voice commands and humming. The aim is to provide an accessible tool tailored to users without deep technical knowledge, enabling them to realize their musical ideas in an intuitive and efficient way. The proposed solution integrates into the Reaper music production environment and leverages technologies such as voice recognition, natural language processing, audio-to-MIDI conversion, and process automation, thus removing the need to master complex workflows or traditional interfaces. Furthermore, the assistant incorporates an artificial intelligence model via the OpenRouter service, capable of offering intelligent solutions and personalized assistance during the creative process, guiding the user and resolving questions in real time.

The project focuses on implementing a system capable of interpreting spoken instructions and humming, transforming them into editable musical material and automating common tasks within the digital audio workstation. To achieve this, languages and tools such as Python have been used for most of the development, Lua for automation within Reaper, and the Open Sound Control protocol for communication between modules, with a focus on open-source solutions.

Development followed an iterative and modular methodology, with progressive validation of each component and user testing aimed at ensuring system usability and quality. The results show that the assistant significantly reduces the technical complexity of the creative process, enabling beginner users to experiment and produce music in a more accessible and natural way.

This project delivers an innovative tool that expands users' creative possibilities and contributes to the democratization of digital music production. The modular architecture and integration of artificial intelligence lay the groundwork for future improvements, such as the incorporation of new features, adaptation to other music production environments, or the integration of even more advanced algorithms. In this way, the developed solution not only addresses a current need but also opens the door to new forms of interaction and musical creation, consolidating the role of technology as a facilitator of creativity and accessibility in digital music.

Índice de figuras

Figura 1. Arquitectura de asistente genérico	6
Figura 2. Arquitectura sistema de reconocimiento de voz	8
Figura 3. Flujo de funcionamiento en la interpretación del lenguaje natural	9
Figura 4. Arquitectura MVC genérica [9]	10
Figura 5. Diagrama de ejemplo de arquitectura modular	12
Figura 6. Interfaz de filtrado de modelos en OpenRouter	21
Figura 7 Ejemplo de métricas proporcionadas por OpenRouter para un modelo específico .	21
Figura 8. Diagrama funcional del sistema	30
Figura 9. Arquitectura general del sistema.....	31
Figura 10. Prototipo GUI del sistema.....	33
Figura 11. Comunicación entre archivos del módulo GUI y otros módulos funcionales	34
Figura 12. Flujo de datos entre vosk y wit.ai	37
Figura 13. Ejemplo de entrenamiento del modelo de interpretación de comandos.....	38
Figura 14. Arquitectura módulo comandos	39
Figura 15. Flujo de datos en la creación del MIDI a partir de tarareos del usuario	41
Figura 16. Ejemplo de archivo midi_tempos.json	42
Figura 17. Integración de módulo tarareos con otros módulos	42
Figura 18. Arquitectura del módulo de comunicaciones	43
Figura 19. Integración protocolo OSC en arquitectura del módulo de comunicaciones.....	44
Figura 20. Ejemplo visual del archivo default.ReaperOSC	45
Figura 21. Funcionamiento del módulo de análisis	47
Figura 22. Flujo de funcionamiento de bass_midi.py.....	48
Figura 23. Interacción entre archivos del módulo de IA.....	49
Figura 24. Ejemplo visual del prompt utilizado en synthetizer.py	51
Figura 25. Flujo de proceso de mejoras de melodía mediante efectos o sintetizador	53
Figura 26. Flujo de aplicación de valores de parámetros de efectos en Reaper	54
Figura 27. Gráfica de resultados pruebas de reconocimiento de voz	59
Figura 28. Gráfica de resultados prueba de calidad musical.....	61
Figura 29. Gráfica de resultados prueba de comunicación e integración con Reaper	63
Figura 30. Configuración OSC en Reaper.....	82
Figura 31. Pantalla de inicio del asistente	83
Figura 32. Pantalla del asistente durante la grabación de un comando	84
Figura 33. Mensaje del asistente tras un comando no valido	85
Figura 34. Mensaje de petición para el nombre del MIDI	86
Figura 35. GUI tarareos.....	86
Figura 36. Pantalla de Reaper tras grabar tarareos y abrir con Reaper	87
Figura 37. Petición del asistente para la realización de una pregunta por parte del usuario .	88

Índice de tablas

Tabla 1. Métodos de análisis del archivo MIDI	48
Tabla 2. Funciones de conversión de valores de parámetros	52
Tabla 3. Prueba de caja negra del sistema	57
Tabla 4. Resultados pruebas de reconocimiento de voz	59
Tabla 5. Resultados prueba de calidad musical	60
Tabla 6. Resultados prueba de comunicación e integración con Reaper	62
Tabla 7. Costes materiales	65
Tabla 8. Costes de personal	65
Tabla 9. Costes totales	66
Tabla 10. Funcionalidades y ejemplos de comandos	83

Lista de acrónimos

API	Interfaz de programación de aplicaciones (Application programming interface)
BPM	Pulsos por minuto (Beats per minute)
DAW	Estación de trabajo de audio digital (Digital Audio Workstation)
GUI	Interfaz gráfica de usuario (Graphical user interface)
HTTP	Protocolo de transferencia de hipertexto (Hypertext Transfer Protocol)
IA	Inteligencia Artificial (Artificial Intelligence)
IP	Protocolo de internet (Internet protocol)
JSON	Notación de objeto de JavaScript (JavaScript Object Notation)
MIDI	Interfaz Digital Instrumentos Musicales (Musical Instrument Digital Interface)
ODS	Objetivos de Desarrollo Sostenible (Sustainable Development Goals)
OSC	Open Sound Control
SDK	Kit de desarrollo software (Software Development Kit)
TCP	Protocolo de control de transmisión (Transmission Control Protocol)
UDP	Protocolo de datagramas de usuario (User Datagram Protocol)
VST	Virtual Studio Technology

Índice de contenidos

Resumen	i
Abstract	ii
Índice de figuras	iii
Índice de tablas	iv
Lista de acrónimos.....	v
1. Introducción	1
1.1. Marco y motivación del proyecto.....	1
1.2. Objetivos técnicos y académicos	2
1.3. Estructura del resto de la memoria	3
2. Marco tecnológico.....	5
2.1. Asistente Inteligente	5
2.2. Reconocimiento de voz e interpretación de comandos	7
2.2.1. Reconocimiento de Voz.....	7
2.2.2. Interpretación del Lenguaje Natural	8
2.3. Arquitecturas de software	10
2.3.1. Modelo-Vista-Controlador.....	10
2.3.2. Arquitectura modular	11
2.4. Lenguajes de programación, librerías y entorno de desarrollo	12
2.4.1. Lenguajes de programación.....	13
2.4.2. Librerías Python.....	13
2.4.3. Entorno de desarrollo	14
2.5. Estaciones de Trabajo de Audio Digital.....	14
2.6. Protocolos de comunicación	16
2.7. <i>Plugins</i> de procesamiento de audio en Reaper	16
2.8. <i>Scripting</i> en Reaper	18
2.9. Automatización de procesos.....	18
2.10. Modelos de inteligencia artificial aplicados a la música	19
2.11. API para modelos de Inteligencia Artificial	20
3. Especificaciones y restricciones de diseño	23
4. Descripción de la solución propuesta	25
4.1. Visión general del sistema.....	25
4.2. Análisis de requisitos.....	26
4.2.1. Requisitos funcionales y descripción completa de funcionalidades	26
4.2.2. Requisitos no funcionales	30
4.3. Arquitectura general	31
4.4. Desglose técnico por módulos	32
4.4.1. Módulo GUI	33
4.4.2. Módulo Comandos.....	36
4.4.3. Módulo Tarareos	40
4.4.4. Módulo Comunicaciones.....	42

4.4.5. Módulo análisis	47
4.4.6. Módulo IA.....	49
5. Resultados	55
5.1. Pruebas unitarias	55
5.2. Pruebas de caja negra.....	56
5.3. Pruebas de reconocimiento de voz	58
5.4. Pruebas de calidad musical	60
5.5. Pruebas de comunicación e integración con Reaper	61
5.6. Pruebas modelo de IA.....	64
6. Presupuesto	¡Error! Marcador no definido.
6.1. Costes materiales.....	65
6.2. Costes de personal.....	65
6.3. Costes totales.....	66
7. Impacto del proyecto	67
7.1. Contexto general del proyecto	67
7.2. Identificación del impacto	68
7.3. Conclusión sobre el impacto del proyecto	69
8. Conclusiones	71
8.1. Conclusiones generales	71
8.2. Trabajos futuros.....	73
9. Referencias	75
Anexo	81
A.1. Manual de usuario	81
A.1.1. Manual de instalación	81
A.1.2. Manual de uso	82

1. Introducción

1.1. Marco y motivación del proyecto

La producción musical ha experimentado una profunda transformación en las últimas décadas, permitiendo que tanto artistas profesionales como aficionados puedan crear música de calidad desde entornos domésticos. Sin embargo, el proceso de producción sigue presentando barreras técnicas significativas, especialmente para quienes no disponen de conocimientos avanzados en teoría musical, manejo de estaciones de trabajo de audio digital o utilización de instrumentos virtuales.

En este contexto, la integración de tecnologías de inteligencia artificial y procesamiento de lenguaje natural abre nuevas posibilidades para democratizar la creación musical. El presente proyecto se sitúa en la intersección entre la ingeniería telemática y la producción musical, proponiendo el desarrollo de un asistente inteligente orientado a facilitar la generación y edición de música mediante comandos de voz y tarareos. El objetivo es simplificar el acceso a herramientas avanzadas de producción, permitiendo que cualquier usuario pueda materializar sus ideas musicales sin necesidad de una formación técnica especializada.

La motivación principal de este trabajo radica en la necesidad de reducir la complejidad de los flujos de trabajo en producción musical, promoviendo la creatividad y eliminando barreras técnicas de entrada. El asistente propuesto busca automatizar tareas habituales y guiar al usuario durante el proceso creativo, ofreciendo recomendaciones personalizadas y facilitando la integración con entornos profesionales como Reaper, uno de los DAW (Digital Audio Workstation) más utilizados en el mundo de la producción musical.

A pesar de la democratización de las herramientas digitales, la mayoría de los usuarios aún se enfrenta a una curva de aprendizaje pronunciada para plasmar sus ideas musicales en un DAW. La interacción tradicional, basada en menús, atajos de teclado y configuraciones técnicas, puede resultar poco intuitiva y limitar la inmediatez creativa. La posibilidad de transformar un simple tarareo en una pista MIDI (Musical Instrument Digital Interface) editable, o de modificar instrumentos y efectos mediante comandos de voz, representa un avance significativo en accesibilidad y eficiencia.

El sistema desarrollado en este proyecto combina módulos para el reconocimiento de voz y el procesamiento de comandos, permitiendo que el usuario interactúe con Reaper mediante instrucciones orales. Una de las funcionalidades clave es la conversión automática de tarareos en archivos MIDI, facilitando que ideas musicales espontáneas puedan transformarse en material editable dentro del entorno DAW. La integración se apoya en scripts personalizados en Lua, que permiten ejecutar acciones concretas en Reaper, como la inserción de instrumentos virtuales, la aplicación de efectos o la gestión de pistas, todo ello con una comunicación entre el sistema y el DAW a través del protocolo OSC (Open Sound Control) y sin requerir conocimientos avanzados en producción musical. De este modo, el asistente actúa como un puente entre el usuario y el entorno de producción musical, permitiendo que incluso

quienes no dominan la teoría musical o el software especializado puedan experimentar y producir música de forma natural y accesible.

La visión general de la solución propuesta contempla un asistente modular, extensible y adaptable, capaz de guiar al usuario desde la idea inicial, ya sea un tarareo espontáneo o una instrucción verbal, hasta la producción final, automatizando procesos y adaptándose a diferentes niveles de experiencia.

1.2 Objetivos técnicos y académicos

Los objetivos del presente proyecto fin de grado se dividen en dos bloques principales: los objetivos técnicos, centrados en el desarrollo y la integración de la solución propuesta, y los objetivos académicos, enfocados en las competencias y aprendizajes adquiridos durante el proceso.

Los objetivos de este proyecto fin de carrera son, desde el punto de vista técnico:

- Satisfacer la necesidad de los usuarios de crear música de manera sencilla e intuitiva mediante comandos de voz, facilitando la generación y edición de melodías sin requerir conocimientos avanzados en producción musical.
- Automatizar el reconocimiento e interpretación de comandos de voz, logrando que el sistema detecte e interprete instrucciones orales de forma precisa y eficiente, permitiendo la ejecución de acciones específicas en el entorno de producción musical sin interacción manual adicional.
- Facilitar la transformación automatizada de tarareos y sonidos en datos musicales, ofreciendo la conversión de tarareos en archivos MIDI editables que puedan integrarse directamente en el flujo de trabajo de Reaper, permitiendo así que ideas musicales espontáneas se conviertan en material listo para su desarrollo y producción.
- Guiar y apoyar al usuario en el proceso creativo, proporcionando recomendaciones inteligentes y personalizadas sobre efectos, instrumentos y configuraciones, adaptadas al estilo y dinámica de la melodía generada, para optimizar la toma de decisiones creativas.
- Garantizar una integración fluida y robusta con la estación de trabajo de audio digital Reaper, asegurando la compatibilidad y la importación y edición directa de los archivos generados, sin necesidad de ajustes técnicos adicionales.

Desde el punto de vista académico, el proyectista adquiere las siguientes competencias y habilidades:

- Profundizar en el estudio y aplicación de tecnologías que utilicen inteligencia artificial, especialmente en el reconocimiento de voz y el procesamiento de lenguaje natural, así como en la integración de sistemas de automatización musical.

- Desarrollar competencias avanzadas en programación, tanto en Python como en Lua, y en el uso de protocolos de comunicación como OSC para la integración de sistemas heterogéneos.
- Adquirir experiencia práctica en el diseño y desarrollo de soluciones software orientadas a la producción musical.
- Aplicar metodologías de gestión de proyectos, documentación técnica y validación de resultados, mejorando la capacidad de organización, planificación y comunicación.
- Fomentar la capacidad de análisis crítico y resolución de problemas técnicos complejos, así como la adaptación a nuevas tecnologías y entornos multidisciplinares.
- Consolidar habilidades de investigación, autoaprendizaje y síntesis de información relevante para el desarrollo de soluciones innovadoras en el ámbito de la ingeniería telemática.

1.3 Estructura del resto de la memoria

En los próximos capítulos de la memoria se expone de forma ordenada y progresiva toda la información relevante para la comprensión y evaluación del proyecto.

En primer lugar, se realiza un estudio del marco tecnológico relevante para el desarrollo de un asistente musical inteligente, analizando las tecnologías actuales en reconocimiento de voz, procesamiento de lenguaje natural, conversión de audio a MIDI y automatización de tareas en estaciones de trabajo de audio digital como Reaper. Se justifica la elección de las herramientas y lenguajes empleados, valorando su idoneidad para los requisitos del sistema.

A continuación, se detallan las especificaciones y restricciones de diseño, definiendo los requisitos funcionales y no funcionales del asistente, así como las limitaciones técnicas y de integración que han condicionado las decisiones de desarrollo.

Seguidamente, se expone la descripción de la solución propuesta, profundizando en la arquitectura del sistema, las distintas funcionalidades del asistente, el funcionamiento de cada módulo y la interacción entre los diferentes componentes del sistema.

Posteriormente, se presentan los resultados obtenidos, mostrando ejemplos prácticos de uso, validación funcional y análisis de desempeño del asistente en distintos escenarios. Se incluyen casos de prueba que ilustran el funcionamiento real del sistema y su aportación a la accesibilidad y creatividad musical.

El siguiente apartado recoge el presupuesto, donde se estima el coste de los recursos materiales y de personal empleados, así como una valoración económica de los elementos implicados en el desarrollo del proyecto.

A continuación, se analiza el impacto del proyecto en el ámbito de la producción musical, valorando su potencial para democratizar el acceso a herramientas avanzadas, fomentar la creatividad y facilitar la experimentación sin barreras técnicas.

Finalmente, se exponen las conclusiones y se plantean posibles líneas de trabajo futuro, evaluando el grado de cumplimiento de los objetivos y señalando oportunidades de mejora o ampliación. La memoria se completa con las referencias bibliográficas y un anexo que incluye en manual de usuario, el cual da información al usuario de la instalación y configuración del asistente, además de información sobre el uso de este.

2. Marco tecnológico

En este apartado se presenta una visión general del estado actual de las tecnologías que conforman el contexto en el que se sitúa este proyecto. El conjunto de las siguientes tecnologías integradas entre si completa el funcionamiento global del sistema desarrollado.

2.1. Asistente Inteligente

Los asistentes inteligentes, son sistemas de software diseñados para asistir a los usuarios en la realización de tareas mediante interfaces naturales, como el lenguaje hablado o escrito. Estos sistemas integran tecnologías como el reconocimiento automático de voz, el procesamiento de lenguaje natural y, en muchos casos, modelos de aprendizaje automático, con el fin de interpretar, aprender y ejecutar comandos de manera eficiente [1].

El propósito fundamental de los asistentes inteligentes es facilitar una interacción más fluida y accesible entre las personas y los sistemas informáticos. Ejemplos ampliamente implantados en el mercado son Apple Siri, Amazon Alexa, Google Assistant o Microsoft Cortana, que permiten realizar tareas como búsquedas en internet, gestión de mensajes o control de dispositivos mediante comandos de voz [2].

El diseño de estos asistentes suele requerir la integración de diversos módulos tecnológicos:

- **Interfaz de entrada de voz:** Captura y transforma la señal de voz en texto utilizando técnicas de reconocimiento de voz.
- **Motor de interpretación semántica:** Analiza la entrada textual para identificar la intención del usuario y los elementos clave.
- **Gestor de diálogo:** Mantiene el contexto de la conversación y determina la siguiente acción a ejecutar.
- **Módulo de ejecución:** Conecta con sistemas externos para realizar la acción solicitada.
- **Interfaz de salida:** Presenta la respuesta al usuario, ya sea por voz o texto.

En la figura 1 se muestra la arquitectura de un asistente genérico:

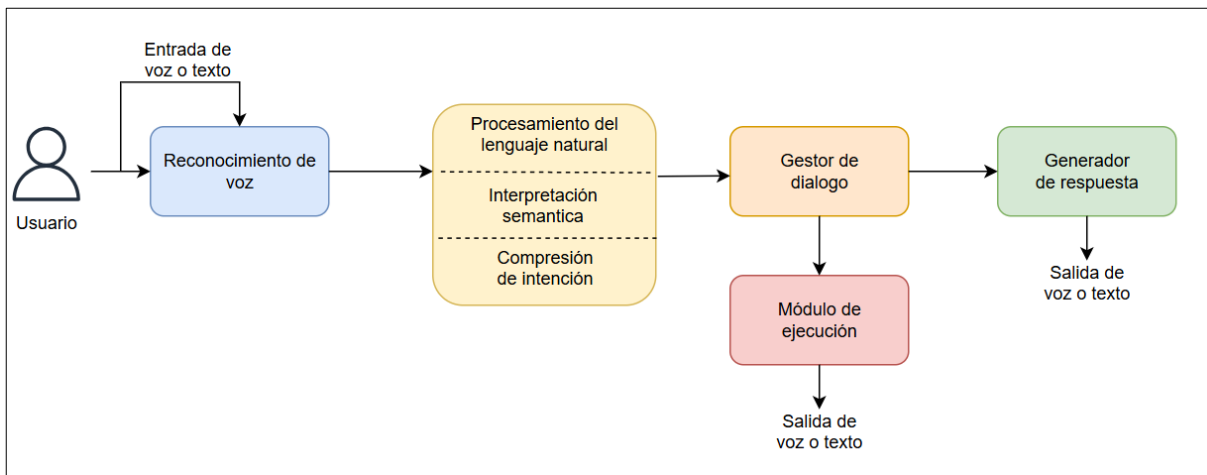


Figura 1. Arquitectura de asistente genérico

El desarrollo de asistentes inteligentes ha sido impulsado por los avances en redes neuronales profundas y el acceso a grandes volúmenes de datos, lo que ha permitido construir sistemas cada vez más precisos, adaptativos y capaces de gestionar múltiples tareas en contextos complejos [3]. Aunque inicialmente su uso se centró en el ámbito doméstico y de consumo, actualmente se extiende a sectores como la educación, la salud, la atención al cliente y el entretenimiento digital.

En el ámbito de la producción musical, el uso de asistentes inteligentes es muy limitado. Aunque existen herramientas comerciales que incorporan inteligencia artificial para algunas tareas, no existen actualmente asistentes inteligentes que permitan controlar y crear música en estaciones de trabajo de audio digital mediante voz y tarareos, ni que ayuden a músicos principiantes en sus creaciones directamente en el entorno de producción musical. Esta ausencia de soluciones integrales en el mercado evidencia la oportunidad y la lógica de abordar el proyecto desde la perspectiva de un asistente.

La elección de desarrollar este sistema como un asistente inteligente responde a la necesidad de ofrecer una herramienta accesible y lógica para el usuario promedio, que suele ser principiante y busca realizar tareas específicas de la forma más sencilla posible. El enfoque de asistente es clave, ya que facilita la guía y el apoyo durante el proceso creativo, permitiendo que cualquier persona pueda materializar sus ideas musicales y aprovechar las ventajas de la producción digital sin enfrentarse a barreras técnicas complejas. De este modo, se contribuye a democratizar la producción musical digital y a fomentar la creatividad en un entorno que, hasta ahora, resulta poco accesible para usuarios inexpertos.

2.2. Reconocimiento de voz e interpretación de comandos

La interacción por voz constituye uno de los pilares fundamentales en la evolución de los sistemas de interacción persona-máquina, permitiendo a los usuarios comunicarse con dispositivos y aplicaciones de forma natural, rápida y sin necesidad de interfaces físicas tradicionales.

Este proceso se articula en dos etapas tecnológicas principales: el reconocimiento automático de voz y la interpretación de comandos mediante procesamiento de lenguaje natural.

2.2.1. Reconocimiento de Voz

El reconocimiento de voz es la tecnología que permite a un sistema convertir señales de audio provenientes del habla humana en texto digital interpretable por una máquina. Este proceso implica varias fases: la captura de la señal de audio, su preprocesamiento para reducir ruido y normalizar el volumen, la extracción de características relevantes y la comparación de estos patrones con modelos estadísticos o redes neuronales previamente entrenadas [4].

Existen dos grandes enfoques tecnológicos para el reconocimiento de voz: los modelos ocultos de Markov y las redes neuronales profundas. Los modelos de Markov segmentan el habla en fonemas y modelan las probabilidades de transición entre ellos, mientras que las redes neuronales permiten capturar dependencias temporales y contextuales más complejas, incrementando la precisión incluso en ambientes ruidosos o con diferentes acentos [5].

El reconocimiento de voz se utiliza ampliamente en asistentes virtuales, sistemas de dictado, control por voz en automóviles, aplicaciones móviles y dispositivos inteligentes, facilitando la accesibilidad y la automatización de tareas cotidianas

Vosk es una de las tecnologías representativas en este campo. Es un motor de reconocimiento automático de voz de código abierto que permite la transcripción de voz a texto en tiempo real y soporta múltiples idiomas. Su principal ventaja es la capacidad de funcionar de manera local, sin requerir conexión a internet, lo que mejora la privacidad y reduce la latencia. Vosk utiliza modelos acústicos y lingüísticos entrenados con herramientas como Kaldi, combinando redes neuronales profundas y modelos estadísticos para lograr una alta precisión y adaptabilidad en diferentes escenarios [6].

En la figura 2 se muestra una arquitectura típica de funcionamiento de un sistema de reconocimiento de voz, además de mostrar la arquitectura específica de Vosk:

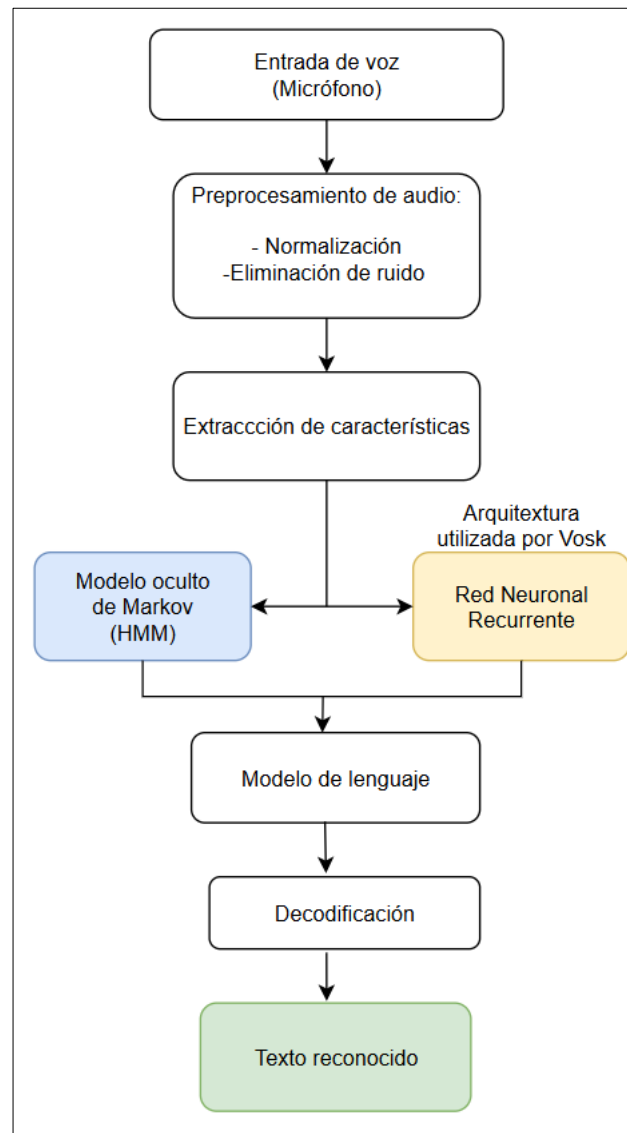


Figura 2. Arquitectura sistema de reconocimiento de voz

2.2.2. Interpretación del Lenguaje Natural

Una vez obtenida la transcripción textual, el siguiente desafío tecnológico es dotar a este tipo de sistemas de la capacidad para comprender el significado de los comandos emitidos en lenguaje natural. Aquí intervienen técnicas de procesamiento de lenguaje natural, que permiten identificar la intención del usuario y extraer entidades o parámetros relevantes de la frase.

La interpretación de comandos implica varios pasos. Inicialmente, el texto transcrito se somete a un preprocesamiento lingüístico de normalización, tokenización y análisis morfosintáctico para facilitar su análisis posterior.

A continuación, se aplican algoritmos de clasificación y extracción, que pueden basarse en reglas, modelos estadísticos o técnicas de aprendizaje automático.

- **Identificación de intenciones:** El sistema determina cuál es la acción principal que el usuario desea realizar.
- **Extracción de entidades:** El sistema localiza y clasifica datos concretos dentro de la instrucción.

Los sistemas modernos de interpretación de comandos suelen emplear modelos de aprendizaje automático y aprendizaje profundo entrenados sobre grandes volúmenes de datos, lo que les permite adaptarse a distintas formas de expresión, gestionar ambigüedades y ofrecer respuestas precisas incluso en contextos complejos. Además, pueden incorporar mecanismos de gestión de contexto conversacional, permitiendo mantener diálogos más naturales y fluidos con el usuario.

Entre las plataformas más destacadas para la interpretación de comandos se encuentra **Wit.ai**, que permite entrenar modelos personalizados para identificar intenciones y extraer entidades a partir de frases en lenguaje natural. Wit.ai ofrece una API (Application programming interface) sencilla para integrar su funcionalidad en diferentes aplicaciones y soporta el entrenamiento y mejora continua de los modelos en la nube, facilitando la adaptación a dominios y contextos específicos [7].

En la figura 3 se muestra un diagrama que resume el flujo de procesamiento implicado en la interpretación de lenguaje natural, mediante un ejemplo de uso y centrado en el funcionamiento de Wit.ai:

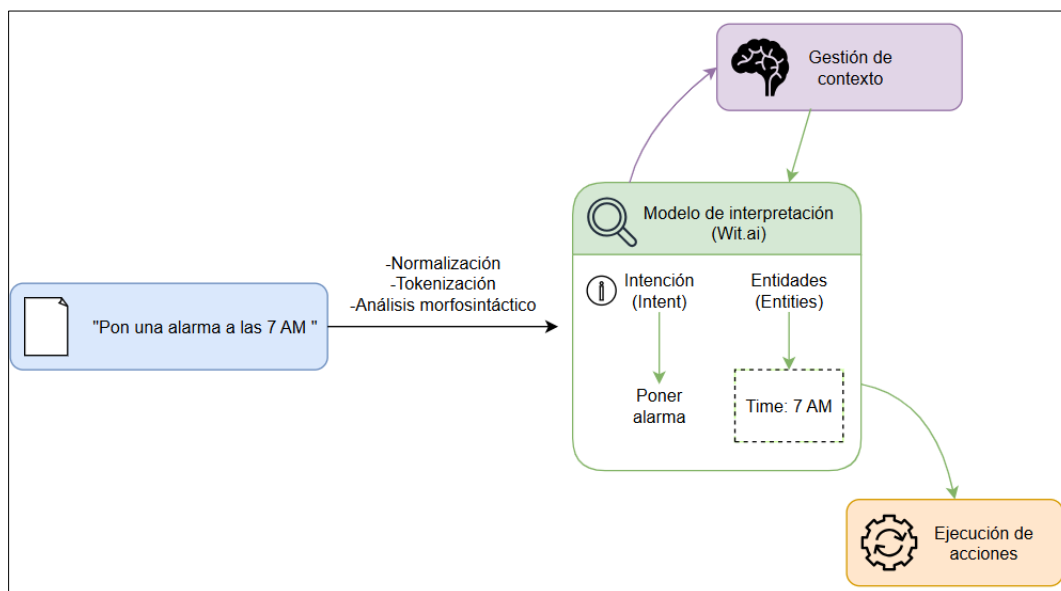


Figura 3. Flujo de funcionamiento en la interpretación del lenguaje natural

2.3. Arquitecturas de software

La arquitectura de software constituye el diseño de más alto nivel de un sistema, definiendo cómo se organizan, interactúan y comunican los distintos componentes que lo conforman. Actúa como un plano que guía el desarrollo, facilita la integración de las distintas piezas y establece las bases para la escalabilidad, el mantenimiento y la calidad del sistema.

La elección de una arquitectura adecuada responde tanto a los objetivos funcionales del proyecto como a requisitos de flexibilidad, colaboración y facilidad de evolución futura. En este marco, dos de los enfoques más utilizados y complementarios en el desarrollo de aplicaciones modernas son la arquitectura modular y el patrón Modelo-Vista-Controlador.

2.3.1. Modelo-Vista-Controlador

El patrón Modelo-Vista-Controlador [8] es una arquitectura de software ampliamente adoptada para la construcción de interfaces de usuario y sistemas interactivos. Su principal objetivo es separar claramente la lógica de presentación, la gestión de la interacción y la lógica de negocio, favoreciendo así la mantenibilidad, la reutilización y la evolución del software.

En este tipo de arquitectura, la Vista es responsable de mostrar la información y recibir la interacción del usuario, el Controlador actúa como intermediario, gestionando los eventos y coordinando la lógica de flujo y el Modelo encapsula los datos y la lógica de negocio, gestionando el estado y las operaciones principales del sistema. En la figura 4 se muestra el funcionamiento y disposición de esta arquitectura:

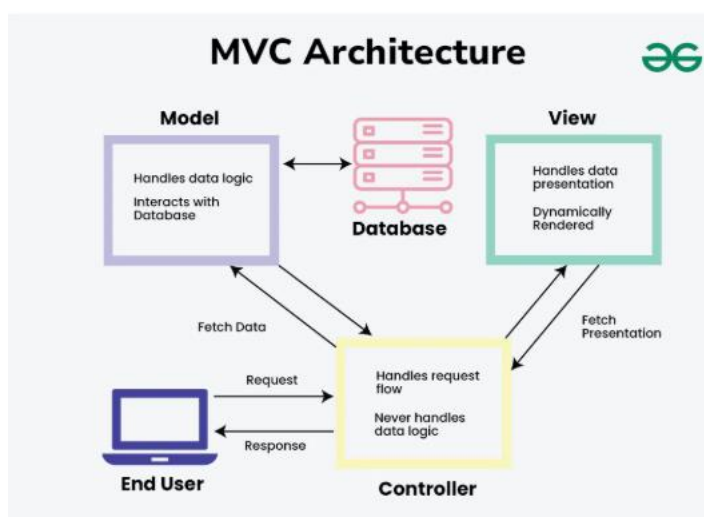


Figura 4. Arquitectura MVC genérica [9]

Esta separación permite que los cambios en la interfaz gráfica no afecten a la lógica interna, y viceversa, facilitando la adaptación a nuevas necesidades o tecnologías. Además, esta estructura se integra de forma natural con la modularidad, ya que cada componente del Modelo puede implementarse como un módulo independiente, reforzando la escalabilidad y la claridad del sistema.

La elección del patrón Modelo-Vista-Controlador y de la arquitectura modular en proyectos de ingeniería de software responde a la necesidad de construir sistemas flexibles, fáciles de mantener y capaz de evolucionar, permitiendo que la interfaz gráfica y la lógica de procesamiento se desarrollen y mejoren de manera independiente y colaborativa.

2.3.2. Arquitectura modular

La programación modular es una técnica de diseño de software que consiste en dividir un programa complejo en partes más pequeñas, autónomas y manejables llamadas módulos. Cada módulo se encarga de una tarea específica y bien definida, lo que facilita la organización, el desarrollo, la comprensión y el mantenimiento del código.

Esta metodología se originó en la década de 1960 como una evolución de la programación estructurada, para abordar problemas de mayor tamaño y complejidad que requerían una mayor organización y reutilización del código.

En la práctica, la modularidad implica que cada módulo puede ser desarrollado, probado y mantenido de forma independiente, siempre que respete una interfaz bien definida para comunicarse con otros módulos. Esto reduce la redundancia, permite el trabajo colaborativo entre equipos y mejora la escalabilidad de los sistemas, ya que los módulos pueden ser reutilizados en diferentes proyectos o ampliados sin afectar al resto de la aplicación.

El proceso de diseño modular suele seguir el principio de análisis descendente, descomponiendo el problema principal en subproblemas más simples hasta que cada módulo sea lo suficientemente sencillo de implementar. En la estructura típica de un programa modular, existe un módulo principal que coordina la ejecución y llama a los módulos auxiliares, los cuales pueden a su vez invocar a otros módulos según sea necesario. En la figura 5 se muestra un ejemplo de diagrama de arquitectura modular:

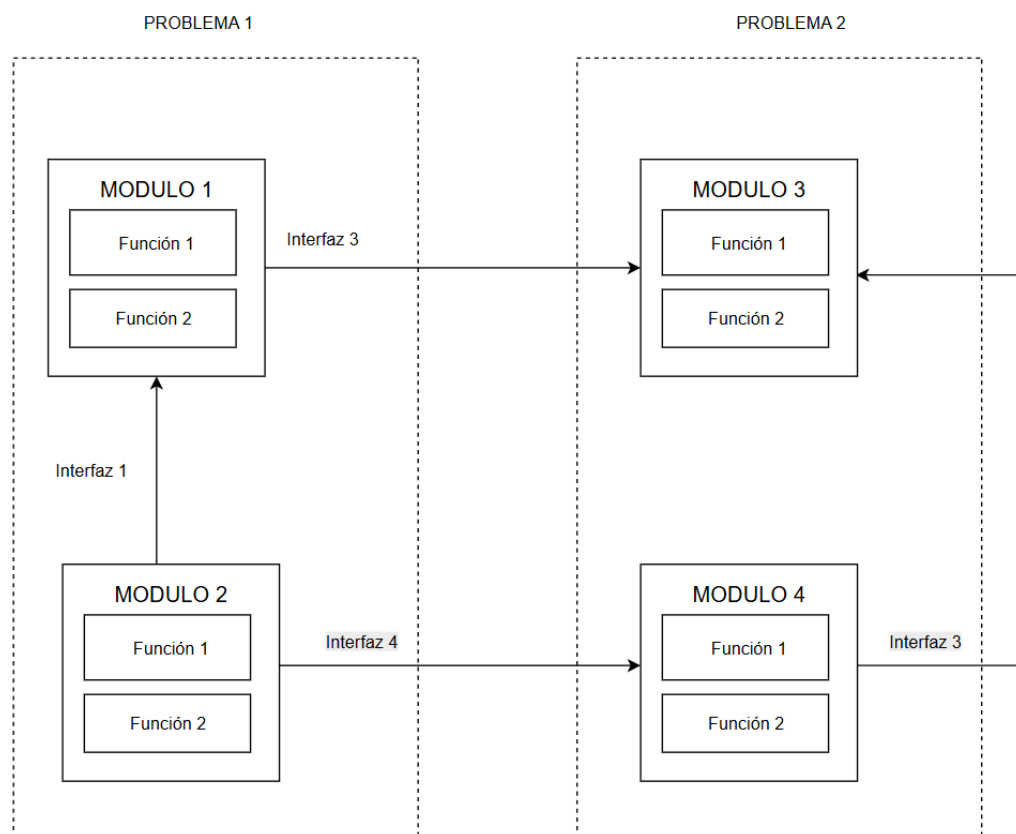


Figura 5. Diagrama de ejemplo de arquitectura modular

La modularidad no solo se limita a funciones o procedimientos, sino que puede abarcar componentes más amplios, como clases, paquetes o incluso servicios independientes, dependiendo del lenguaje y la arquitectura utilizada.

Este enfoque resulta especialmente útil en proyectos que requieren escalabilidad, mantenimiento a largo plazo y la posibilidad de incorporar nuevas funcionalidades sin reescribir el sistema completo [10].

2.4. Lenguajes de programación, librerías y entorno de desarrollo

El desarrollo de muchas soluciones tecnológicas en la actualidad se apoya en lenguajes de programación versátiles, librerías especializadas y entornos de desarrollo integrados que permiten abordar problemas complejos de forma eficiente y escalable. Estos componentes forman parte de la infraestructura lógica y de software que constituye el núcleo de muchos sistemas informáticos modernos.

2.4.1. Lenguajes de programación

Python [11] es un lenguaje de programación interpretado, de alto nivel y propósito general, ampliamente utilizado en diversos campos como la automatización, el análisis de datos, la inteligencia artificial, el desarrollo web y el procesamiento de señales. Destaca por su sintaxis clara, su gran comunidad de usuarios y la disponibilidad de una extensa colección de librerías, lo que facilita el desarrollo rápido de prototipos y la integración de soluciones complejas.

Lua [12] es un lenguaje de *scripting* ligero y eficiente, diseñado para ser embebido en aplicaciones y ofrecer capacidades de extensión y personalización. Su bajo consumo de recursos y su facilidad de integración lo han convertido en una opción popular para la automatización y la personalización de *software* en diferentes sectores tecnológicos.

2.4.2. Librerías Python

Dentro del marco tecnológico de este proyecto, se han identificado una serie de librerías de Python especialmente relevantes por su utilidad y aplicación en tareas de procesamiento de datos, audio y creación de interfaces gráficas. Estas librerías han sido seleccionadas por su robustez, su amplia adopción y su capacidad para facilitar el desarrollo de soluciones tecnológicas avanzadas. A continuación, se describen brevemente las principales:

- **PyQt5**: [13] Es un marco de trabajo ampliamente utilizado para el desarrollo de interfaces gráficas de usuario en Python. Permite crear aplicaciones visuales complejas y multiplataforma de manera eficiente, integrando botones, menús, cuadros de diálogo y otros elementos interactivos necesarios para la interacción usuario-sistema.
- **sounddevice**: [14] Esta librería proporciona una interfaz sencilla para la reproducción y grabación de audio en tiempo real. Es especialmente útil para aplicaciones que requieren capturar audio del micrófono o reproducir sonidos generados o procesados por el programa, y se integra fácilmente con *arrays* de datos de NumPy.
- **requests**: [15] Es la librería estándar para realizar peticiones HTTP (Hypertext Transfer Protocol) en Python. Permite conectar aplicaciones con servicios web y APIs externas, facilitando la obtención y envío de datos a través de Internet de manera sencilla y segura.
- **numpy**: [16] Fundamental para el cálculo numérico y la manipulación eficiente de grandes volúmenes de datos, numpy es la base para la mayoría de operaciones matemáticas y de procesamiento de señales en Python. Su estructura de *arrays* multidimensionales es clave para el análisis de audio y datos musicales.
- **aubio**: [17] Especializada en la extracción de características musicales a partir de señales de audio, aubio permite detectar eventos como el *tempo*, los pulsos, las notas o los ataques en una grabación. Es especialmente útil en tareas de análisis musical automático.

- **mido:** [18] Esta librería facilita el trabajo con mensajes y archivos MIDI, permitiendo leer, escribir y manipular datos musicales en este formato estándar. Es muy utilizada en aplicaciones que requieren comunicación con instrumentos electrónicos, controladores MIDI o edición de secuencias musicales.
- **librosa:** [19] Considerada una de las bibliotecas más avanzadas para el análisis y procesamiento de música y audio en Python, librosa ofrece herramientas para la extracción de características espectrales, segmentación, transformación de señales y visualización de datos musicales. Es ampliamente utilizada en investigación y desarrollo de aplicaciones de análisis musical.
- **music21:** [20] Proporciona un conjunto de herramientas para el análisis, procesamiento y generación de información musical estructurada, como partituras, acordes, escalas y análisis armónico. Es especialmente útil para el estudio y manipulación de música en formato simbólico, permitiendo realizar análisis musicológicos y generar nuevas composiciones.

Estas librerías forman parte del conjunto de herramientas tecnológicas más relevantes y utilizadas en el desarrollo de aplicaciones modernas que requieren procesamiento de audio, análisis musical y creación de interfaces gráficas.

2.4.3. Entorno de desarrollo

Visual Studio Code [21] es un editor de código fuente multiplataforma y altamente configurable, que se ha consolidado como uno de los entornos de desarrollo más populares en la actualidad. Ofrece soporte para numerosos lenguajes de programación, herramientas de depuración, integración con sistemas de control de versiones y una amplia gama de extensiones que facilitan el desarrollo y mantenimiento de software en distintos ámbitos tecnológicos.

2.5. Estaciones de Trabajo de Audio Digital

Las DAW son aplicaciones software que permiten la grabación, edición, mezcla y masterización de audio digital, integrando en un solo entorno herramientas para manipular múltiples pistas, aplicar efectos, gestionar instrumentos virtuales, etc. Gracias a la evolución de la informática, los DAW han transformado la producción musical y el tratamiento de audio, ofreciendo capacidades antes reservadas a estudios profesionales a cualquier usuario con un ordenador personal.

Entre los DAW más reconocidos y utilizados en la actualidad destacan FL Studio, Ableton Live y Reaper, cada uno con enfoques y características particulares que los hacen adecuados para distintos perfiles y necesidades [22]:

- **FL Studio** [23] es una DAW desarrollada por Image-Line, reconocida por su potente secuenciador por pasos que facilita la creación rápida de patrones rítmicos y melodías. Su interfaz visual y orientada a la composición por bloques es especialmente apreciada

en géneros como la música electrónica, hip-hop y pop. FL Studio incluye una amplia gama de *plugins* nativos, desde sintetizadores hasta efectos de procesamiento, lo que permite a los usuarios diseñar sonidos complejos sin necesidad de software adicional. Además, ofrece soporte para grabación multipista y edición de audio, aunque su enfoque principal sigue siendo la creación basada en bucles y patrones. Su sistema de automatización es intuitivo, permitiendo controlar parámetros de efectos e instrumentos de forma dinámica. Sin embargo, para flujos de trabajo que requieren edición avanzada o integración profunda con hardware externo, FL Studio puede presentar ciertas limitaciones.

- **Ableton Live** [24], desarrollado por Ableton AG, se ha consolidado como una herramienta versátil tanto para la producción en estudio como para la interpretación en directo. Su característica más distintiva es la Vista Session que permite lanzar clips de audio y MIDI de forma no lineal, facilitando la improvisación y la construcción dinámica de arreglos musicales. Esta funcionalidad la convierte en la DAW preferida de DJs, músicos electrónicos y artistas de performance. Ableton Live cuenta con un motor de audio de baja latencia, soporte para distintos tipos de *plugins*, y una integración profunda con hardware controlador como Ableton Push, que optimiza la interacción en tiempo real. Además, su compatibilidad con Max for Live amplía sus capacidades permitiendo crear dispositivos personalizados para procesamiento de audio, MIDI y control. Aunque es muy potente en manipulación creativa, su enfoque puede resultar menos adecuado para proyectos que demanden grabación multipista tradicional o edición minuciosa de audio.
- **Reaper** [25], desarrollado por Cockos, destaca por ser una DAW ligera, altamente configurable y con un modelo de licencia accesible. Su arquitectura modular y eficiente permite gestionar proyectos con gran número de pistas y efectos sin un consumo excesivo de recursos. Reaper ofrece soporte para una amplia variedad de formatos de audio y *plugins*, y su motor de audio es reconocido por su estabilidad y calidad. Uno de sus mayores puntos fuertes es el soporte avanzado para scripting mediante lenguajes como Lua y Python, permitiendo a los usuarios automatizar tareas complejas, personalizar la interfaz y extender funcionalidades sin necesidad de programación profunda del núcleo del DAW. Esta capacidad convierte a Reaper en una plataforma ideal para desarrolladores y usuarios técnicos que requieren un control granular sobre su flujo de trabajo. Además, su comunidad activa y la disponibilidad de numerosos scripts y extensiones contribuyen a su flexibilidad y adaptabilidad a diferentes entornos profesionales.

La elección de Reaper en el marco tecnológico de este proyecto se fundamenta en su potente sistema de scripting y su arquitectura abierta, que facilitan la integración con otros sistemas y la automatización avanzada de procesos dentro del DAW. Estas características permiten adaptar la plataforma a necesidades técnicas específicas y desarrollar soluciones personalizadas de manera eficiente, lo que resulta especialmente relevante en proyectos que requieren flexibilidad y control sobre el flujo de trabajo.

2.6. Protocolos de comunicación

En el desarrollo de aplicaciones y sistemas que requieren interacción entre diferentes programas o dispositivos, los protocolos de comunicación desempeñan un papel fundamental. Permiten transmitir información y órdenes de manera estructurada y eficiente, facilitando la interoperabilidad y la automatización de procesos. En el contexto del marco tecnológico de este proyecto destacan los siguientes protocolos [26]:

- **MIDI** [27] es un protocolo estándar que define cómo se comunican instrumentos electrónicos, ordenadores y otros dispositivos para intercambiar información musical y de control. MIDI transmite mensajes que indican, por ejemplo, cuándo se pulsa o libera una nota, la intensidad, cambios de programa o variaciones de parámetros de control. Es importante destacar que MIDI no transmite audio, sino datos que describen acciones musicales o de control. Además, estos mensajes pueden almacenarse en archivos con extensión .mid, conocidos como archivos MIDI. Un archivo .mid es un formato estándar que guarda secuencias de eventos MIDI, permitiendo que diferentes programas y dispositivos reproduzcan, editen o interpreten la información musical contenida en ellos. Esta capacidad de almacenar y transferir información musical ha convertido a MIDI en un pilar de la producción y edición musical digital, así como en la automatización de tareas en numerosos entornos tecnológicos.
- **OSC** [28] es un protocolo abierto diseñado para facilitar la comunicación entre aplicaciones y dispositivos a través de redes, como ordenadores, sintetizadores y controladores multimedia. OSC define un formato de mensajes estructurados que pueden contener rutas y argumentos personalizados, y se transmite habitualmente mediante paquetes a través de UDP (User Datagram Protocol) o TCP (Transmission Control Protocol). Su arquitectura cliente-servidor permite que una aplicación actúe como cliente enviando mensajes a un servidor que los recibe y ejecuta. OSC se caracteriza por su flexibilidad, permitiendo definir comandos específicos y transmitir datos complejos o acciones concretas entre programas. Por ejemplo, es común utilizar OSC para enviar órdenes desde una aplicación externa a un software como Reaper, permitiendo controlar funciones como la reproducción, el volumen o la activación de pistas de forma remota y automatizada. Esta capacidad de personalización y comunicación en red ha hecho que OSC sea ampliamente adoptado en entornos multimedia, instalaciones interactivas y control de software desde interfaces personalizadas.

2.7. Plugins de procesamiento de audio en Reaper

En el marco de la producción musical digital, los *plugins* de procesamiento de audio constituyen una herramienta esencial para modificar, mejorar y personalizar el sonido dentro de una DAW. Reaper, destaca por su compatibilidad con una amplia gama de *plugins* y por incluir un conjunto propio de efectos nativos que cubren las necesidades fundamentales de cualquier flujo de trabajo musical.

Reaper soporta los principales formatos de *plugins* de la industria, como VST (Virtual Studio Technology), VST3 y JSFX, lo que permite integrar tanto desarrollos de terceros como efectos propios del programa. Sin embargo, una de sus principales fortalezas radica en la calidad y versatilidad de sus *plugins* nativos, conocidos como ReaPlugs [29].

Entre los efectos más representativos y utilizados en Reaper se encuentran:

- **ReaComp:** Compresor versátil que permite controlar la dinámica de las señales de audio, ajustando el rango dinámico y aportando cohesión a las mezclas.
- **ReaVerb y ReaVerbate:** *Plugins* de reverberación que permiten simular espacios acústicos y añadir profundidad a las pistas. Mientras ReaVerb ofrece un enfoque más avanzado basado en convolución, ReaVerbate proporciona una solución sencilla y eficiente para añadir reverberación de tipo algorítmico.
- **ReaEQ:** Ecualizador paramétrico multibanda que permite esculpir el espectro de frecuencias de cualquier señal de audio, facilitando tanto la corrección como la creatividad en la mezcla.
- **Stereo Enhancer:** Efecto destinado a ampliar o modificar la imagen estéreo de una pista, útil para dar mayor amplitud y presencia a elementos concretos dentro de la mezcla.
- **ReaSynth:** Sintetizador básico incluido en Reaper, que permite generar sonidos electrónicos y es especialmente útil para pruebas, diseño sonoro sencillo o la creación de líneas melódicas y acompañamientos.
- **DSK Virtuoso:** [30] Instrumento virtual orquestal gratuito que ofrece hasta seis capas de instrumentos simultáneos, permitiendo la creación de arreglos complejos y ricos en matices. Incluye más de 40 instrumentos agrupados en seis categorías. Cada capa es asignable a un canal MIDI diferente y cuenta con controles independientes de volumen, panorámica y envolvente, además de efectos integrados como coro, eco y reverberación. Su interfaz sencilla facilita la combinación y ajuste de sonidos, siendo una herramienta versátil para la composición y producción musical en entornos DAW como Reaper.

Estos *plugins* nativos están optimizados para un bajo consumo de recursos y ofrecen una integración total con el entorno de Reaper, permitiendo su automatización, modulación y control mediante scripts o superficies de control externas. Además, su diseño transparente y su enfoque modular facilitan tanto el aprendizaje por parte de usuarios principiantes como la personalización avanzada por parte de usuarios experimentados.

El ecosistema de *plugins* en Reaper se completa con el acceso a una comunidad activa que desarrolla y comparte utilidades adicionales, pero el conjunto de efectos nativos ya cubre de manera eficiente las necesidades básicas y medias de procesamiento de audio y MIDI en la mayoría de los proyectos de producción musical.

2.8. Scripting en Reaper

El *scripting* en estaciones de trabajo de audio digital es una herramienta fundamental para ampliar la funcionalidad del software, automatizar tareas repetitivas y personalizar flujos de trabajo. En este contexto, ReaScript [31] es el sistema de scripting integrado en Reaper que permite a los usuarios crear scripts para controlar y modificar prácticamente cualquier aspecto del DAW.

ReaScript soporta varios lenguajes de programación, entre los que destacan Python, Lua y EEL. Esta variedad ofrece flexibilidad a los usuarios, permitiendo desde scripts sencillos para automatizar tareas básicas hasta desarrollos complejos que interactúan con múltiples elementos del proyecto. Python y Lua, en particular, son lenguajes ampliamente conocidos y utilizados en la comunidad de desarrollo, lo que facilita la adopción y el aprendizaje.

Mediante ReaScript, es posible acceder a una extensa API que expone funciones para manipular pistas, clips, efectos, parámetros, automatizaciones y mucho más. Esto abre un amplio abanico de posibilidades para personalizar la interfaz, crear herramientas específicas, integrar Reaper con otros programas o dispositivos, y optimizar procesos de producción musical [32].

2.9. Automatización de procesos

La automatización de procesos [33] en el ámbito de la tecnología digital se refiere a la capacidad de los sistemas para ejecutar de forma encadenada y autónoma una serie de tareas que, tradicionalmente, requerirían intervención manual. Este enfoque resulta especialmente relevante en entornos donde la gestión eficiente de aplicaciones, archivos y parámetros es fundamental para optimizar el flujo de trabajo y reducir la posibilidad de errores humanos.

La automatización ha evolucionado desde sistemas basados en reglas y scripts predefinidos hasta soluciones más flexibles y adaptativas, apoyadas en algoritmos y herramientas capaces de gestionar procesos completos de manera desatendida. Estas tecnologías permiten, por ejemplo, que, tras la activación de un comando o evento inicial, se desencadenen automáticamente acciones como la apertura de aplicaciones especializadas, la importación y organización de archivos, y la aplicación de configuraciones o efectos preestablecidos, todo ello sin requerir pasos intermedios manuales.

La integración de la automatización en los flujos de trabajo digitales no solo agiliza la ejecución de tareas repetitivas, sino que también facilita la coordinación entre diferentes aplicaciones y sistemas, permitiendo una mayor coherencia y escalabilidad en los procesos. Además, la automatización es un elemento clave para garantizar la eficiencia operativa en entornos donde la rapidez y la precisión en la gestión de recursos digitales son esenciales.

2.10. Modelos de inteligencia artificial aplicados a la música

En los últimos años, la inteligencia artificial ha irrumpido de manera decisiva en ámbitos creativos, y la música no ha sido una excepción. Este campo se encuentra en plena expansión, con desarrollos que buscan generar, transformar y analizar música de manera autónoma o asistida. Los modelos de IA (Inteligencia Artificial) aplicados a la música combinan avances en aprendizaje profundo, redes neuronales y procesamiento de secuencias, ofreciendo herramientas capaces de crear melodías, armonías, ritmos o piezas completas a partir de instrucciones textuales, ejemplos de audio o estructuras simbólicas. A pesar de los avances, se trata de un área todavía en evolución, en la que los resultados varían en calidad, nivel de control y aplicabilidad práctica, y donde persisten retos relacionados con la coherencia, la expresividad y la personalización musical.

Entre los desarrollos más relevantes destacan los siguientes modelos generativos:

- **Magenta (Google Brain):** Proyecto de investigación de código abierto que explora la creatividad musical mediante técnicas de aprendizaje profundo. Utiliza redes neuronales recurrentes y modelos generativos entrenados con grandes colecciones de música, y permite la generación de melodías, armonías y patrones rítmicos. Su aportación más destacada es la integración con lenguajes como TensorFlow y la disponibilidad de librerías prácticas que han impulsado numerosas investigaciones académicas y aplicaciones experimentales [34], [35].
- **MuseNet (OpenAI):** Modelo basado en redes neuronales profundas capaz de generar composiciones polifónicas con hasta diez instrumentos simultáneos y en estilos muy variados, desde música clásica hasta géneros modernos. Su funcionamiento se centra en la predicción de secuencias en formato MIDI, lo que facilita la manipulación posterior en estaciones de trabajo digitales. Aunque el proyecto ya no está disponible públicamente, supuso un hito al demostrar la capacidad de los modelos de lenguaje para abordar estructuras musicales complejas [36].
- **Jukebox (OpenAI):** Representa un avance significativo respecto a los modelos simbólicos, ya que es capaz de generar directamente audio crudo con instrumentación y voces cantadas. Utiliza una arquitectura jerárquica de redes neuronales para modelar tanto la melodía como el timbre y el estilo de los intérpretes. Sus limitaciones actuales son el elevado coste computacional y el reducido nivel de control sobre los detalles de la composición, pero abrió nuevas posibilidades para la creación de música realista con IA [37].
- **MusicLM (Google, 2023):** Modelo de última generación que genera música a partir de descripciones textuales. Destaca por su capacidad para mantener coherencia durante varios minutos y por admitir entradas multimodales, como ejemplos de audio que definen el estilo deseado. Representa uno de los avances más prometedores en la democratización de la creación musical asistida por IA, aunque plantea desafíos éticos y legales relacionados con los derechos de autor [38].

- **Riffusion:** Proyecto innovador que utiliza modelos de difusión, originalmente diseñados para la generación de imágenes, para crear música. Su método consiste en generar espectrogramas que posteriormente son convertidos en audio. Riffusion se orienta a la producción de bucles y fragmentos musicales breves, con una clara vocación experimental y creativa [39], [40].

Además de estos sistemas centrados en la generación, existen modelos orientados al análisis musical, que cumplen un papel complementario al facilitar la extracción de información estructural a partir de grabaciones de audio:

- **Onsets and Frames:** Modelo basado en aprendizaje profundo para la transcripción automática de interpretaciones de piano. Combina redes convolucionales y recurrentes para detectar tanto los ataques como la duración de las notas, logrando un rendimiento muy superior al de métodos tradicionales [41].
- **Crepe:** Modelo especializado en la detección de la altura tonal a partir de audio monofónico. Es ampliamente reconocido por su precisión, incluso en entornos con ruido o distorsión, y se utiliza en múltiples aplicaciones de análisis musical y educación [42].

En conjunto, estos modelos muestran la diversidad de enfoques de la IA musical, desde la generación de composiciones hasta el análisis automático de audio. Aunque no sustituyen el proceso creativo humano, aportan herramientas que amplían las posibilidades expresivas de músicos, productores y compositores.

2.11. API para modelos de Inteligencia Artificial

OpenRouter [43] es una plataforma que centraliza el acceso a cientos de modelos de lenguaje de gran tamaño a través de una única API, facilitando la integración de modelos de inteligencia artificial en aplicaciones y sistemas digitales. Su funcionamiento se basa en ofrecer una interfaz unificada y compatible con el estándar de OpenAI, lo que permite a los desarrolladores utilizar cualquier SDK (Software Development Kit) o biblioteca diseñada para OpenAI sin apenas modificaciones para cada modelo. La autenticación se realiza mediante API keys, y la comunicación se establece a través de *endpoints* RESTful, tanto para generación de texto como para tareas multimodales, lo que simplifica enormemente la integración y el mantenimiento del sistema. Cambiar de modelo es tan sencillo como modificar el identificador del modelo en el código Python, sin necesidad de reescribir la lógica de la aplicación, lo que aporta una gran flexibilidad y capacidad de adaptación a diferentes necesidades o pruebas.

Además, OpenRouter ofrece una amplia variedad de modelos muy similares a los comerciales de grandes compañías, los cuales pueden utilizarse de forma gratuita para desarrollo a través de esta API. Estos modelos pueden filtrarse a través de la plataforma de manera sencilla y en función de varias características como el tipo de entrada, el tamaño de contexto del modelo o el precio por uso, entre otras.

En la figura 6 se ilustra la facilidad de exploración de modelos que ofrece la plataforma:

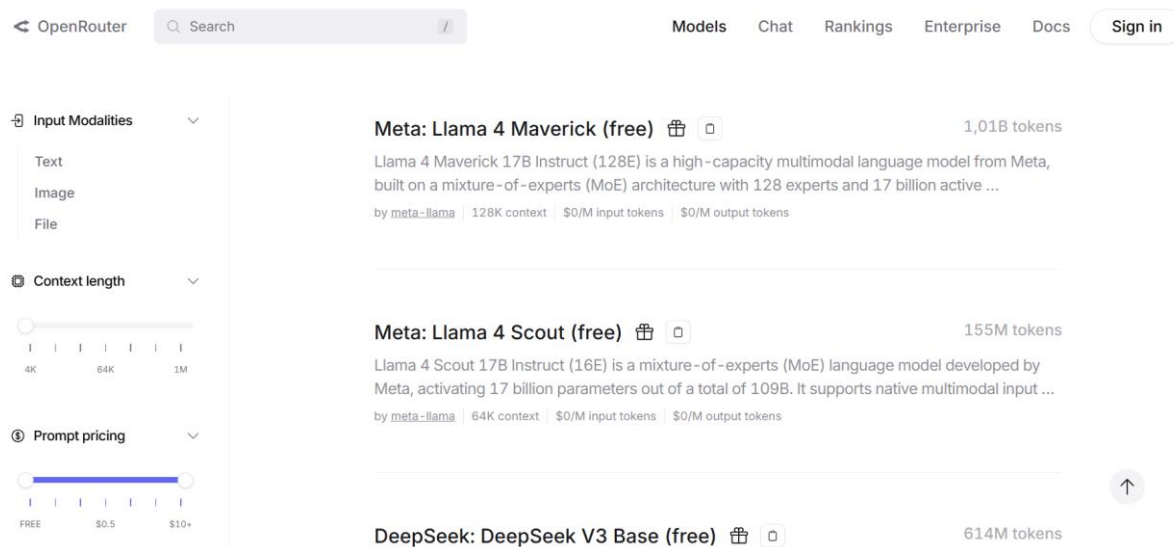


Figura 6. Interfaz de filtrado de modelos en OpenRouter

Una de las características más relevantes de OpenRouter es la posibilidad de monitorizar y comparar el rendimiento de los distintos modelos disponibles. La plataforma permite acceder a métricas como el coste total por modelo, el tiempo medio de respuesta, la eficiencia, el porcentaje de respuestas correctas y fallidas, el tiempo de disponibilidad, el rendimiento y las aplicaciones que utilizan cada modelo. Esta información es fundamental para tomar decisiones informadas sobre qué modelo utilizar en función de los requisitos del proyecto, ya sea priorizando la calidad de las respuestas, la velocidad, la disponibilidad o el coste. En la figura 7 se muestran algunas de las estadísticas que permiten la comparación crítica entre modelos:

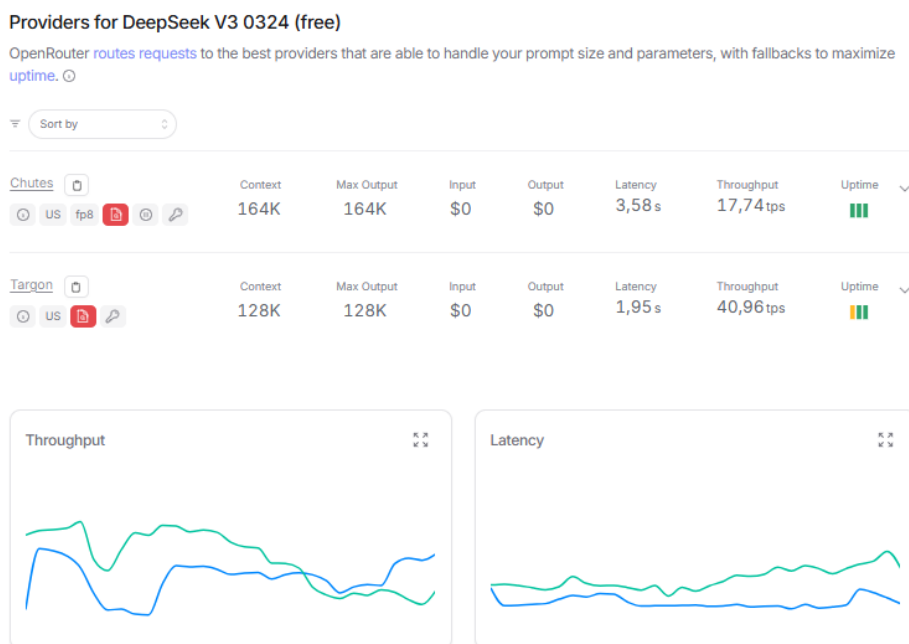


Figura 7 Ejemplo de métricas proporcionadas por OpenRouter para un modelo específico

De entre los modelos gratuitos más utilizados en OpenRouter, destacan varias alternativas con diferente grado de especialización y rendimiento [44]:

- **deepseek-chat-v3-0324:free:** [45] Este modelo es la versión más reciente de DeepSeek disponible gratuitamente en OpenRouter. Utiliza una arquitectura *mixture of experts* y cuenta con 685.000 millones de parámetros. Ofrece acceso gratuito, una ventana de contexto muy amplia, y ha mostrado un rendimiento sobresaliente en tareas de razonamiento, programación y matemáticas, situándose a la altura de modelos comerciales de alta gama. Su integración es sencilla y no requiere hardware especializado, lo que lo hace extremadamente accesible para desarrolladores y proyectos académicos o personales.
- **Deepseek R1:** [46] Es una alternativa gratuita de DeepSeek, con un enfoque más específico en tareas de razonamiento profundo y análisis avanzado. Es valorado por su robustez y precisión, aunque puede ser algo más lento que v3-0324 en algunas tareas.
- **Gemini 2.0 Flash Experimental:** [47] Modelo experimental de Google, orientado a tareas multimodales y contextos extensos. Su uso es frecuente en aplicaciones que requieren tanto generación de texto como manejo de imágenes, y destaca por su capacidad de adaptación a diferentes tipos de consultas.
- **Gemma 3:** [48] Modelo adecuado para tareas generales de procesamiento de lenguaje natural. Es conocido por su facilidad de integración y su buen rendimiento en tareas estándar, siendo una opción versátil y gratuita.
- **Scout 109B:** [49] Variante optimizada de Llama 4, con arquitectura *Mixture of Experts* y gran capacidad de contexto. Es ideal para tareas que requieren manejo de grandes volúmenes de información y despliegues eficientes.

La elección de **deepseek-chat-v3-0324:free** como modelo principal en el proyecto se fundamenta en varios aspectos clave que lo diferencian de otras alternativas gratuitas disponibles en OpenRouter. En primer lugar, destaca por su fiabilidad operativa. A diferencia de otros modelos gratuitos que pueden experimentar caídas frecuentes o interrupciones en el servicio, este modelo ha demostrado una mayor estabilidad y disponibilidad. Esta robustez se ve reforzada por los mecanismos de balanceo y monitorización de OpenRouter, pero la base es un modelo que rara vez deja de estar disponible, minimizando así el riesgo de fallos inesperados en el flujo de trabajo.

Otro factor determinante es su capacidad para comprender *prompts* complejos y específicos del contexto musical. DeepSeek V3-0324 incorpora una arquitectura avanzada y una ventana de contexto ampliada de hasta 163.000 *tokens*, lo que le permite manejar grandes volúmenes de información y mantener la coherencia en conversaciones largas o con múltiples referencias contextuales. Estas características son especialmente útiles cuando se trabaja con descripciones musicales, análisis de archivos MIDI o instrucciones técnicas, ya que el modelo puede procesar y responder de forma lógica, precisa y eficiente, incluso cuando los *prompts* contienen detalles técnicos o referencias cruzadas.

3. Especificaciones y restricciones de diseño

El presente capítulo detalla las especificaciones y restricciones que se han marcado para este proyecto y que han guiado el diseño y desarrollo del sistema. Estas condiciones aseguran que la solución propuesta sea coherente con los objetivos planteados, viable desde el punto de vista técnico y eficiente en su funcionamiento:

- **Interacción por voz y precisión en la interpretación de comandos vocales:** El sistema debe ser capaz de interpretar comandos de voz con alta precisión. Esto es esencial para que el usuario pueda interactuar de forma natural y eficiente, permitiendo controlar las distintas funcionalidades del asistente, sin necesidad de gran intervención manual.
- **Generación precisa de archivos MIDI a partir de tarareos:** Una de las funciones principales es la generación de archivos MIDI a partir de los tarareos del usuario, empleando técnicas de análisis de frecuencia y reconocimiento de tonos. Se busca que la conversión sea lo más fiel posible a la melodía original, minimizando el margen de error para que el resultado sea útil en contextos musicales reales.
- **Integración eficiente con estación de trabajo de audio digital:** El asistente debe integrarse de forma fluida con una DAW, permitiendo la importación automática de pistas MIDI y la aplicación de efectos o instrumentos sin intervención manual. Esta integración agiliza el flujo de trabajo y facilita la producción musical.
- **Ajuste automático del ritmo y tempo:** El sistema debe ser capaz de ajustar automáticamente el ritmo de los tarareos para adaptarse a diferentes tempos, facilitando la sincronización y coherencia rítmica en las composiciones.
- **Sugerencias contextuales basadas en análisis del usuario:** El asistente debe analizar las acciones del usuario y ofrecer recomendaciones personalizadas para mejorar la calidad musical o la eficiencia del proceso creativo. Esta capacidad se basa en técnicas de inteligencia artificial que interpretan el contexto y las necesidades específicas.
- **Interfaz gráfica intuitiva y respuesta textual:** La aplicación debe contar con una interfaz gráfica clara y fácil de usar, a través de la cual el asistente responderá al usuario mediante mensajes de texto, proporcionando información y recomendaciones de manera comprensible, además de realizarle peticiones en determinados momentos.
- **Arquitectura modular, ligera y portable:** El sistema debe estar diseñado de forma modular, con componentes independientes, pero bien cohesionados, lo que facilita futuras actualizaciones y el mantenimiento. Además, la aplicación debe ser ligera, fácil de instalar y funcionar correctamente en una variedad de equipos sin requerir hardware especializado.

- **Funcionamiento estable y continuo:** El asistente debe garantizar un funcionamiento robusto durante sesiones prolongadas, evitando caídas o pérdidas de datos que puedan afectar la experiencia del usuario.
- **Búsqueda de soluciones de código abierto o gratuitas:** Como limitación principal, el desarrollo debe optar por soluciones de código abierto o gratuitas, evitando software de pago para asegurar la accesibilidad y la replicabilidad del proyecto.

4. Descripción de la solución propuesta

En este capítulo se presenta en detalle la solución desarrollada para el asistente inteligente de producción musical. Se inicia con una visión global del sistema, donde se describe el enfoque general del proyecto y su arquitectura principal. A continuación, se exponen las funcionalidades clave que ofrece el asistente, explicando cómo se han implementado y articulado a través de los distintos módulos que componen la solución. De este modo, se podrá comprender tanto el funcionamiento global del sistema como el papel específico de cada uno de sus componentes dentro de la solución propuesta.

4.1. Visión general del sistema

La solución propuesta consiste en un asistente inteligente orientado a la producción musical doméstica, diseñado para facilitar el proceso creativo a usuarios que desean plasmar sus ideas musicales sin necesidad de dominar una estación de trabajo de audio digital compleja. La interacción con el sistema se basa en la interpretación precisa de comandos de voz del usuario, lo que permite una interacción natural e intuitiva, eliminando barreras técnicas y facilitando el acceso a herramientas avanzadas de producción musical.

A través de una interfaz gráfica intuitiva y la interpretación precisa de comandos de voz, el usuario puede grabar tarareos, generar archivos MIDI, manipular instrumentos y efectos, y recibir sugerencias creativas personalizadas gracias a la integración de modelos de IA. El asistente automatiza tareas como la conversión de audio a MIDI, la aplicación de efectos y la gestión de pistas, democratizando el acceso a herramientas avanzadas de producción y permitiendo que músicos de cualquier nivel técnico experimenten y personalicen su proceso creativo.

La elección de una arquitectura modular responde a la necesidad de escalabilidad, facilidad de mantenimiento y flexibilidad para futuras ampliaciones. Este enfoque permite aislar funcionalidades en módulos independientes, facilitando tanto la depuración como la incorporación de nuevas características sin afectar al resto del sistema. Se ha optado por una interacción basada en comandos de voz para maximizar la accesibilidad y naturalidad en el uso, eliminando barreras técnicas y permitiendo que usuarios sin experiencia previa en DAWs complejas puedan aprovechar el asistente desde el primer momento. La integración con Reaper se justifica por su amplia compatibilidad, su soporte para scripting avanzado y su ecosistema de *plugins*, lo que permite automatizar procesos y personalizar el flujo de trabajo musical de manera eficiente. Además, la inclusión de modelos de inteligencia artificial responde a la necesidad de ofrecer sugerencias creativas y personalizadas, así como de automatizar tareas complejas como la conversión de audio a MIDI, aportando un valor añadido que diferencia al sistema frente a soluciones tradicionales.

4.2. Análisis de requisitos

El análisis de requisitos constituye un paso fundamental en el diseño de la solución, ya que permite establecer de forma clara qué funcionalidades debe ofrecer el sistema y en qué condiciones de calidad debe operar. En este apartado se definen, por un lado, los requisitos funcionales, que recogen las capacidades concretas que debe proporcionar el asistente musical, y por otro, los requisitos no funcionales, que establecen criterios de rendimiento, usabilidad, compatibilidad y mantenibilidad para garantizar la correcta operación del sistema en un entorno real.

4.2.1. Requisitos funcionales y descripción completa de funcionalidades

En este apartado se detallan los requisitos funcionales del asistente inteligente y se proporciona una descripción completa de cada funcionalidad, mostrando cómo estas facilitan la creación y edición musical. El sistema permite a usuarios sin experiencia técnica plasmar sus ideas de manera sencilla e intuitiva, automatizando tareas técnicas y apoyando la creatividad musical. Las funcionalidades seleccionadas responden a las principales necesidades detectadas en usuarios que desean producir música sin tener conocimientos avanzados en estaciones de trabajo de audio digital. Se priorizan tareas que suelen ser repetitivas o técnicas y que, al automatizarse, permiten centrar el proceso creativo en la música en sí. Además, se ha buscado cubrir tanto la generación y edición de material musical como la asistencia creativa y la integración fluida con herramientas profesionales como Reaper. La posibilidad de interactuar por voz y recibir recomendaciones personalizadas mediante el modelo de IA refuerza la accesibilidad y la adaptabilidad del asistente a distintos perfiles de usuario.

A continuación, se describen los requisitos funcionales implementados. En cada caso, se indica primero qué capacidad debe ofrecer el sistema y, a continuación, se describe cómo se desarrolla dicha funcionalidad una vez identificado el comando por parte del asistente.

1. Grabar tarareos y abrir MIDI en Reaper

Permite al usuario grabar un tarareo a través del micrófono y, tras su procesamiento y transformación en archivo MIDI, abre automáticamente el archivo MIDI resultante de este tarareo en la DAW Reaper para su edición o reproducción. Se importa el archivo MIDI en la siguiente pista disponible, ajustando el *tempo* en el programa, y asignándole un piano genérico como instrumento, todo esto de manera automática, sin intervención alguna por parte del usuario. Una vez identificado el comando, el programa le pide al usuario que introduzca el nombre que desea ponerle el MIDI y una vez que este lo introduzca se abre la GUI (Graphical user interface) de tarareos, la cual es una pequeña ventana en la cual el usuario puede mediante dos botones decidir cuándo desea iniciar y detener el tarareo, una vez que el usuario pulse el botón detener, comienza toda la automatización anteriormente mencionada.

- **Ejemplo de comando:** “Quiero grabar un tarareo y abrirlo con Reaper.”

2. Grabar tarareos y guardar MIDI

El usuario puede grabar un tarareo y guardar el archivo MIDI generado en una carpeta local creada específicamente para este cometido, facilitando la organización y posterior uso de sus ideas musicales. Una vez identificado el comando, el programa le pide al usuario que introduzca el nombre que desea ponerle el MIDI y una vez que este lo introduzca se abre la GUI de tarareos, una vez que el usuario pulse el botón detener el MIDI se guarda con el nombre propuesto por el usuario anteriormente.

- **Ejemplo de comando:** “Quiero grabar un tarareo y guardarlo en mi carpeta.”

3. Abrir archivo MIDI en Reaper

Permite abrir directamente en Reaper cualquier archivo MIDI existente en la carpeta del proyecto utilizada para esto, simplificando el flujo de trabajo y evitando la necesidad de realizar la importación manual. Se importa el archivo MIDI en la siguiente pista disponible, ajustando el *tempo* en el programa, y asignándole un piano genérico como instrumento todo esto de manera automática, sin intervención alguna por parte del usuario. Una vez identificado el comando, el programa le pide al usuario que introduzca el nombre del MIDI que desea abrir y si se encuentra en la carpeta comenzará toda la automatización anteriormente mencionada.

- **Ejemplo de comando:** “Quiero abrir un MIDI en Reaper.”

4. Cambio de instrumento en una pista MIDI

Ofrece la posibilidad de cambiar el instrumento asignado a la última pista MIDI creada, permitiendo experimentar con diferentes sonidos de forma sencilla. Una vez identificado el comando y el propio instrumento a cambiar dentro del comando, el programa de manera automática abre el efecto DSK Virtuoso y cambia el instrumento al solicitado en la última pista seleccionada.

- **Ejemplo de comando:** “Cambia el instrumento de mi pista a violín.”

5. Aplicación de efectos para mejorar la melodía

Permite añadir efectos como ecualización, compresión, reverberación u otros procesamientos sonoros a una pista o melodía, mejorando la calidad y creatividad de la composición. Todo esto se hace de manera automática y utilizando el modelo de IA, el cual ajusta los parámetros de los distintos efectos en función de las características de la creación musical del usuario. Una vez identificado el comando el programa extrae las características clave sobre el MIDI al que se le desean añadir los efectos. Mediante un *prompt* preparado previamente al cual se le añaden estas características se le pide a la IA que devuelva en formato JSON (JavaScript Object Notation) los valores cambiantes de los parámetros de los distintos efectos que se le aplicarán al MIDI, en función de las características del MIDI. Tras esto se procesan esos datos y se pasan

a Reaper, donde se le aplicarán los distintos efectos y se le añaden estos valores propuestos por la IA y previamente procesados para la adaptación a cada efecto.

- **Ejemplo de comando:** “Mejora mi melodía con efectos.”

6. Consulta de mejora sobre la creación

El asistente puede analizar una creación musical y sugerir posibles mejoras, apoyando al usuario en el proceso creativo. Una vez identificado el comando, se analizan las distintas características de la creación musical del usuario y se le añaden al *prompt* creado para esta petición o se consulta el *prompt* generado anteriormente y con contexto actualizado en caso de que ya se hayan hecho otras acciones con el asistente. A continuación, se introducen en el modelo de IA del sistema, el cual responde sobre cómo mejorar la creación musical del usuario, mostrándole el consejo por pantalla.

- **Ejemplo de comando:** “¿Cómo puedo mejorar mi creación musical?”

7. Preguntas específicas

El usuario puede realizar preguntas al asistente, tanto por texto como por voz, relacionadas con la producción musical, el uso de la DAW o aspectos técnicos y creativos. Una vez identificado el comando, se le pide al usuario que realice la pregunta específica ya sea escrita o de forma hablada, a preferencia del usuario. Tras esto, se analizan las distintas características de la creación musical del usuario y se le añaden al *prompt* creado para esta petición junto con la pregunta específica del usuario o se consulta el *prompt* generado anteriormente con contexto actualizado en caso de que ya se hayan hecho otras acciones con el asistente al que se le añade esta pregunta. A continuación, se introduce en el modelo de IA del sistema, la cual responde a la pregunta del usuario, mostrándole la respuesta por pantalla.

- **Ejemplo de comando:** “Quiero hacerte una pregunta.”

8. Generación automática de línea de bajo

A partir de una melodía principal, el asistente puede generar automáticamente una línea de bajo que complemente la composición, agilizando el desarrollo de arreglos musicales. Una vez identificado el comando el programa crea una línea de bajo sobre la melodía del último MIDI seleccionado, guarda el bajo en la carpeta de MIDIs y lo importa automáticamente en una nueva pista en Reaper, añadiéndole el efecto “DSK Virtuoso” con el instrumento por defecto de contrabajo, todo sin intervención manual del usuario.

- **Ejemplo de comando:** “Genera una línea de bajo basada en esta melodía.”

9. Añadir sintetizador

Permite añadir un sintetizador a la melodía actual utilizando el modelo de IA y en función de las características de la propia creación musical hasta el momento, ampliando las posibilidades sonoras y experimentales del usuario. Una vez identificado el comando el programa extrae las

características clave sobre el MIDI sobre el que se desea crear el sintetizador. Mediante un *prompt* preparado previamente al cual se le añaden estas características se le pide al modelo de IA que devuelva en formato JSON los valores cambiantes de los parámetros de “ReaSynth”, que es el efecto que se utiliza para crear el sintetizador, en función de las características del MIDI. Tras esto se procesan esos datos y se pasan a Reaper, donde se importa en una nueva pista el MIDI sobre el que se realiza el sintetizador, se le aplica el efecto “ReaSynth” y se le añaden estos valores propuestos por el modelo de IA previamente procesados para la adaptación al efecto.

- **Ejemplo de comando:** “Añade un sintetizador a la melodía.”

Estas funcionalidades deben poder ser activadas mediante comandos de voz o mediante la interfaz gráfica, adaptándose a las preferencias y necesidades de cada usuario. El sistema está diseñado para interpretar de forma flexible las peticiones y ejecutar las acciones correspondientes de manera automática, facilitando así el flujo creativo y la interacción con la DAW Reaper.

A continuación, en la figura 8 se muestra un diagrama funcional que resume la estructura de funcionalidades del sistema y el flujo de funcionamiento de cada una:

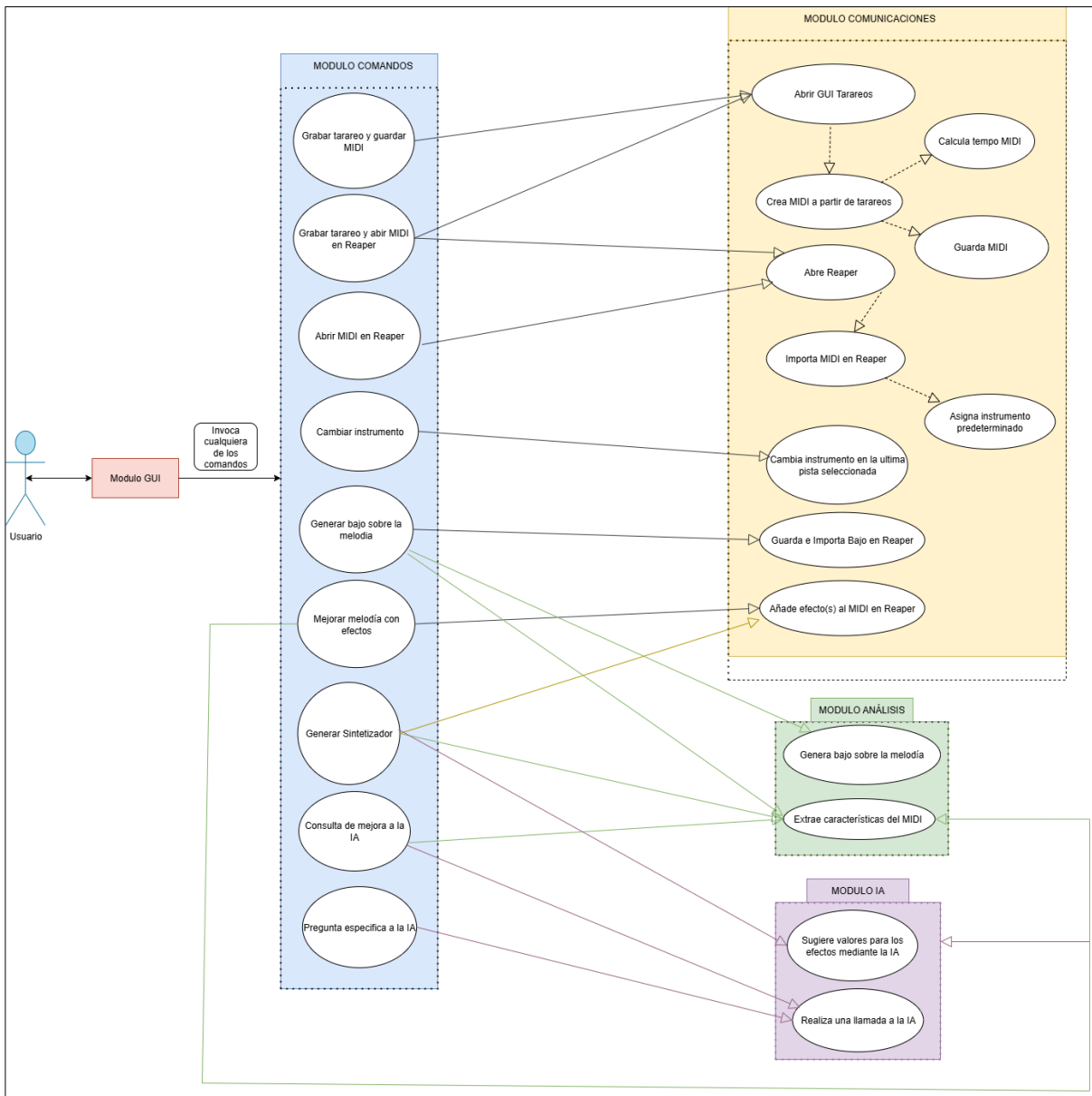


Figura 8. Diagrama funcional del sistema

4.2.2. Requisitos no funcionales

Además de las funcionalidades descritas en el apartado 4.2.1, el sistema debe cumplir una serie de requisitos no funcionales que garantizan su calidad, fiabilidad y facilidad de uso. Estos requisitos abarcan aspectos esenciales para asegurar que la solución propuesta sea práctica y sostenible en el tiempo:

- **Usabilidad:** La aplicación debe ofrecer una interfaz gráfica sencilla e intuitiva, que permita a usuarios sin experiencia técnica interactuar cómodamente. El sistema debe aceptar tanto comandos de voz como de texto, proporcionando respuestas claras y comprensibles.

- **Compatibilidad:** El sistema debe integrarse de forma fluida con Reaper en entornos Windows, y trabajar con formatos estándar como MIDI para garantizar la interoperabilidad con otras herramientas musicales.
- **Rendimiento:** El procesamiento de los comandos debe realizarse con baja latencia, de manera que la respuesta del asistente sea prácticamente inmediata. Las tareas de generación de MIDI y aplicación de efectos deben completarse en tiempos adecuados para no interrumpir el flujo creativo del usuario.
- **Escalabilidad:** La arquitectura del sistema debe permitir la incorporación de nuevas funcionalidades o la integración con otros modelos de inteligencia artificial sin requerir modificaciones drásticas.
- **Robustez:** El sistema debe ser capaz de manejar errores o entradas inesperadas sin detener su funcionamiento. Ante fallos, debe ofrecer mensajes claros al usuario y asegurar la recuperación de la sesión de trabajo.
- **Mantenibilidad:** El código debe estar organizado en componentes bien definidos, con documentación suficiente que facilite la comprensión, corrección de errores y futuras ampliaciones.

4.3. Arquitectura general

El asistente inteligente propuesto se organiza mediante una arquitectura modular, en la que cada componente desempeña una función específica dentro del flujo de trabajo musical. La Figura 9 muestra el diagrama general del sistema, en el que se representan los principales módulos y el flujo de funcionalidades entre ellos.

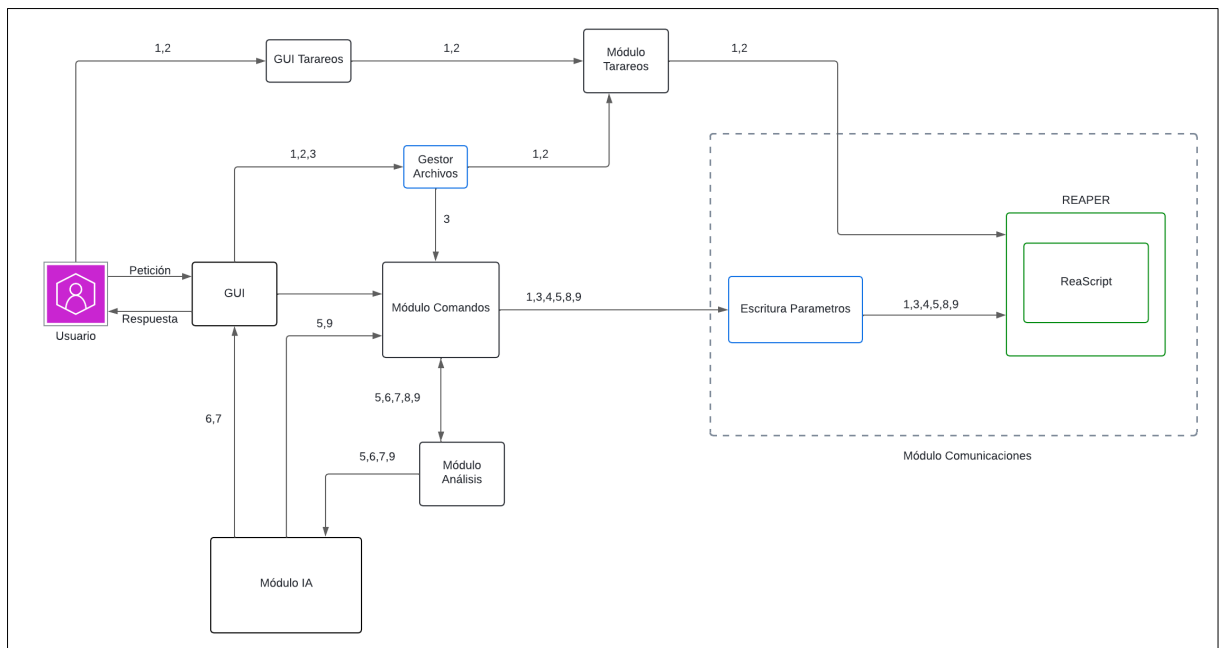


Figura 9. Arquitectura general del sistema

El usuario interactúa con el sistema a través de una GUI, enviando peticiones mediante comandos de voz. La GUI recibe estas solicitudes y las transmite al Módulo de Comandos, que actúa como el núcleo central encargado de gestionar la lógica del asistente. Según la naturaleza de la petición, este módulo coordina la interacción con los distintos módulos especializados:

- El Módulo Tarareos y la GUI de los tarareos gestionan la captura y procesamiento de tarareos para su conversión en archivos MIDI.
- El Gestor de Archivos se encarga de la organización, almacenamiento y recuperación de los archivos generados o utilizados.
- El Módulo IA procesa las solicitudes que requieren análisis avanzado, generación de sugerencias o respuestas a preguntas a través de la IA.
- El Módulo Análisis realiza la extracción de características musicales y el análisis estructural de los archivos MIDI, además de generar el bajo en base a las características de la melodía cuando sea necesario.
- El Módulo de Comunicaciones, mediante ReaScript y OSC, ejecuta las acciones solicitadas en la DAW Reaper y se encarga de la comunicación completa entre asistente y DAW.
- El componente Escritura de Parámetros se utiliza como intermediario para pasar datos relevantes desde el asistente al DAW como parámetros, rutas de los MIDI, etc.

El flujo de información entre los módulos está representado en la figura X mediante flechas numeradas, que corresponden a las funcionalidades principales descritas en el apartado 4.2 de este documento, es decir los números que aparecen en las flechas corresponden con los números de cada funcionalidad disponible en el asistente. Esto permite identificar de manera visual qué módulos intervienen en cada proceso y cómo se interrelacionan para cumplir con las solicitudes del usuario.

Esta arquitectura modular facilita la escalabilidad y el mantenimiento del sistema, permitiendo la incorporación de nuevas funcionalidades de forma sencilla y eficiente. Además, la comunicación automática con Reaper garantiza que las acciones solicitadas por el usuario se ejecuten de manera eficiente, integrando la inteligencia artificial y la automatización dentro del entorno habitual de producción musical.

4.4. Desglose técnico por módulos

A continuación, se analiza en detalle el funcionamiento interno de cada uno de los módulos que integran el sistema, abordando su desarrollo, funcionalidad, tecnologías empleadas y cómo está implementado cada componente. Este análisis permite comprender cómo cada parte contribuye a la operatividad y eficiencia del asistente, completando así la visión técnica

presentada en apartados anteriores y facilitando una comprensión profunda del flujo de información y control dentro del sistema.

4.4.1. Módulo GUI

El módulo GUI constituye el punto de entrada y el principal canal de interacción entre el usuario y el asistente inteligente para producción musical. Su objetivo es proporcionar una experiencia intuitiva, clara y eficiente, permitiendo que el usuario comunique sus necesidades al sistema tanto por voz como por texto, y reciba información relevante, notificaciones y solicitudes de manera visualmente atractiva.

La interfaz está diseñada para cubrir los principales flujos de trabajo del usuario en el contexto de la producción musical asistida. La pantalla principal muestra mensajes informativos, preguntas del sistema, errores y respuestas generadas por los diferentes módulos o componentes del sistema, diferenciando cada tipo de mensaje mediante colores y estilos personalizados para mejorar la comprensión y la experiencia de usuario.

El usuario dispone de un botón específico para grabar comandos por voz, lo que permite una interacción natural y rápida, especialmente útil en entornos creativos donde la inmediatez es clave. Además, la GUI incluye una línea de entrada de texto con su correspondiente botón de envío, pensada para introducir nombres de archivos, preguntas escritas o cualquier comando textual que requiera precisión. Esta dualidad de entrada garantiza que el asistente sea accesible y versátil, adaptándose a diferentes preferencias y situaciones de uso.

La funcionalidad de la GUI no se limita a la recogida de entradas: también actúa como canal de retroalimentación, informando al usuario sobre el estado actual del sistema, confirmando acciones realizadas, mostrando resultados de análisis musicales, sugerencias del modelo de IA o solicitando información adicional cuando es necesario. Todo esto contribuye a que el usuario mantenga el control y la comprensión del flujo de trabajo en todo momento.

En la figura 10 se muestra el prototipo de la GUI diseñada para el sistema:

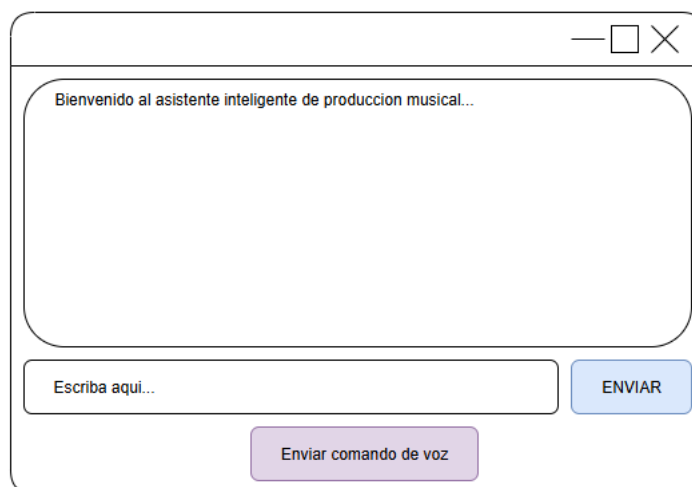


Figura 10. Prototipo GUI del sistema

La implementación del módulo GUI sigue una arquitectura modular y desacoplada, basada en la separación de responsabilidades a través de dos archivos principales: uno para la vista llamado **GUIView.py** y otro para el controlador llamado **GUIController.py**. Este enfoque, inspirado en el patrón Modelo-Vista-Controlador, facilita la mantenibilidad, escalabilidad y claridad del sistema, permitiendo que la lógica de presentación y la lógica de control evolucionen de manera independiente. En la figura 11 se muestra la comunicación entre los dos archivos principales del módulo GUI y los módulos funcionales del sistema:

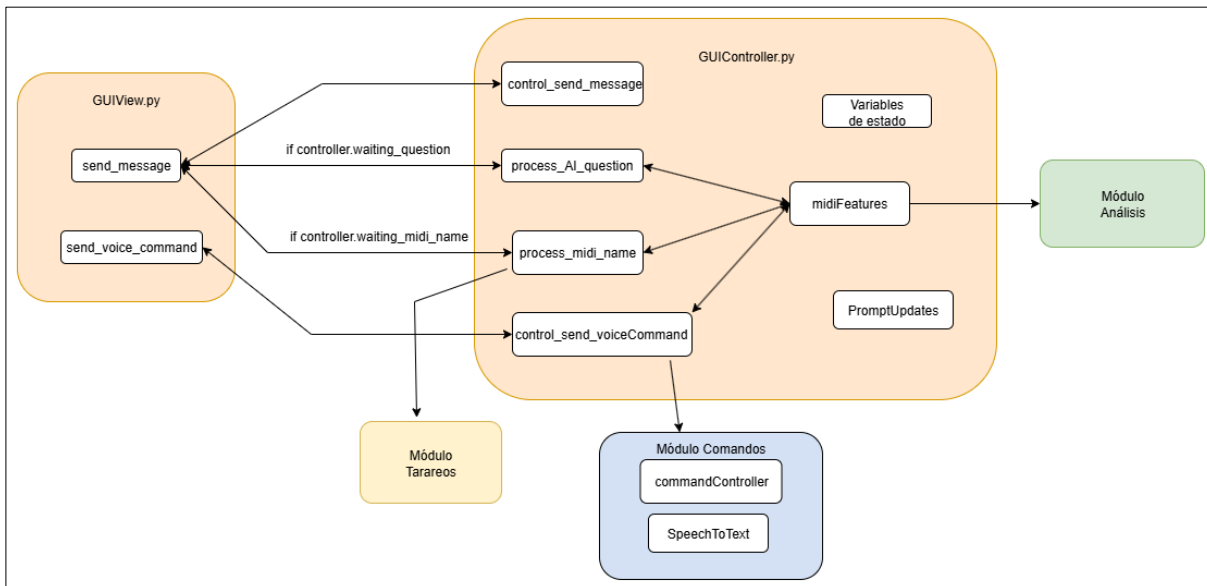


Figura 11. Comunicación entre archivos del módulo GUI y otros módulos funcionales

El archivo de **GUIView.py**, desarrollado principalmente con la biblioteca PyQt5, es responsable de construir y gestionar la interfaz gráfica. Este archivo define la ventana principal, organiza los elementos visuales mediante *layouts* verticales y horizontales, y aplica estilos personalizados para lograr una apariencia moderna y profesional. Los componentes clave de la interfaz son:

- **Pantalla de mensajes:** Creada con el widget QTextEdit [50] de la biblioteca PyQt5. Muestra mensajes del sistema, diferenciados por tipo de mensaje mediante colores personalizados. Los tipos de mensaje son de información, de petición, de error, de la IA, de usuario y de escucha. Es de solo lectura para evitar modificaciones accidentales y utiliza una fuente clara para facilitar la lectura.
- **Línea de entrada de texto y botón de envío:** Creados con los widgets QLineEdit [51] y QPushButton [52] de la biblioteca PyQt5. Permiten al usuario escribir nombres de archivos, preguntas y demás interacciones necesarias mediante texto escrito. El botón de envío está estilizado para destacar su función y mejorar la experiencia de uso.
- **Botón de grabación de voz:** Creado con el widget QPushButton de la biblioteca PyQt5. Facilita la captura de comandos hablados. Al pulsarlo, la interfaz informa al usuario de que el sistema está escuchando y gestiona la transición hacia el procesamiento de

audio, escuchando los comandos de voz y demás interacciones entre usuario y sistema de forma hablada.

La vista también se encarga de gestionar la actualización dinámica de la interfaz, desplazando automáticamente la pantalla de mensajes al final tras cada nueva entrada, y mostrando mensajes contextuales según el flujo de la interacción.

El archivo **GUIController.py** constituye el núcleo lógico que gestiona la interacción entre la interfaz gráfica y el resto del sistema, desempeñando un papel fundamental tanto en la lógica de flujo conversacional como en la coordinación con los módulos funcionales del asistente. Este controlador mantiene múltiples variables de estado que permiten gestionar contextos conversacionales complejos, como la espera de un nombre de archivo MIDI, la formulación de preguntas a la IA, la gestión de comandos repetidos, o el seguimiento de características avanzadas de los archivos MIDI y su evolución a lo largo de la sesión.

Cuando el usuario introduce texto a través de la GUI, el controlador evalúa el contexto actual y decide cómo procesar la entrada. El método **control_sendMessage** es clave en este proceso, ya que interpreta si el sistema está esperando un nombre de archivo MIDI, una pregunta para la IA o simplemente un comando textual, y genera el mensaje de retorno adecuado, actualizando el estado interno del controlador para mantener la coherencia del flujo de trabajo.

Para los comandos de voz, el método **control_send_voiceCommand** integra el reconocimiento de voz y, gracias a la conexión con el módulo de comandos, realiza un análisis detallado de la instrucción recibida. Identifica el tipo de acción solicitada y desencadena los flujos de interacción necesarios, que pueden incluir la solicitud de datos adicionales al usuario o la ejecución de acciones sobre el DAW a través de los módulos de comunicación.

Un aspecto diferencial del controlador es la gestión dinámica del *prompt* contextual que alimenta al módulo de IA. Este *prompt* se va enriqueciendo con cada acción relevante que realiza el usuario: creación y carga de archivos MIDI, modificaciones, adición de efectos, cambios de tempo, instrumentos utilizados, análisis realizados, etc. Así, el *prompt* refleja en todo momento el estado real y actualizado del proyecto musical, permitiendo que el modelo de IA genere respuestas y sugerencias adaptadas a la evolución concreta del trabajo del usuario, lo que resulta fundamental para mantener la coherencia y la utilidad de la interacción.

El controlador también coordina el lanzamiento y la gestión de interfaces auxiliares, como la ventana de tarareos, permitiendo que el usuario cree melodías mediante esta vía y gestionando la integración de los datos musicales generados en el flujo de trabajo general. Además, se encarga de actualizar los mensajes de la GUI con información relevante sobre el estado del sistema, errores o resultados de las operaciones realizadas, asegurando que el usuario reciba retroalimentación clara y contextualizada en cada momento.

El controlador automatiza la gestión de archivos MIDI, incluyendo su carga, creación y análisis cada vez que el usuario realiza una acción relevante. Cuando se carga o modifica un archivo

MIDI, el controlador solicita al módulo de análisis que extraiga información musical detallada, como el tempo, la estructura de las pistas, los instrumentos asignados, los efectos aplicados, la tonalidad, los acordes y las notas presentes. Todos estos datos se incorporan automáticamente en el *prompt* contextual, asegurando que la IA siempre disponga de un contexto actualizado y preciso sobre el estado del proyecto. Si el usuario modifica el tempo, añade nuevas pistas o realiza cualquier cambio que afecte a la estructura musical, el controlador vuelve a activar el análisis y actualiza el *prompt* con la información más reciente, manteniendo la coherencia y sincronización del flujo de trabajo musical. Este proceso continuo de extracción y actualización de características musicales permite que las recomendaciones y análisis generados por el modelo de IA sean siempre precisos y adaptados a la evolución real de la creación del usuario.

La colaboración entre la vista y el controlador garantiza un flujo de trabajo claro y eficiente. La vista actúa como capturador y presentador de información, mientras que desde el controlador se interpreta, decide y coordina las acciones del sistema. Este diseño modular permite que futuras ampliaciones, como la incorporación de nuevos tipos de comandos o mejoras en la presentación, se realicen de manera sencilla y sin afectar la arquitectura global.

4.4.2. Módulo Comandos

El módulo de comandos es el encargado de interpretar y procesar las instrucciones del usuario, y transformarlas en acciones concretas dentro del sistema. Para ello, está estructurado en tres archivos principales que colaboran entre sí: `speechToText.py`, `witAI.py` y `commandController.py`. Cada uno desempeña un papel específico en la cadena de procesamiento de los comandos, permitiendo una gestión eficiente y flexible de la interacción con el usuario.

El archivo **`speechToText.py`** se encarga de capturar el audio del micrófono y convertirlo a texto mediante el uso de la biblioteca Vosk. Este componente utiliza un modelo de reconocimiento de voz entrenado para el idioma español, que procesa el audio en tiempo real y transcribe las palabras pronunciadas por el usuario. El proceso comienza con la inicialización del modelo Vosk y la configuración de un flujo de audio que captura los datos del micrófono a través de una cola. Cuando se detecta una frase completa, el texto reconocido se extrae y se devuelve como resultado, permitiendo al sistema trabajar con la instrucción del usuario de forma textual. Este archivo es fundamental para habilitar la interacción por voz, ofreciendo una experiencia natural y fluida en entornos creativos donde la inmediatez es clave.

El archivo **`witAI.py`** es el encargado de analizar el texto recibido y extraer tanto la intención de este, es decir, el tipo de acción solicitada, como las entidades asociadas, es decir, los parámetros relevantes.

- **Intención del usuario (*Intent*):** Acción que el usuario desea realizar al enviar el mensaje hablado.

- **Entidades (*Entities*):** Partes específicas del mensaje del usuario que son relevantes para alguna acción, por ejemplo, el nombre del instrumento en la acción de cambiar de instrumento.

En la figura 12 se muestra el flujo de datos entre vosk y wit.ai cuando el usuario introduce un comando mediante la voz.

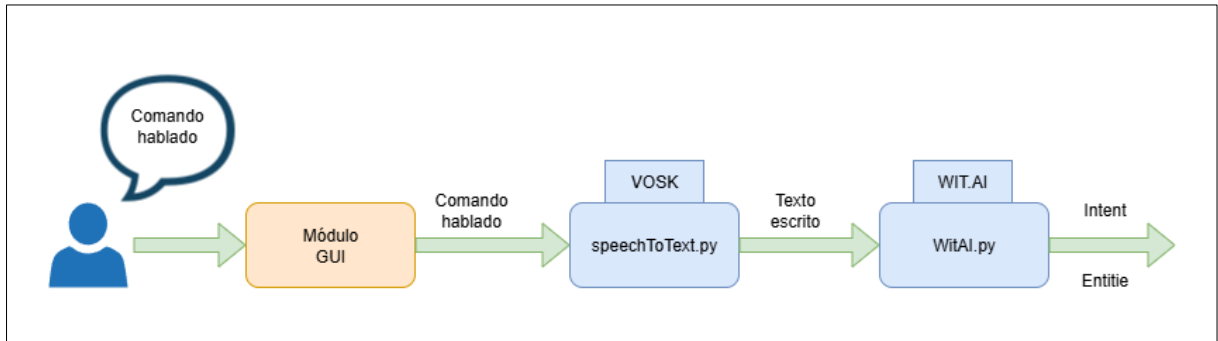


Figura 12. Flujo de datos entre vosk y wit.ai

Para obtener la intención y las posibles entidades, se utiliza la API de Wit.ai, una plataforma de inteligencia artificial que permite entrenar modelos de lenguaje natural desde su interfaz web.

El entrenamiento del modelo de Wit.ai se realiza desde su interfaz web, donde el desarrollador define *intents* y *entities* personalizadas para cada tipo de comando posible en el sistema. Para ello, se introducen múltiples ejemplos de frases que el usuario podría decir, y se anotan manualmente las partes del texto que corresponden a cada entidad, por ejemplo, el nombre de un instrumento. Este proceso de anotación permite que Wit.ai aprenda a identificar tanto la intención como los parámetros relevantes en cualquier entrada de usuario.

A medida que se añaden más ejemplos y se refinan las anotaciones, el modelo mejora su capacidad de reconocimiento y extracción de información. Además, Wit.ai permite validar y corregir las entidades identificadas, asegurando que el sistema interprete correctamente los comandos complejos y variados. El resultado es un modelo robusto y adaptable, capaz de manejar una gran variedad de instrucciones y parámetros en el contexto de la producción musical.

Para entrenar este modelo se ha definido una aplicación de Wit.ai y se han definido nueve *intents*, uno por cada funcionalidad del asistente y un *entity*, para definir el instrumento en el comando en el que sea necesario, por lo tanto, va asociado al *intent* creado para ese comando. Tras esto se han entrenado los distintos *intents* y el *entity* con distintos tipos de frase para referirse a cada tipo de comando y cada tipo de instrumento. La interfaz web muestra como con cada frase que se añade durante el entrenamiento el porcentaje de acierto va subiendo para cada *intent* y cada *entity*. El entrenamiento se realiza hasta tener un porcentaje de acierto por encima del 90% para cada uno de ellos. En la figura 13 se muestra un ejemplo de parte del entrenamiento desde la interfaz web de wit.ai, en el cual tenemos una frase con la que

entrenar el modelo, el *intent* detectado, el *entity* detectado con su valor correspondiente y el porcentaje de acierto de estos.

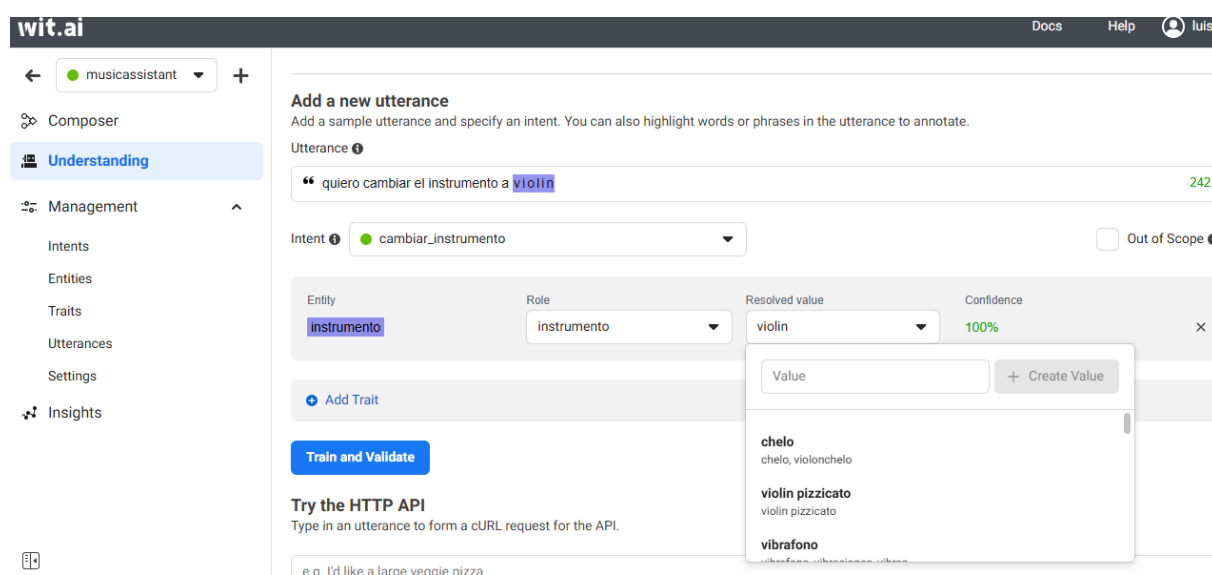


Figura 13. Ejemplo de entrenamiento del modelo de interpretación de comandos

El archivo **commandController.py** es el núcleo que orquesta el procesamiento de los comandos y coordina la ejecución de las acciones en el sistema. El presente módulo recibe el texto reconocido por `speechToText.py` y lo envía a `witAl.py` para su análisis. Una vez se obtienen los *intents* y *entities*, `commandController.py` decide qué acción ejecutar y con qué parámetros, adaptándose al contexto actual del sistema y gestionando los estados internos necesarios. La acción a ejecutar depende del *intent* extraído del comando introducido por el usuario, es decir, hay un *intent* para cada funcionalidad y una vez que este se extrae, se comienza a realizar la funcionalidad específica.

La colaboración entre este archivo y el **GUIController.py** del módulo GUI es fundamental para la gestión de los flujos de interacción con el usuario. Mientras que **GUIController.py** se encarga de captar los eventos del usuario y de mantener actualizado el estado de la interfaz, **commandController.py** es el responsable de interpretar la intención y los parámetros del comando, y de determinar qué funcionalidad activar en función de la información recibida. Esta combinación permite que el sistema sea capaz de manejar contextos conversacionales complejos, como la espera de información adicional o la gestión de errores, manteniendo siempre una coherencia entre lo que el usuario ve y lo que el sistema está haciendo. En la figura 14 se muestra como **commandController.py** interactúa con otros archivos del módulo comandos y cómo sirve de enlace con otros módulos funcionales:

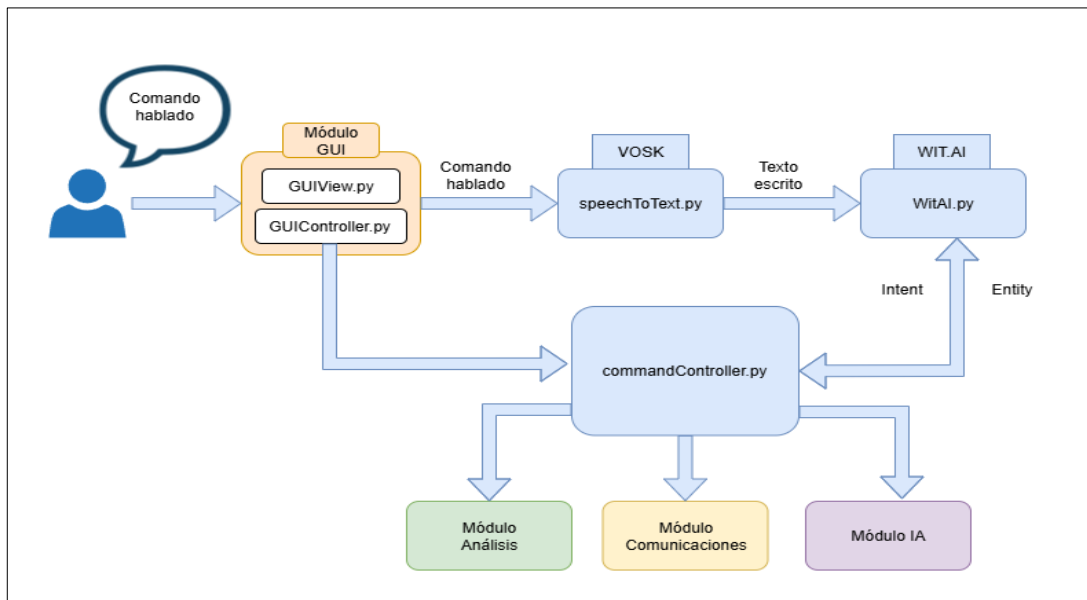


Figura 14. Arquitectura módulo comandos

Dentro de este archivo, el código está estructurado para manejar los diferentes tipos de funcionalidades del asistente. Para cada tipo de comando, el módulo verifica los parámetros recibidos, y realiza las acciones específicas para cada tipo de funcionalidad. Para ello, **commandController.py** integra la lógica para interactuar con otros módulos del sistema:

- **Módulo de comunicaciones:** Para gestionar la comunicación con el DAW, enviar instrucciones al mismo y gestionar la ejecución de acciones sobre el proyecto musical, todo mediante OSC. También se utiliza para el intercambio de parámetros entre el asistente y el DAW, utilizando el submódulo **EscrituraParametros**.
- **Módulo de análisis:** Se encarga de generar líneas de bajo automáticas para archivos MIDI. El módulo de comandos delega la tarea de creación de bajos a este módulo, que analiza el archivo y genera una pista de bajo derivada.
- **Módulo de IA:** Procesa mejoras de efectos y genera recomendaciones musicales basadas en el estado del proyecto. El módulo de comandos envía el *prompt* contextual al módulo de IA, que devuelve sugerencias o ejecuta mejoras automáticas de efectos sobre las pistas seleccionadas. Los parámetros resultantes se almacenan en **EscrituraParametros** para su posterior aplicación en el DAW.

La gestión de parámetros y el estado del sistema es un aspecto clave del funcionamiento de **commandController.py**. Para ello, se utiliza el Submódulo **EscrituraParametros**, que actúa como un mecanismo centralizado para almacenar y recuperar información crítica del sistema, como el tipo y número de instrumento asignado a una pista, el tempo, las rutas de archivos MIDI, o los parámetros de efectos FX generados por la IA y aplicados en *Reaper*. Este Submódulo permite registrar de forma estructurada los parámetros relevantes para cada acción, facilitando su consulta posterior y asegurando la coherencia del estado global del proyecto.

Por ejemplo, dentro del archivo **commandController.py**, cuando el usuario elige un instrumento para una pista, el sistema utiliza un diccionario interno para mapear el nombre del instrumento a su correspondiente ID en Reaper, ya que el efecto que se utiliza en Reaper para añadir un instrumento asigna un ID distinto para cada instrumento de forma interna y la manera de invocar a cada instrumento es pasándole su ID. El submódulo **EscrituraParametros** almacena esta información, permitiendo que el sistema recuerde la configuración de cada pista y pueda gestionar cambios dinámicos de instrumentos de forma sencilla. Del mismo modo, cuando el modelo de IA genera recomendaciones para aplicar efectos FX en Reaper, los parámetros resultantes se almacenan en **EscrituraParametros**, de modo que puedan ser recuperados y aplicados automáticamente mediante el módulo de comunicaciones.

La interacción con los demás módulos se realiza a través de llamadas a funciones específicas, pasando los parámetros necesarios y recibiendo los resultados para mostrarlos al usuario a través de la GUI. El código de **commandController.py** está diseñado para ser modular y extensible, facilitando la incorporación de nuevas funcionalidades y la adaptación a diferentes flujos de trabajo.

4.4.3. Módulo Tarareos

El módulo Tarareos permite transformar melodías cantadas o tarareadas por el usuario en archivos MIDI listos para ser utilizados en el entorno de producción musical. Este módulo está compuesto por dos clases principales, ambas implementadas en el archivo **hum.py**: la clase **humGUI**, que gestiona la interfaz gráfica secundaria para la grabación, y la clase **humToMIDI**, encargada del procesamiento del audio y generación del archivo MIDI.

La clase **humGUI** ofrece una interfaz gráfica sencilla e intuitiva, diseñada específicamente para la grabación de tarareos. Esta ventana secundaria presenta dos botones principales: uno para iniciar la grabación y otro para detenerla, permitiendo al usuario controlar el proceso de forma directa y sin complicaciones. El diseño visual mantiene la coherencia estética con el resto de la aplicación, y la integración de señales utilizando **pyqtSignal** [53] facilita la comunicación con la ventana principal, notificando cuando el proceso de grabación ha finalizado. Gracias a esta interfaz, el usuario puede grabar su melodía de manera natural, sin necesidad de conocimientos técnicos avanzados.

La funcionalidad principal se basa en la clase **humToMIDI**, que gestiona todo el flujo desde la grabación hasta la creación del archivo MIDI. En la figura 15 se muestra el flujo de datos que sigue esta clase para conseguir transformar los tarareos del usuario en un archivo MIDI importable en Reaper:

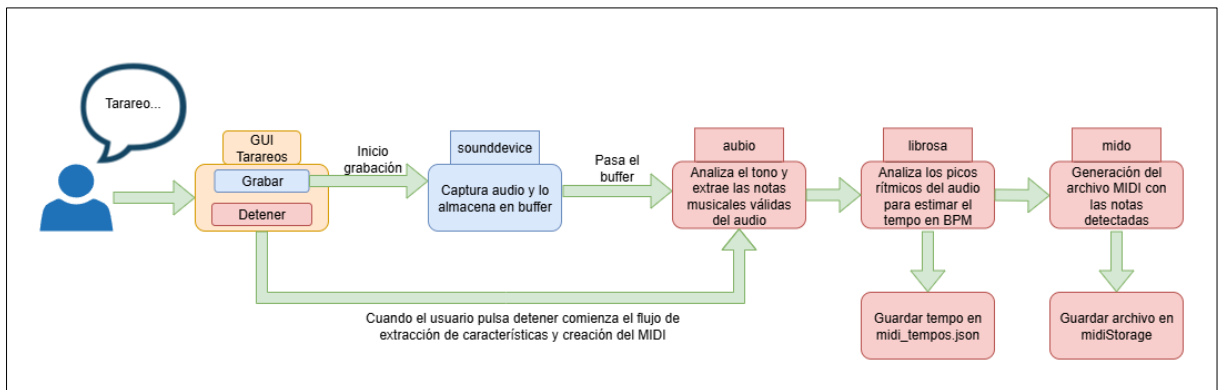


Figura 15. Flujo de datos en la creación del MIDI a partir de tarareos del usuario

El proceso comienza cuando el usuario inicia la grabación a través de la interfaz gráfica secundaria a través de la clase **humGUI**. El módulo utiliza la biblioteca **sounddevice** para capturar el audio del micrófono en tiempo real, almacenando los datos de la señal en un *buffer* mientras la grabación está activa. Una vez el usuario detiene la grabación, el audio capturado se procesa para extraer tanto el tempo como las notas musicales.

La extracción de notas se realiza mediante la librería **aubio**, que analiza el tono de la señal de audio y lo convierte en notas MIDI. El algoritmo recorre el audio en ventanas y detecta los cambios de tono, identificando así el inicio y la duración de cada nota. Solo se consideran válidas aquellas notas cuya duración supera un umbral mínimo, lo que ayuda a filtrar ruidos o sonidos accidentales. Además, se calcula la velocidad de cada nota en función de la intensidad del audio, permitiendo una interpretación más expresiva en el archivo MIDI resultante.

Para la detección del tempo, el módulo emplea la librería **librosa**, que analiza la señal de audio para estimar el tempo en BPM (Beats per minute) a partir de la fuerza de los ataques y los patrones rítmicos presentes en el tarareo. Esta información es esencial para que el archivo MIDI generado esté sincronizado y sea musicalmente coherente con el resto del proyecto.

Una vez extraídas las notas, duraciones, velocidades y tempo, el método **create_midi** genera el archivo MIDI utilizando la librería **mido**. Cada nota detectada se traduce en eventos *note_on* y *note_off* con la duración y la velocidad adecuadas. El archivo resultante se almacena en una carpeta específica del proyecto llamada **midiStorage**, asegurando una organización clara de los recursos generados. Además, el tempo detectado se guarda en un archivo JSON asociado llamado **midi_tempos.json**, lo que permite reutilizar esta información en otros módulos del sistema. En la figura 16 se muestra una captura del archivo **midi_tempos.json** tras la creación de varios MIDIs a través de los tarareos del usuario, se aprecia además las características de este archivo el cual almacena la información en formato JSON con la ruta del MIDI como clave y el valor de su tempo como valor:

```
{ midi_tiempos.json > ...
1
2 "C:\\Users\\luisb\\OneDrive\\Escritorio\\TFG\\Programa\\midiStorage\\default.mid": 151.99908088235293,
3 "C:\\Users\\luisb\\OneDrive\\Escritorio\\TFG\\Programa\\midiStorage\\f.mid": 60.799632352941174,
4 "C:\\Users\\luisb\\OneDrive\\Escritorio\\TFG\\Programa\\midiStorage\\pa.mid": 139.6748310810811,
5 "C:\\Users\\luisb\\OneDrive\\Escritorio\\TFG\\Programa\\midiStorage\\jsh.mid": 105.46875,
6 "C:\\Users\\luisb\\OneDrive\\Escritorio\\TFG\\Programa\\midiStorage\\invento.mid": 246.09375,
7 "C:\\Users\\luisb\\OneDrive\\Escritorio\\TFG\\Programa\\midiStorage\\outputMejorado.mid": 75.0,
8 "C:\\Users\\luisb\\OneDrive\\Escritorio\\TFG\\Programa\\midiStorage\\titanic.mid": 112.34714673913044,
9 "C:\\Users\\luisb\\OneDrive\\Escritorio\\TFG\\Programa\\midiStorage\\titanicp.mid": 120.18531976744185,
10 "C:\\Users\\luisb\\OneDrive\\Escritorio\\TFG\\Programa\\midiStorage\\l.mid": 117.45383522727273,
11 "C:\\Users\\luisb\\OneDrive\\Escritorio\\TFG\\Programa\\midiStorage\\midiPrueba.mid": 135.99917763157896,
12 "C:\\Users\\luisb\\OneDrive\\Escritorio\\TFG\\Programa\\midiStorage\\j.mid": 105.46875,
13 "C:\\Users\\luisb\\OneDrive\\Escritorio\\TFG\\Programa\\midiStorage\\njd.mid": 97.50884433962264
14 }
```

Figura 16. Ejemplo de archivo midi_tiempos.json

Un aspecto clave del módulo es su integración con el flujo de trabajo general del asistente. Tras la conversión de la melodía tarareada en un archivo MIDI, el módulo Tarareos coordina automáticamente el siguiente paso: invoca directamente la función **getCommandAction** del módulo Comandos, pasándole como los parámetros necesarios. Esta coordinación permite que la melodía generada por el usuario pueda ser utilizada inmediatamente en el DAW, sin intervención manual adicional. En la figura 17 se muestra esta integración con los módulos funcionales necesarios para llevar a cabo esta automatización:

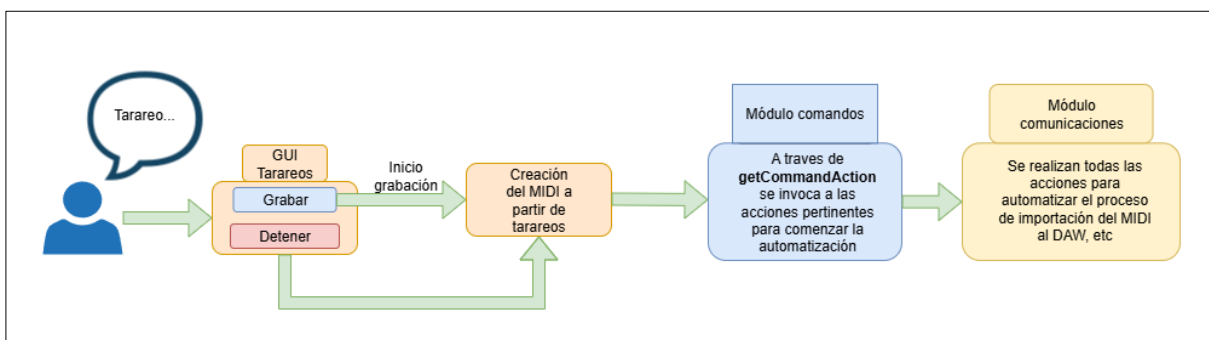


Figura 17. Integración de módulo tarareos con otros módulos

Gracias a esta integración, el sistema puede realizar de forma automatizada todas las tareas necesarias para incorporar la melodía al proyecto musical: importa el archivo MIDI en Reaper, ajusta el tempo del proyecto al detectado en el tarareo y asigna el instrumento por defecto a la pista, en este caso el piano, todo ello siguiendo la configuración y los parámetros definidos previamente por el usuario o por el sistema. De este modo, se asegura una experiencia fluida y eficiente, potenciando la creatividad vocal del usuario y facilitando la integración de nuevas ideas musicales en el entorno de producción.

4.4.4. Módulo Comunicaciones

El módulo de comunicaciones es el núcleo técnico que permite la integración entre el asistente inteligente y el DAW externo, en este caso Reaper. Su función principal es automatizar y sincronizar tareas como la gestión de pistas, la asignación de instrumentos, el ajuste de tempo y la aplicación de efectos, asegurando que el flujo creativo del usuario sea fluido y eficiente. La automatización de estos procesos no solo reduce la carga de trabajo manual, sino que

también potencia la creatividad al permitir que el usuario se centre en las decisiones musicales mientras el sistema se encarga de la ejecución técnica.

La arquitectura principal del módulo de comunicaciones se fundamenta en tres archivos principales, cada uno de los cuales desempeña un papel muy concreto dentro del sistema. La figura 18 muestra como esta arquitectura se integra en el plano general del sistema:

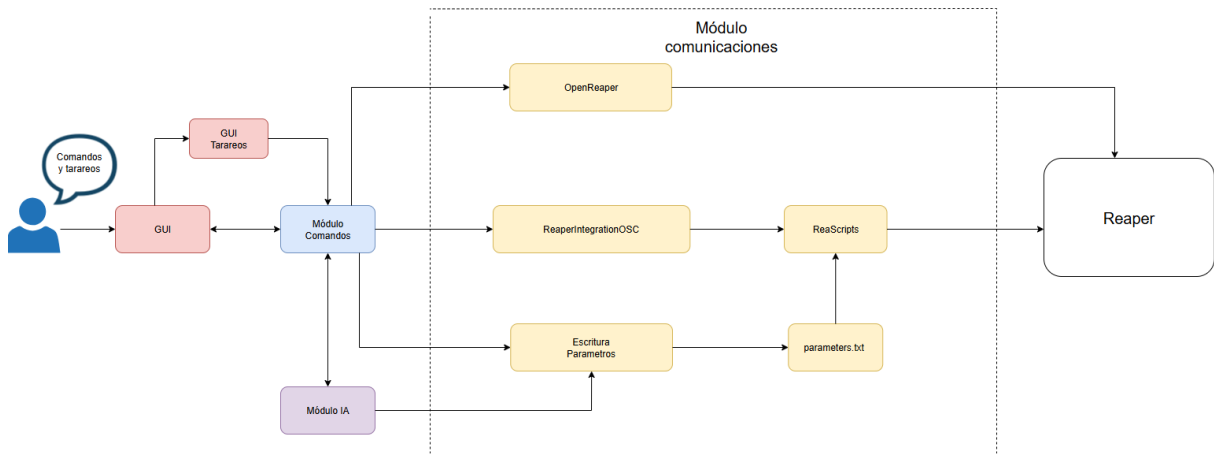


Figura 18. Arquitectura del módulo de comunicaciones

El primer archivo, **writing_parameters.py**, denominado **Escritura Parámetros** en la figura 18, se utiliza para escribir los parámetros necesarios en el archivo **parameters.txt**, cuya función real es actuar como un punto intermedio de comunicación entre el asistente y el DAW. Este archivo sirve para transferir información de forma rápida y temporal, facilitando el paso de datos esenciales del proyecto musical entre distintos componentes que los necesitan casi de inmediato. Desde el tipo de instrumento asignado a una pista hasta el tempo del proyecto, pasando por rutas de archivos MIDI o los parámetros de los efectos FX generados por la IA, este archivo asegura que la información crítica pueda comunicarse de manera correcta entre los distintos componentes. Su integración con el resto del sistema permite, por ejemplo, que cuando el usuario decide aplicar un efecto o cambiar un instrumento, los parámetros correspondientes se guarden de forma estructurada y puedan recuperarse desde otros puntos del sistema para su aplicación en el DAW.

El segundo archivo, **openReaper.py**, se encarga de gestionar la presencia y el estado del DAW Reaper en el entorno del usuario. Sus funciones van más allá de simplemente abrir el programa, ya que busca la instalación de Reaper en el registro de Windows, inicia el DAW automáticamente, gestiona posibles ventanas emergentes y verifica si Reaper ya está en ejecución, todo esto mediante distintos métodos los cuales son llamados desde otros módulos cuando es necesario. Esta gestión activa del DAW es fundamental para garantizar que el asistente pueda interactuar con Reaper en cualquier momento, sin requerir intervención manual por parte del usuario. Además, el archivo incluye métodos para cerrar ventanas emergentes y asegurar que la experiencia de uso sea lo más fluida posible.

Por último, **reaperIntegrationOSC.py** es el corazón de la comunicación entre el asistente y el DAW. A través del protocolo OSC, este archivo permite enviar comandos y parámetros de

forma directa a Reaper. El archivo tiene dos funciones principales: una llamada **send_command**, que se utiliza para que en Reaper se ejecute la acción con el ID que se le pase por parámetro, y otra llamada **modify_tempo**, la cual se utiliza para modificar el *tempo* en Reaper de forma automática pasando el valor del tempo como parámetro. Esto convierte a este archivo en el puente esencial entre la lógica del asistente y las acciones reales dentro del entorno de producción musical. En la figura 19 se muestra el funcionamiento de este archivo y como se integra con el resto del sistema:

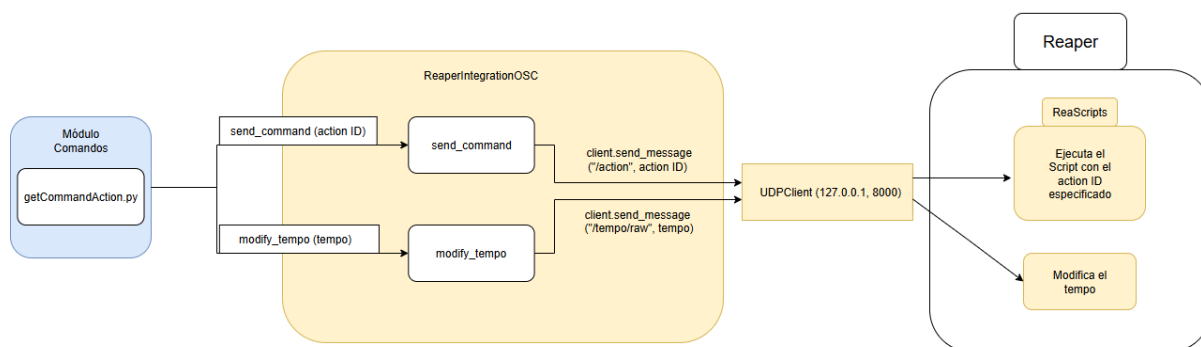


Figura 19. Integración protocolo OSC en arquitectura del módulo de comunicaciones

Para que esta comunicación funcione correctamente, es imprescindible que tanto el asistente como Reaper estén configurados adecuadamente. En el archivo **reaperIntegrationOSC.py**, se define un cliente UDP que envía los mensajes OSC a la dirección IP (Internet protocol) local 127.0.0.1 y al puerto específico 8000, que es el mismo que debe estar configurado en Reaper para que el DAW escuche los comandos enviados desde el asistente. Esta configuración se realiza desde las preferencias de Reaper, en la sección de Control/OSC/web, donde se añade un nuevo dispositivo OSC y se define el puerto de escucha local.

OSC es un protocolo de red diseñado específicamente para el control en tiempo real de dispositivos y aplicaciones musicales. A diferencia de MIDI, que utiliza mensajes binarios con un vocabulario limitado y valores de control restringidos, OSC emplea mensajes estructurados en formato de texto, lo que permite una gran flexibilidad y precisión en la definición y transmisión de comandos.

Cada mensaje OSC consta de una ruta y uno o más argumentos. La dirección es similar a una URL, con una estructura jerárquica que facilita la organización y el filtrado de comandos. Los argumentos pueden ser números enteros, flotantes, cadenas de texto, booleanos, etc., lo que permite enviar valores de alta resolución y datos complejos.

Una de las principales ventajas de OSC es su capacidad de definir mensajes personalizados. Cada aplicación puede crear sus propias rutas y argumentos, adaptándose a sus necesidades específicas. En el contexto de Reaper, la comunicación OSC no solo depende de la correcta configuración del puerto y la dirección IP, sino también de la definición de cómo los mensajes recibidos deben traducirse en acciones concretas dentro del DAW. Aquí es donde entra en juego el archivo `default.ReaperOSC`.

El archivo **default.ReaperOSC** es un archivo de texto que contiene la definición de las rutas OSC que Reaper puede interpretar y las acciones asociadas a cada una. Este archivo actúa como un traductor entre los mensajes OSC recibidos y las funciones internas de Reaper. Por ejemplo, si el asistente envía un mensaje a la ruta /action, el archivo **default.ReaperOSC** indica qué acción debe ejecutar Reaper en respuesta a ese mensaje. El parámetro que acompaña a /action es el ID de la acción que se desea ejecutar en Reaper. Estos IDs corresponden a acciones que pueden estar predefinidas por el propio Reaper o creadas por el usuario mediante ReaScripts.

La estructura del archivo permite definir asignaciones personalizadas para cada ruta OSC, aunque las utilizadas en este proyecto se encuentran por defecto en el archivo **default.ReaperOSC**. En la figura 20 se muestra parte del contenido del archivo default.ReaperOSC:

```
466 ACTION i/action s/action/str t/action/@ f/action/@/cc
467 ACTION_SOFT f/action/@/cc/soft
468 ACTION_RELATIVE f/action/@/cc/relative
469 MIDIACTION i/midiaction t/midiaction/@
470 MIDILISTACTION i/midilistaction t/midilistaction/@
471 |
472 # -----
473
474 # The following messages are sent from the device, to inform REAPER
475 # of a change in the device state, behavior, or display capabilities.
476
477 DEVICE_TRACK_COUNT i/device/track/count t/device/track/count/@
478 DEVICE_SEND_COUNT i/device/send/count t/device/send/count/@
```

Figura 20. Ejemplo visual del archivo default.ReaperOSC

La automatización avanzada en el sistema se logra gracias a la integración de scripts personalizados de Reaper, desarrollados principalmente en Lua, que permiten ejecutar acciones complejas dentro del DAW de forma precisa y eficiente. Estos scripts se agrupan en un submódulo específico dentro del módulo de comunicaciones, facilitando la gestión y el mantenimiento de las funcionalidades más sofisticadas del asistente.

La cohesión entre el módulo de comandos y el módulo de comunicaciones es fundamental para el funcionamiento del sistema. Cuando el módulo de comandos interpreta una intención del usuario, coordina la ejecución de la acción correspondiente invocando funciones del módulo de comunicaciones, que a su vez envía comandos OSC para ejecutar los ReaScripts necesarios en Reaper incluyendo el ID del script como parámetro en la función send_command de reaperIntegrationOSC.py. Este flujo garantiza que cada acción del usuario se traduzca en una operación automatizada y precisa dentro del DAW, manteniendo la coherencia y la sincronización entre el asistente y Reaper.

Un aspecto clave del sistema es el uso del archivo **parameters.txt** como punto central de comunicación. Todos los valores de parámetros necesarios para la ejecución de los scripts,

como el número de pista, el tipo de instrumento, los ajustes de efectos o cualquier otro valor relevante se escriben en este archivo desde el asistente. Cada script, al ejecutarse en Reaper, lee los valores necesarios directamente de **parameters.txt**, asegurando que la configuración sea siempre la correcta y que los datos estén perfectamente sincronizados entre el asistente y el DAW, para ello en `commandController.py` del módulo comandos se añaden estos valores a `parameters.txt` e inmediatamente se llama a la acción correspondiente en Reaper.

A continuación, se detalla la función de cada uno de los scripts de ReaScript creados para el correcto funcionamiento de las distintas funcionalidades del asistente:

- **Cargar DSKVirtuoso:** Carga el efecto DSK Virtuoso en la pista especificada, utilizando los parámetros leídos de `parameters.txt`.
- **CargarEfectos:** Aplica los efectos de mejora automática a una pista seleccionada, según la funcionalidad de mejora automática de la melodía del asistente. Los parámetros de cada efecto se obtienen del archivo `parameters.txt`.
- **CargarInstrumento:** Asigna un instrumento de DSK Virtuoso a una pista, utilizando el número de instrumento y otros parámetros leídos de `parameters.txt`.
- **CargarParametros:** Carga los valores de los parámetros generados previamente por la IA y almacenados en `parameters.txt`, aplicándolos a cada uno de los efectos de la funcionalidad de mejora automática de la melodía.
- **CargarReaSynth:** Carga el sintetizador nativo de Reaper en la pista especificada, utilizando los parámetros leídos de `parameters.txt`.
- **CargarParametrosReaSynth:**
Carga los valores de los parámetros generados por el modelo de IA y almacenados en `parameters.txt`, aplicándolos al efecto ReaSynth para crear el sintetizador de la funcionalidad específica.
- **CerrarDSKVirtuoso:** Cierra la ventana de DSK Virtuoso cuando ya no es necesaria.
- **ImportMidi:** Importa un archivo MIDI en el proyecto, utilizando la ruta y otros parámetros leídos de `parameters.txt`.
- **MoverTimeline:** Mueve la línea de tiempo o el cursor de reproducción al inicio, es decir, al segundo 0.

Este enfoque modular y automatizado permite adaptar el sistema a nuevas necesidades o requisitos del proyecto con facilidad. Además, la documentación y el mantenimiento de los scripts son clave para facilitar la colaboración y el desarrollo futuro, asegurando que el sistema sea robusto, flexible y fácil de ampliar.

4.4.5. Módulo análisis

El módulo de análisis se encarga de procesar y extraer información relevante de los archivos MIDI generados o importados en el sistema, permitiendo tanto la generación automática de una línea de bajo sobre una melodía como el análisis detallado de la estructura musical. Este módulo está compuesto por dos archivos principales: `bass_midi.py` y `midi_analyzer.py`, cada uno con una función específica dentro del flujo de trabajo del asistente. En la figura 21 se muestran los funcionamientos principales del módulo presente:

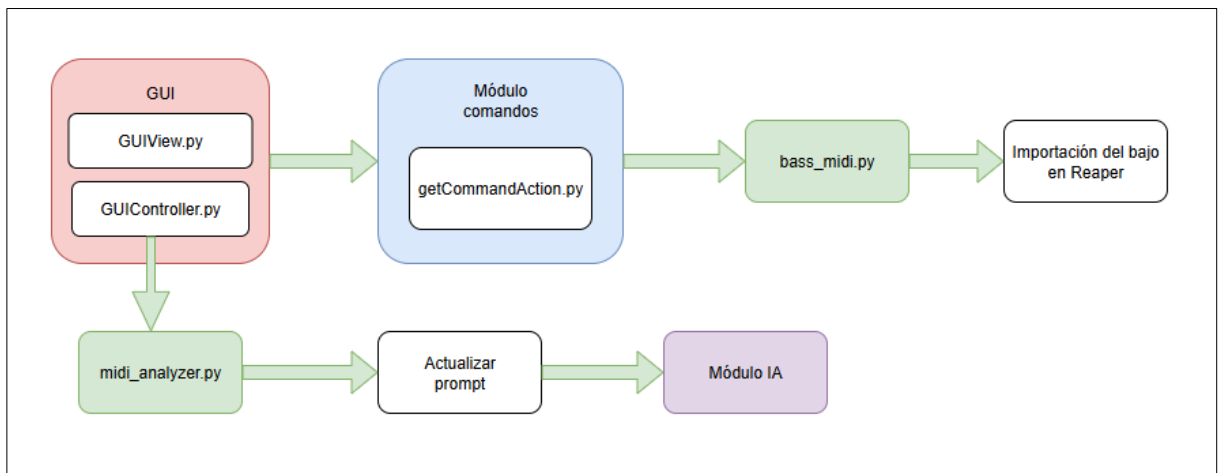


Figura 21. Funcionamiento del módulo de análisis

El archivo **bass_midi.py** utiliza la librería `music21` para procesar archivos MIDI y generar automáticamente una línea de bajo a partir de la melodía principal. El proceso comienza cargando el archivo MIDI y analizando su estructura. Si la melodía no tiene compases definidos, se asignan compases de 4/4 por defecto y se segmenta la melodía en medidas. A continuación, se analiza la tonalidad del archivo y se genera una línea de bajo sencilla, tomando como referencia la primera nota de cada compás de la melodía principal y transponiéndola una octava por debajo. Cada nota de bajo generada tiene una duración igual al compás correspondiente, asegurando que el bajo acompañe a la melodía de manera coherente. Una vez generada la línea de bajo, el archivo permite guardar tanto la partitura completa como la línea de bajo por separado, facilitando su uso posterior en el proyecto musical. El archivo está diseñado para funcionar de forma robusta, verificando en cada paso que los datos sean correctos y que los archivos se guarden correctamente. En la figura 22 se muestra el flujo de funcionamiento del archivo **bass_midi.py**:

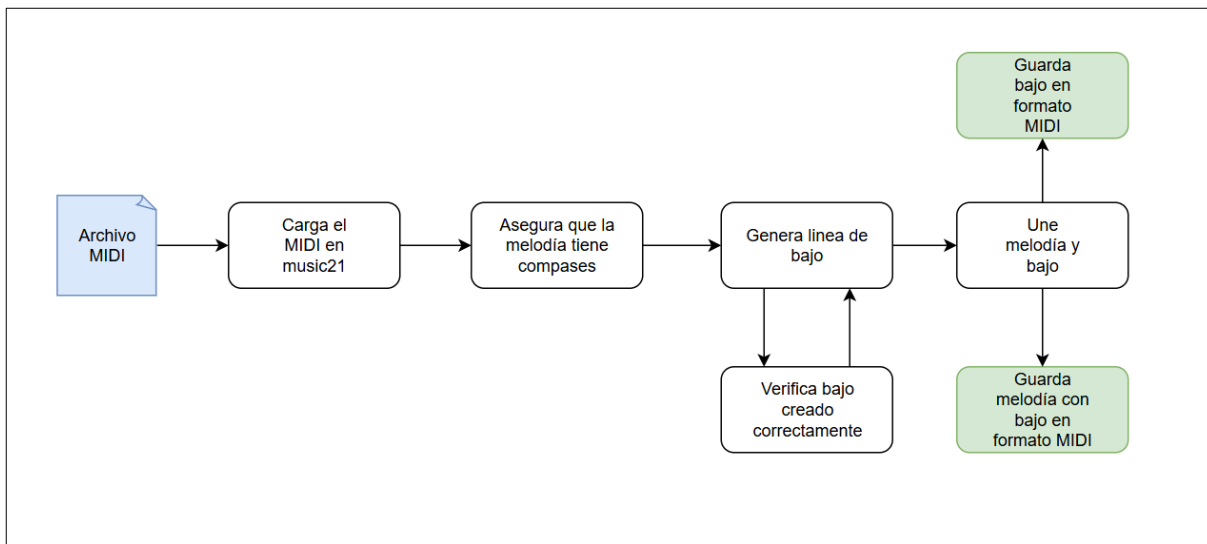


Figura 22. Flujo de funcionamiento de `bass_midi.py`

El archivo **midi_analyzer.py** implementa una clase llamada **MIDIAnalyzer**, que permite analizar archivos MIDI de forma avanzada. Utilizando la librería `music21`, esta clase extrae información como la tonalidad, las notas más frecuentes, el rango melódico, los intervalos entre notas, los valores de velocidad de las notas y la progresión de acordes detectada en el archivo. El análisis se realiza tanto para notas individuales como para acordes, permitiendo obtener una visión detallada de la estructura musical del archivo. En la tabla 1 se muestran todos los métodos de análisis del MIDI junto a la descripción de estos:

Método análisis	Descripción
<code>get_tonality</code>	Devuelve la tonalidad
<code>get_most_common_notes</code>	Devuelve las notas más frecuentes
<code>get_melody_range</code>	Devuelve el rango de alturas de la melodía
<code>get_intervals</code>	Devuelve los intervalos entre notas consecutivas
<code>get_velocity_values</code>	Devuelve los valores de volumen de las notas
<code>get_chord_progression</code>	Devuelve la progresión de acordes

Tabla 1. Métodos de análisis del archivo MIDI

Para comprender en que parte del proyecto se utiliza este módulo, el controlador del módulo GUI se comunica directamente con **midi_analyzer.py** para solicitar el análisis de archivos MIDI en diferentes momentos del flujo de trabajo, permitiendo al usuario obtener información detallada sobre la estructura musical de sus proyectos. Por su parte, el módulo de comandos utiliza **bass_midi.py** para generar automáticamente la línea de bajo correspondiente a la melodía principal, integrando este resultado en el proyecto musical según la funcionalidad requerida.

4.4.6. Módulo IA

El módulo de IA es uno de los pilares fundamentales del sistema, ya que aporta la capacidad de generar recomendaciones y parámetros musicales personalizados de manera automática, adaptándose al contexto y necesidades de cada creación musical. Este módulo permite que el asistente no solo automatice tareas técnicas, sino que también aporte valor creativo, facilitando la toma de decisiones musicales complejas y optimizando el flujo de trabajo del usuario, dándole ese punto de inteligencia al sistema. El módulo de IA está compuesto por tres archivos principales: **AIRecommendations.py**, **FXAutomaticImprovements.py** y **synthesizer.py**. Cada uno cumple una función específica dentro del proceso de generación y aplicación de parámetros musicales. En la figura 23 se muestra la interacción que realizan entre si los tres archivos de este módulo entre ellos y con el módulo de comandos desde el cual se llaman:

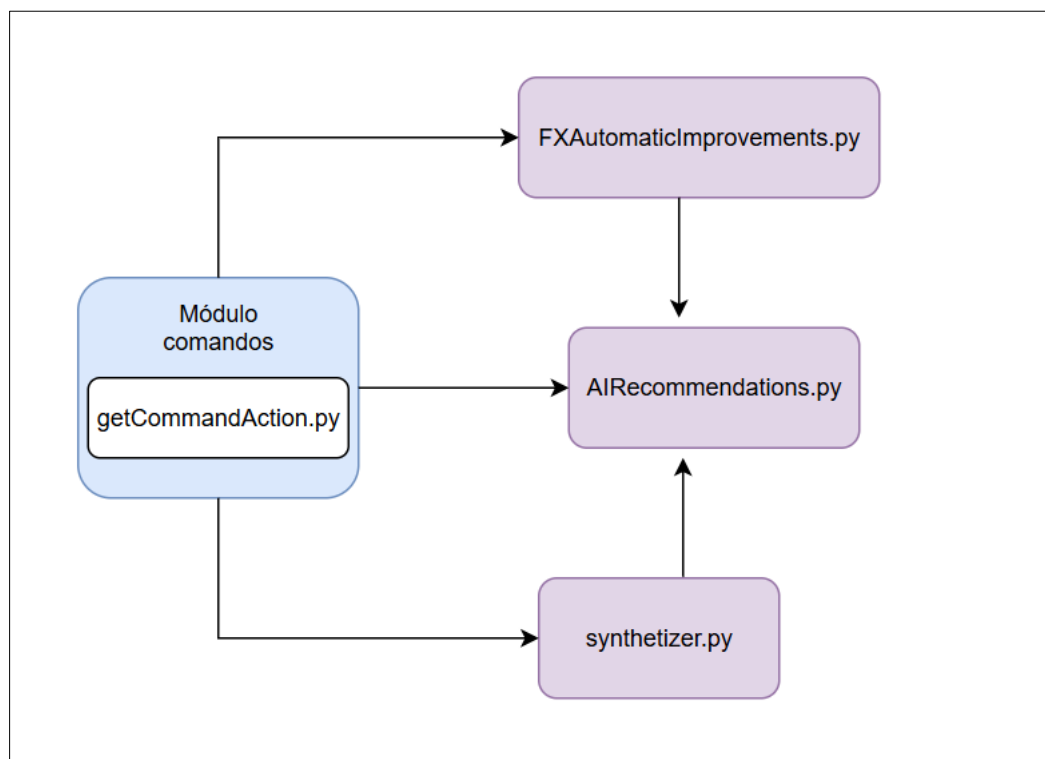


Figura 23. Interacción entre archivos del módulo de IA

El primer archivo de este módulo, **AIRecommendations.py**, actúa como puerta de entrada entre el sistema y el modelo de lenguaje avanzado utilizado en el proyecto a través de la API de OpenRouter. Se configura el cliente para conectarse a la plataforma OpenRouter, que permite acceder a una variedad de modelos de IA de última generación. El modelo seleccionado para este sistema es **deepseek/deepseek-chat-v3-0324:free**, elegido tras un proceso de comparación exhaustivo con otros modelos disponibles en OpenRouter.

La elección de este modelo se debe a varios factores clave:

- **Estabilidad y fiabilidad:** Durante las pruebas, el modelo se mostró como uno de los más estables, con una tasa mínima de caídas y tiempos de respuesta bajos, lo que es esencial para un flujo de trabajo fluido y sin interrupciones.
- **Precisión en las respuestas:** El modelo es capaz de comprender *prompts* musicales complejos y devolver respuestas estructuradas en formato JSON, lo que facilita la integración directa con el sistema y reduce la necesidad de procesamiento adicional, es decir, el modelo es capaz de responder de manera precisa ante los requerimientos del sistema.
- **Consistencia:** Otros modelos alternativos presentaron problemas de inestabilidad, de corte de funcionalidad durante horas, respuestas menos estructuradas o mayores latencias, mientras que el elegido mantuvo un rendimiento óptimo incluso bajo carga.

La integración con OpenRouter aporta ventajas adicionales, como la posibilidad de cambiar de modelo fácilmente, cambiando simplemente el nombre del modelo en el código, si en el futuro aparecen opciones más potentes o adaptadas a nuevas necesidades, y la garantía de un servicio con balanceo de carga y alta disponibilidad.

Este archivo también incluye una función que, al ser llamada con el *prompt* deseado como parámetro, devuelve la respuesta generada por el modelo.

Los archivos **FXAutomaticImprovements.py** y **synthesizer.py** comparten una filosofía y estructura de trabajo muy similar. En ambos casos, el proceso se basa en la construcción de un *prompt* detallado y contextualizado, que es fundamental para obtener respuestas precisas y útiles del modelo de IA. Estos *prompts* incluyen información sobre el estilo, el contexto musical, las necesidades del usuario y una descripción clara de los parámetros que se deben generar, así como el formato de la respuesta esperada, en formato JSON. Esta estrategia asegura que el modelo de IA reciba toda la información necesaria para devolver valores relevantes y ajustados para la integración con Reaper. En la figura 24 se muestra un ejemplo visual del formato del *prompt* utilizado en el archivo **synthesizer.py**:

```

improvementPrompt = """Quiero crear un sintetizador básico sobre la melodía de mi MIDI en Reaper (ReaSynth).
Me gustaría que el sintetizador no sea muy agresivo, que sea un toque suave pero que se note el sonido sintetizado.
Debes completar los siguientes parámetros basándote en el estilo que indique en la producción:

---
Tuning (-1200 - 1200 cent)
Attack (0-500ms)
Decay (0-5000ms)
Sustain (entre -60dB y 0dB)
Release (0-5000ms)
Portamento (0-500ms)
Square Mix (0-1)
Sawtooth Mix (0-1)
Triangle Mix (0-1)
Extra Sine Mix (0-1)

(No uses Pulse Width ni Extra Sine Tune, déjalos fijos en 0.5 y 0 respectivamente.)
---

Responde solo en JSON, así:

{
  "Tuning": "0",
  "Attack": "10ms",
  "Decay": "1200ms",

```

Figura 24. Ejemplo visual del prompt utilizado en synthetizer.py

El archivo FXAutomaticImprovements.py está diseñado para automatizar la configuración de efectos de audio en Reaper, como ecualización, compresión, reverberación y otros efectos clave para la producción musical, mejorando de manera automática y con el uso del modelo de IA la melodía. El proceso comienza con la generación de un *prompt* detallado, que describe el contexto musical completo y las necesidades específicas del usuario o del proyecto. Este *prompt* se envía al modelo de IA a través de la función definida en AIRecommendations.py, solicitando valores óptimos para cada parámetro de los efectos.

La respuesta del modelo de IA, en formato JSON, incluye el valor de cada uno de los parámetros de cada efecto, lo que facilita su procesamiento posterior. Aquí entra en juego una parte crucial del sistema: las funciones de conversión. Estas funciones son necesarias porque los valores que devuelve la IA, por ejemplo, 1200ms del parámetro *attack* o -6dB del parámetro *sustain*, no siempre coinciden con los rangos o formatos que utilizan los *plugins* de Reaper de forma interna, es decir el valor que observa el usuario en el *plugin* no es el mismo que el que está manejando el *plugin* por detrás. Por eso, se implementan funciones específicas para cada parámetro, que transforman los valores musicales a los rangos numéricos reales que requiere el software. Esto garantiza que los parámetros sean directamente aplicables y evita errores de automatización o sonidos no deseados. Estas funciones son diferentes para cada caso de parámetro, hay algunas que realizan una transformación lineal para los parámetros que funcionan de esa manera mientras que otras se realiza una transformación logarítmica debido a que son parámetros que trabajan con decibelios.

En la tabla 2 se detalla el funcionamiento de cada una de las funciones de conversión:

Función (x)	Efecto	Funcionamiento
scale_frequency	ReaEQ	Convierte la frecuencia a una escala interna no lineal mediante una curva logística para control más musical de filtros.
convert_bandwidth	ReaEQ	Convierte el parámetro ancho de banda a escala 0 – 100 con una relación lineal multiplicándolo por 25.
db_to_percent_in_REAEQ	ReaEQ	Transforma la ganancia en dB a un porcentaje interno usado por ReaEQ. Usa curva exponencial para bajos valores y lineal para valores altos.
percent_to_db_in_REACOMP	ReaComp / ReaVerbate	Convierte porcentaje de interfaz a dB reales en parámetros como Threshold, Dry Level o Wet Level. Curva mixta: exponencial y polinómica.
ratio_filter	ReaComp	Ajusta el ratio con una curva personalizada que introduce una corrección decimal adicional para mejorar la respuesta en ReaComp.
percent_lineal_in_REACOMP	ReaComp / ReaVerbate	Convierte parámetros como Attack, Release, Pre-comp, Low Cut y High Cut a escala 0–100 según su valor máximo. Proporcional lineal.
stereo_width	Stereo Enhancer	Convierte el parámetro de Width de rango 0–1 a escala 0–100.
whape_shape	ReaSynth	Convierte mezclas de ondas de escala 0–1 a 0–100.
portamento	ReaSynth	Convierte el parámetro Portamento dividiéndolo por 10 para adaptarlo al rango interno esperado por ReaSynth.
tunning	ReaSynth	Ajusta Tuning mediante una transformación lineal para mapearlo al rango interno del sintetizador.
attack	ReaSynth	Convierte el parámetro Attack reduciéndolo con una división por 5 para su aplicación directa en ReaSynth.
release	ReaSynth	Escala el parámetro Release dividiendo por 50 para ajustarse a la interpretación del <i>plugin</i> .
decay	ReaSynth	Transforma el parámetro Decay con una relación lineal personalizada para compresión a la escala esperada.
convert_db_to_output	ReaSynth	Convierte valores de Sustain en dB a una escala arbitraria usada internamente, mediante interpolación entre puntos de referencia.

Tabla 2. Funciones de conversión de valores de parámetros

El archivo **synthesizer.py** se centra en la creación de ajustes personalizados para el sintetizador nativo ReaSynth de Reaper. Utilizando descripciones textuales y *prompts* contextuales, el modelo de IA genera valores para los diez parámetros clave del sintetizador, como *Tuning*, *Attack*, *Decay*, *Sustain*, *Release*, *Portamento*, y las mezclas de formas de onda.

El objetivo es conseguir un sonido sintético equilibrado, suave y adaptado a la melodía principal del proyecto.

Aquí, el *prompt* que se envía al modelo de IA incluye una descripción del tipo de sonido deseado para el sintetizador, junto con una lista clara de los parámetros que debe devolver la IA y sus respectivos formatos. La respuesta, al igual que en el caso de la mejora con efectos FX, se procesa con funciones de conversión que adaptan los valores a los rangos que utiliza ReaSynth. Esto asegura que el sonido final sea musicalmente coherente y técnicamente válido.

En ambos archivos, el flujo de trabajo es casi idéntico: se construye el *prompt*, se envía al modelo de IA, se recibe la respuesta en JSON, se procesan los valores con funciones de conversión, y finalmente se guardan en **parameters.txt**. Esta uniformidad metodológica facilita el mantenimiento del sistema y la posibilidad de escalar o añadir nuevas funcionalidades en el futuro. En la figura 25 se muestra el funcionamiento interno de estos archivos desde su llamada desde el módulo de comandos hasta la salida de datos mediante el archivo **parameters.txt** para ser usados por el módulo de comunicaciones e importar los datos a los efectos correspondientes en Reaper:

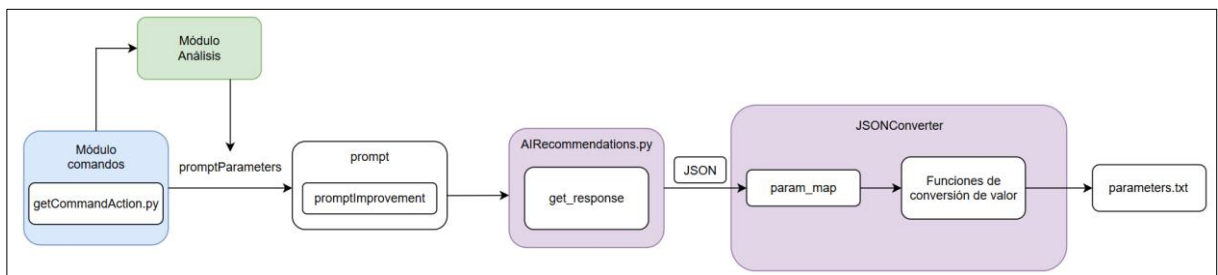


Figura 25. Flujo de proceso de mejoras de melodía mediante efectos o sintetizador

Uno de los aspectos más relevantes del módulo de IA es su integración con el resto del sistema, especialmente con el módulo de comunicaciones. El flujo de datos para las funcionalidades de mejora con efectos FX y la funcionalidad de creación de sintetizados sigue una secuencia clara y estructurada:

- **Generación de parámetros:** El usuario solicita una mejora de efectos o la creación de un sintetizador sobre su melodía. El módulo de comandos llama a la función correspondiente en FXAutomaticImprovements.py o synthesizer.py, que a su vez genera el *prompt* adecuado a cada caso y lo añade al *prompt* que contiene todo el contexto de la creación musical hasta el momento y lo envía al modelo de IA a través de AIRecommendations.py.
- **Recepción y procesamiento:** La IA devuelve una respuesta en formato JSON, que es procesada y convertida a los valores numéricos requeridos por los *plugins* o sintetizadores de Reaper a través de unas funciones de conversión.

- **Almacenamiento en parameters.txt:** Los valores finales se escriben en el archivo parameters.txt en un formato estándar (número de pista, identificador de parámetro, valor), que sirve como punto de intercambio entre el módulo de IA y el módulo de comunicaciones.
- **Aplicación en Reaper:** Cuando corresponde, mediante el módulo de comunicaciones se envía un comando OSC a Reaper para ejecutar el ReaScript adecuado (por ejemplo, para cargar parámetros en ReaSynth o en los efectos FX). El ReaScript lee los valores de parameters.txt y los aplica automáticamente a cada uno de los efectos necesarios en cada caso, cerrando así el ciclo entre la generación creativa y la ejecución técnica.

En la figura 26 se muestra el flujo que siguen los datos de salida del módulo presente para poder ser importados en los efectos correspondientes en Reaper, gracias a la integración con el módulo de comunicaciones:

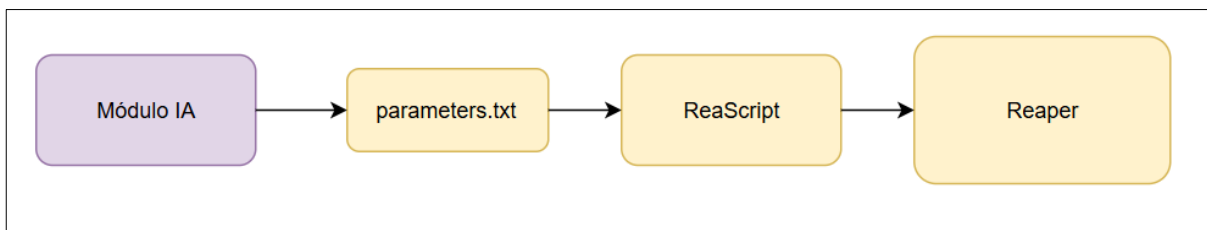


Figura 26. Flujo de aplicación de valores de parámetros de efectos en Reaper

Este flujo automatizado permite que el usuario pase de una idea musical a una configuración técnica avanzada mejorada en cuestión de segundos, sin necesidad de conocimientos profundos de síntesis o procesamiento de audio.

5. Resultados

El objetivo de este capítulo es analizar y presentar de manera sistemática las pruebas realizadas sobre el asistente inteligente desarrollado, con el fin de comprobar su correcto funcionamiento y validar que cumple con los requisitos planteados en el diseño. Se han diseñado diferentes tipos de pruebas que abarcan tanto la robustez interna de los módulos como la experiencia de usuario final, permitiendo así evaluar la fiabilidad, eficiencia y utilidad real del sistema.

A lo largo de este apartado se describen los métodos utilizados para cada tipo de prueba, los resultados obtenidos y una interpretación crítica de los mismos. El enfoque adoptado busca garantizar la reproducibilidad de los experimentos y facilitar la comprensión de los puntos fuertes y las posibles áreas de mejora del sistema.

5.1. Pruebas unitarias

Las pruebas unitarias consisten en verificar el correcto funcionamiento de las unidades más pequeñas y aisladas del código, como funciones o métodos, asegurando que cada una de ellas cumpla su cometido de manera independiente y eficiente. Aunque lo ideal es automatizar este tipo de pruebas para facilitar su ejecución y repetibilidad, en este proyecto se optó por una metodología manual, apoyada en herramientas de depuración y registros detallados, dadas las características del desarrollo y la integración de múltiples tecnologías externas.

Durante el proceso de desarrollo, se emplearon de manera intensiva las capacidades del *debugger* del entorno de programación, lo que permitió inspeccionar el flujo de ejecución paso a paso y analizar el comportamiento de cada función en situaciones tanto habituales como excepcionales. Además, se implementaron *logs* en puntos críticos del sistema, registrando las entradas y salidas de las funciones más relevantes, lo que facilitó la monitorización en tiempo real y la identificación rápida de posibles anomalías.

Cada módulo clave del asistente fue sometido a pruebas específicas, comprobando que las funciones de conversión de audio a MIDI, la generación y tratamiento de *prompts* para el modelo de IA, la gestión de archivos intermedios y la comunicación OSC con Reaper respondieran correctamente ante diferentes tipos de entrada, incluyendo casos límite y valores inesperados. Esta estrategia permitió validar no solo el comportamiento esperado, sino también la robustez del sistema ante posibles errores o situaciones imprevistas.

Los resultados obtenidos confirmaron que todas las funciones críticas del sistema operan de manera estable y fiable, gestionando adecuadamente tanto los casos estándar como las excepciones. La exhaustividad de la revisión manual y la repetición de los casos de prueba más relevantes permitieron detectar y corregir cualquier anomalía antes de avanzar a fases de integración y pruebas de usuario. Aunque no se dispuso de un sistema automatizado de pruebas, la combinación de depuración paso a paso y *logging* detallado resultó suficiente para asegurar la calidad y estabilidad del código en esta fase del proyecto [54].

5.2. Pruebas de caja negra

Las pruebas de caja negra [55] se enfocan en evaluar el comportamiento del sistema desde la perspectiva del usuario final, sin considerar la estructura interna o el código subyacente. El objetivo principal de estas pruebas es verificar que el asistente responda correctamente a una amplia variedad de comandos y escenarios, asegurando que las funcionalidades implementadas se comporten según lo esperado en condiciones reales de uso.

Para ello, se diseñó una batería exhaustiva de pruebas que abarcan no solo las funcionalidades básicas del asistente, sino también flujos alternativos, combinaciones de acciones y situaciones límite que podrían presentarse durante la interacción del usuario con el sistema. Esta aproximación garantiza una evaluación completa y robusta del sistema, permitiendo identificar posibles fallos, inconsistencias o áreas de mejora.

A continuación, en la tabla 3 se resumen las pruebas realizadas, describiendo cada escenario o acción probada y el resultado real observado durante la ejecución:

Nº	Descripción de la prueba	Resultado
1	Grabar un tarareo y abrir el archivo MIDI resultante automáticamente en Reaper	MIDI generado y abierto en Reaper en pista nueva automáticamente
2	Grabar un tarareo y guardar el archivo MIDI en la carpeta local del usuario	MIDI generado y guardado correctamente en la carpeta designada
3	Abrir un archivo MIDI existente desde la carpeta del proyecto en Reaper	Archivo MIDI abierto en pista nueva sin intervención manual
4	Cambiar el instrumento asignado a una pista MIDI específica	Instrumento cambiado automáticamente al solicitado
5	Aplicar efectos automáticos con parámetros optimizados por IA a una pista	Efectos aplicados con parámetros adaptados a la melodía
6	Solicitar sugerencias de mejora sobre la creación musical actual	Sugerencias mostradas en pantalla basadas en análisis IA
7	Realizar una pregunta específica al asistente sobre producción musical	Respuesta clara y contextualizada proporcionada por IA
8	Generar una línea de bajo complementaria a la melodía principal	Línea de bajo generada, guardada e importada en pista nueva
9	Añadir un sintetizador con parámetros personalizados a la melodía actual	Sintetizador añadido con parámetros adaptados y aplicados en pista nueva
10	Grabar y guardar un tarareo, luego abrir el MIDI guardado en Reaper	MIDI guardado y abierto correctamente en Reaper
11	Abrir un MIDI, cambiar instrumento y aplicar efectos en secuencia	Todas las acciones ejecutadas correctamente y en orden

Nº	Descripción de la prueba	Resultado
12	Generar bajo y luego añadir sintetizador sobre la pista de bajo	Bajo generado y sintetizador añadido sin conflictos
13	Intentar abrir un archivo MIDI inexistente	Mensaje del asistente claro indicando que el archivo no existe
14	Cancelar la grabación de tarareo antes de finalizar	Proceso cancelado sin errores ni efectos secundarios
15	Cambiar instrumento en pista vacía	Mensaje del asistente indicando que la pista está vacía
16	Ejecutar múltiples comandos en rápida sucesión	El sistema mantiene estabilidad y ejecuta comandos en orden
17	Abrir varios archivos MIDI en pistas consecutivas	Cada MIDI se añade en una pista nueva y secuencial, sin sobrescribir pistas previas
18	Generar línea de bajo sobre melodía compleja	Bajo generado correctamente adaptado al contenido musical
19	Añadir sintetizador tras aplicar efectos a la melodía	Sintetizador añadido manteniendo efectos previos
20	Aplicar efectos al último MIDI abierto tras abrir varios MIDIs	Efectos aplicados únicamente a la pista del último MIDI abierto
21	Generar bajo sobre el primer MIDI abierto, luego abrir otro MIDI y generar bajo sobre este	Cada línea de bajo se asigna correctamente a la pista correspondiente
22	Aplicar efectos a una pista tras haber cambiado el instrumento en esa pista	Los efectos se aplican correctamente sobre la pista teniendo en cuenta el nuevo instrumento
23	Abrir un MIDI, aplicar efectos, luego abrir otro y comprobar que los efectos no se aplican automáticamente al nuevo	Solo la pista original mantiene los efectos, la nueva pista permanece limpia hasta que se le apliquen

Tabla 3. Prueba de caja negra del sistema

Esta batería de pruebas demuestra que el sistema es capaz de gestionar correctamente la apertura y manipulación de múltiples archivos MIDI, asignando acciones como efectos, cambios de instrumento, generación de bajo o sintetizador a la pista adecuada según el contexto. Se valida que no existen sobreescrituras accidentales, que las pistas se mantienen independientes y que el flujo de trabajo es predecible y robusto incluso en escenarios complejos o con acciones encadenadas. Además, el sistema gestiona adecuadamente situaciones de error, como intentos de abrir archivos inexistentes o aplicar acciones sobre pistas vacías, proporcionando mensajes claros y evitando comportamientos inesperados.

En conjunto, estas pruebas de caja negra validan la funcionalidad, usabilidad y robustez del sistema desde la perspectiva del usuario final, cumpliendo con los objetivos de calidad y experiencia de usuario propios de una solución de asistencia para producción musical.

5.3. Pruebas de reconocimiento de voz

El reconocimiento de voz constituye una funcionalidad esencial para el asistente, ya que permite una interacción natural y fluida mediante comandos hablados. Para evaluar su eficacia y robustez, se diseñaron pruebas diferenciadas según el tipo de entrada y el motor de reconocimiento empleado.

Se seleccionó un conjunto representativo de comandos habituales, que fueron repetidos múltiples veces para obtener datos estadísticamente significativos. Estas pruebas se realizaron bajo dos configuraciones principales: primero utilizando únicamente el motor de reconocimiento Vosk, y luego combinando Vosk con Wit.ai, donde Vosk se encarga de la transcripción y Wit.ai interpreta semánticamente el contenido. Los comandos evaluados incluyeron tanto palabras sueltas como frases completas, para cubrir distintos niveles de complejidad en la interacción.

Para la primera configuración se utilizó únicamente el motor Vosk para reconocer palabras individuales relacionadas con el dominio musical y los comandos básicos del asistente. Se seleccionaron términos clave como grabar, abrir, guardar, efectos, bajo, sintetizador, pista, etc. El objetivo era comprobar la precisión de Vosk al identificar palabras aisladas.

Para la segunda configuración se evaluó la capacidad de Vosk para transcribir frases completas, utilizando comandos naturales que cubren todas las funcionalidades del asistente. El objetivo era medir la robustez del sistema ante comandos más largos y complejos, que son habituales en el uso real del asistente. Para ello se considera como válida toda frase que se transcriba de manera correcta en todas sus palabras y el orden de estas, es decir, en caso de que hubiese una palabra cambiada, no detectada o en orden distinto se considera fallida.

Para la última configuración se probó la combinación en la que Vosk realiza la transcripción inicial y Wit.ai interpreta semánticamente el texto resultante, es decir, la que realmente se utiliza en el sistema. Se utilizan los mismos comandos que para evaluar a Vosk en solitario y como criterio se propone que si la salida del sistema coincide con el comando que se quiere probar se considera como acierto y si muestra uno diferente se considera como fallo. Es decir, en caso de que se esté probando un comando para la funcionalidad de cambio de instrumento y la salida es la funcionalidad cambio de instrumento se considera acierto y si es otra diferente se considera como fallo.

Tras realizar un total de 50 pruebas para cada una de las configuraciones, se obtienen los resultados mostrados en la tabla 4 y en la figura 27:

Configuración	Nº validas	Nº fallidas	Tasa de acierto
Vosk (Palabras)	43	7	86%
Vosk (Frases)	26	24	52%
Vosk + Wit.ai (Frases)	48	2	96%

Tabla 4. Resultados pruebas de reconocimiento de voz

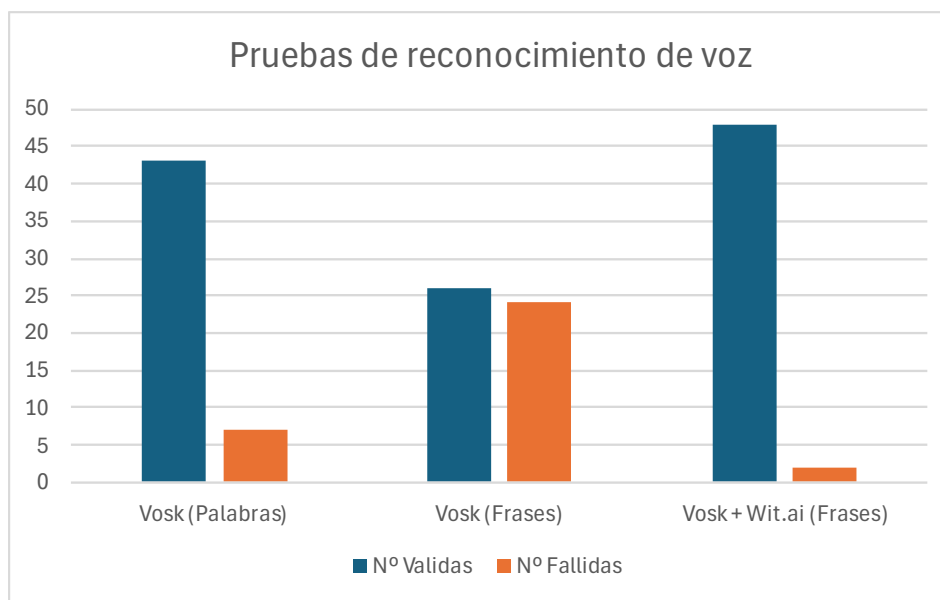


Figura 27. Gráfica de resultados pruebas de reconocimiento de voz

Tanto en la tabla 4 como en la figura 27 observa que el uso combinado de Vosk + Wit.ai incrementa la tasa de acierto en 44 puntos porcentuales respecto al uso exclusivo de Vosk en frases, pasando del 52% al 96% en la tasa de acierto. Asimismo, la configuración híbrida supera en 10 puntos porcentuales incluso al mejor caso de Vosk en solitario, es decir, el caso de las palabras sueltas, el cual además no resuelve por sí mismo el problema planteado. Estos resultados cuantifican la mejora aportada por Wit.ai al combinarlo con Vosk en escenarios de comandos reales.

Por lo tanto, los resultados obtenidos muestran claramente que la combinación de Vosk y Wit.ai mejora de forma significativa la tasa de acierto, especialmente en el reconocimiento de frases completas y comandos complejos. Mientras que Vosk por sí solo ofrece un buen rendimiento para palabras sueltas, su tasa de error aumenta notablemente en situaciones más exigentes en las cuales se tienen comandos reales del asistente. Por el contrario, la integración con Wit.ai reduce considerablemente estos errores, proporcionando una interpretación más precisa y fiable de las órdenes habladas, aumentando notablemente el porcentaje de acierto. Este análisis confirma que la estrategia de utilizar ambos motores de reconocimiento en conjunto es adecuada para garantizar una experiencia de usuario satisfactoria, minimizando malentendidos y facilitando una comunicación natural y efectiva con el asistente.

5.4. Pruebas de calidad musical

La calidad musical de las producciones generadas por el asistente es un aspecto clave para determinar su utilidad real en el proceso creativo. Aunque la evaluación musical tiene un componente subjetivo, se ha intentado objetivar el proceso mediante una metodología estructurada que combina la valoración de usuarios con distintos niveles de experiencia y algunos indicadores básicos de coherencia musical. Para ello, se diseñó una prueba en la que participaron 10 personas, incluyendo tanto músicos aficionados como personas sin experiencia previa en el ámbito musical, con el objetivo de obtener una valoración diversa y representativa. Las puntuaciones se asignaron en una escala de 1 a 5 y se pidió a los participantes que evaluaran de forma independiente los siguientes aspectos concretos:

- **Calidad de la melodía generada a partir de un tarareo:** Se valoró la naturalidad y similitud de la melodía creada a partir de una grabación vocal, la cual tiene muchas variaciones de frecuencia y ruidos.
- **Calidad de la melodía generada a partir de una melodía mostrada desde otro dispositivo:** Se evaluó cómo el sistema interpreta y transforma melodías externas creadas con instrumentos, valorando su fidelidad y musicalidad.
- **Coherencia de la línea de bajo o sintetizador con la melodía principal:** Se analizó si el acompañamiento generado por el modelo de IA resulta armónicamente adecuado y musicalmente coherente con la melodía base.
- **Nivel de mejora percibido tras la aplicación de efectos automáticos:** Se evaluó si los efectos añadidos por el sistema realmente mejoran la experiencia auditiva y aportan valor creativo.
- **Experiencia de usuario y calidad de las contestaciones y ayudas de la IA:** Se valoró la utilidad, claridad y pertinencia de las respuestas y sugerencias proporcionadas por el asistente durante el proceso creativo.

A continuación, en la tabla 5 y en la figura 28 se muestran los resultados medios obtenidos en cada una de estas dimensiones:

Aspecto evaluado	Valoración media (1-5)
Melodía generada a partir de tarareo	3,8
Melodía generada desde otro dispositivo	4,8
Coherencia de bajo/sintetizador con la melodía principal	4,5
Mejora percibida tras la aplicación de efectos	4,2
Experiencia de usuario y ayudas/contestaciones de la IA	4,7

Tabla 5. Resultados prueba de calidad musical

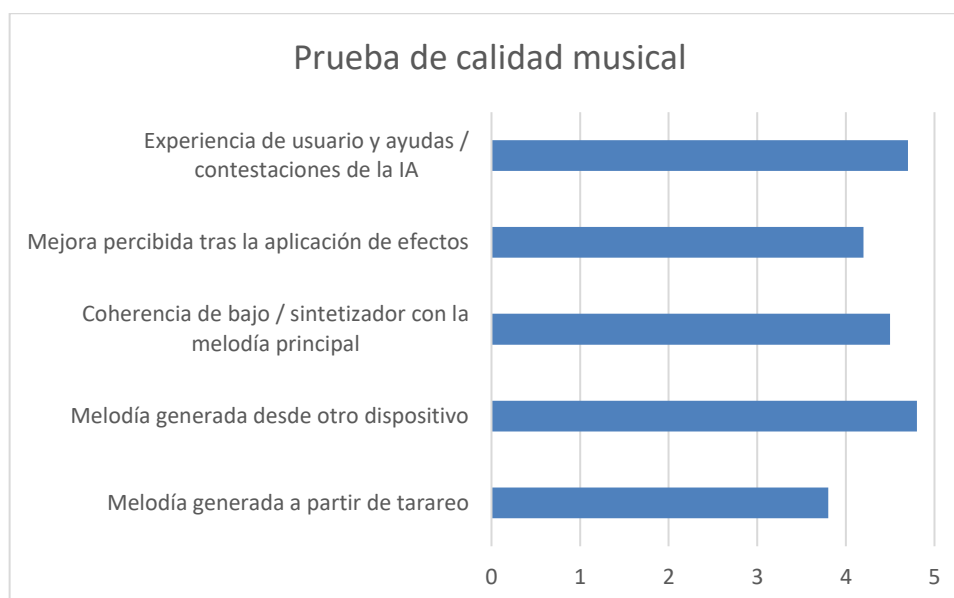


Figura 28. Gráfica de resultados prueba de calidad musical

Tanto en la tabla 5 como en la figura 28 se observa que la puntuación más alta corresponde a la calidad de melodías generadas desde otro dispositivo, con una valoración de 4,8 sobre 5, seguida de la experiencia de usuario y ayudas de la IA, con una valoración de 4,7 sobre 5. La puntuación más baja fue para las melodías generadas a partir de tarareo, la cual se valoró en 3,8 sobre 5, lo que supone una diferencia de un punto completo respecto a la mejor valoración obtenida. La coherencia de bajo/sintetizador con una valoración de 4,5 y la mejora aportada por efectos automáticos con un 4,2 se sitúan en un rango intermedio, confirmando la estabilidad general del sistema en diferentes dimensiones evaluadas.

Por lo tanto, los resultados reflejan una valoración globalmente positiva, especialmente en la naturalidad de las melodías generadas mediante grabaciones desde otro dispositivo y la utilidad de las respuestas y sugerencias del asistente, aunque algunos usuarios señalaron que, en melodías muy complejas tarareadas, el sistema puede perder parte del contexto armónico. La coherencia de los acompañamientos automáticos y la calidad de los efectos aplicados también fueron bien recibidas. La diversidad del grupo de evaluación, que incluyó tanto músicos aficionados como personas ajenas al mundo musical, permitió obtener una perspectiva equilibrada sobre la accesibilidad y el potencial creativo del sistema. En conjunto, estos resultados confirman que el asistente aporta valor real como herramienta de apoyo y experimentación, democratizando el acceso a la producción musical y facilitando la exploración creativa para usuarios de distintos perfiles.

5.5. Pruebas de comunicación e integración con Reaper

La integración entre el asistente y Reaper requiere una gestión cuidadosa de los parámetros de los efectos FX, ya que los valores generados por el modelo de IA no coinciden con los que manejan internamente los *plugins* dentro de Reaper, debido a que los valores que el usuario observa visualmente en los efectos no son los que el programa utiliza internamente. Para garantizar que el usuario vea el valor exacto sugerido por la IA, es necesario implementar

funciones de conversión específicas que adapten el valor generado al formato y rango que cada *plugin* espera internamente.

El proceso no consiste en enviar directamente el valor propuesto por el modelo de IA al *plugin*, sino que antes se aplica un filtrado mediante estas funciones de conversión. Esto tiene sentido porque, el valor que el usuario percibe en la interfaz del *plugin* no es el mismo que el valor interno que maneja el *plugin* en el software, como se ha comentado anteriormente. Así, las funciones de conversión aseguran que el valor que finalmente llega al *plugin* sea el adecuado para que, al mostrarse en la interfaz, coincida con el deseado por el usuario y sugerido por el modelo de IA.

La metodología empleada consistió en tomar el valor generado por el modelo de IA para cada parámetro, aplicar la función de conversión correspondiente y, a continuación, enviar el resultado al *plugin*. Se verificó entonces que el valor mostrado en la interfaz de Reaper fuera idéntico o lo más cercano posible al valor propuesto inicialmente. Se comprobó cada parámetro individualmente, registrando tanto los aciertos exactos como las desviaciones en los parámetros no exactos, con el objetivo de identificar y analizar cualquier error de transmisión o conversión.

Los resultados obtenidos se resumen en la tabla 6, que muestra el número total de parámetros evaluados por cada efecto, los que coincidieron exactamente, los que presentaron alguna variación y la magnitud relativa de estas desviaciones, es decir, que dimensión tiene el error en los parámetros que no son exactos:

Efecto evaluado	Parámetros evaluados	Parámetros exactos	Porcentaje acierto exacto	Parámetros con variación	Media de desviación relativa
ReaEQ	11	6	54,5%	5	6,2%
ReaComp	6	5	83,3%	1	3,3%
ReaVerbate	7	4	57,1%	3	8,1%
Stereo Enhancer	3	3	100%	0	0%
ReaSynth	10	9	90%	1	4,6%

Tabla 6. Resultados prueba de comunicación e integración con Reaper

En la figura 29 se muestra de manera gráfica el número de parámetros exactos obtenidos respecto al total para cada uno de los efectos, junto con el porcentaje de acierto que esto representa.

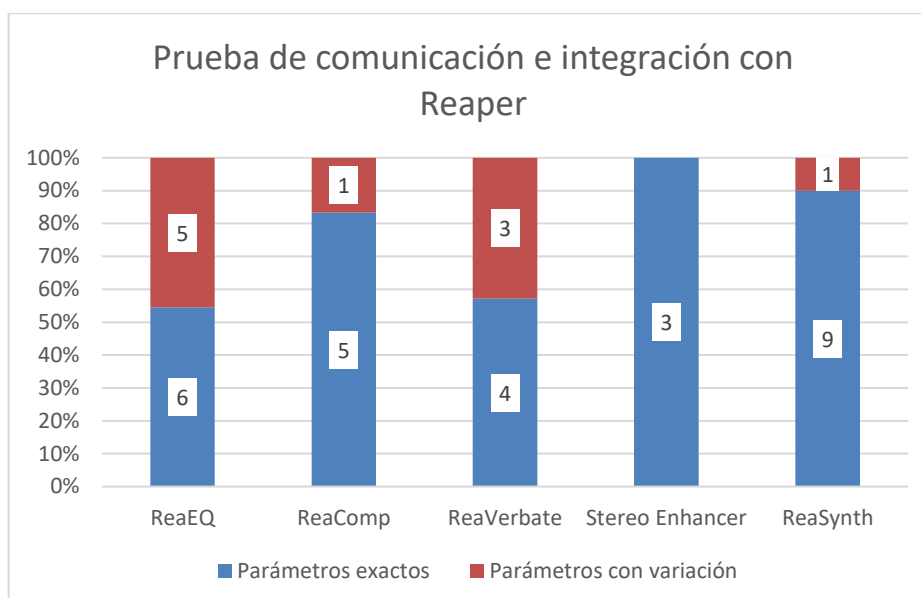


Figura 29. Gráfica de resultados prueba de comunicación e integración con Reaper

Tanto en la tabla 6 como en la figura 29 se observa que la mayor precisión se alcanzó en el efecto Stereo Enhancer, con un 100% de acierto en todos los parámetros. En contraste, los porcentajes más bajos correspondieron a ReaEQ con un 54,5% y ReaVerbate con un 57,1%, mientras que ReaComp con un 83,3% y ReaSynth con un 90% se situaron en un rango de fiabilidad más alto. En cuanto a la magnitud de las desviaciones, el valor más elevado se registró en ReaVerbate con un 8,1%, seguido de ReaEQ con un 6,2%, mientras que en ReaComp con un 3,3% y ReaSynth con un 4,6% las variaciones fueron menores.

Al analizar estos resultados, se observa que la mayoría de los parámetros se transmiten correctamente y coinciden exactamente con los valores propuestos por el modelo de IA, gracias a las funciones de conversión implementadas. Sin embargo, en algunos casos, especialmente en los parámetros de ganancia y volumen expresados en valores decibélicos, se detectaron pequeñas desviaciones debido a la complejidad de la conversión entre escalas no lineales y a la diversidad de rangos que manejan los distintos *plugins*. Por ejemplo, en ReaEQ, la mayoría de las variaciones en frecuencia fueron mínimas y no afectan significativamente la experiencia auditiva, mientras que en ganancia la desviación fue algo mayor, aunque sigue dentro de márgenes aceptables para el usuario. En ReaComp y ReaSynth, las desviaciones también se concentraron en parámetros de ganancia, mientras que en ReaVerbate se detectaron tanto variaciones en porcentaje como en ganancia.

Es importante destacar que, a pesar de estas pequeñas diferencias, el sistema garantiza una transmisión robusta y fiable de los parámetros, permitiendo que los ajustes sugeridos por el modelo de IA se reflejen de manera coherente en la interfaz de Reaper. Esto facilita un flujo de trabajo profesional y eficiente, en el que el usuario puede confiar en que los valores mostrados en los *plugins* corresponden a las recomendaciones generadas por el asistente.

5.6. Pruebas modelo de IA

La calidad y utilidad del asistente dependen en gran medida de la capacidad del modelo de IA para interpretar correctamente las indicaciones del usuario y mantener el contexto de la producción musical a lo largo de la sesión. Para validar este aspecto, se realizaron pruebas centradas en dos objetivos principales: evaluar la efectividad de distintos *prompts* para obtener respuestas adecuadas según el tipo de consulta, y comprobar que el modelo genera el formato JSON esperado con los valores correctos para funcionalidades clave.

Las pruebas consistieron en el uso de una variedad de *prompts*, tanto simples como complejos, que cubrían diferentes escenarios de uso real. Cada *prompt* fue evaluado según la precisión de la respuesta generada y la correcta estructuración de la información en el formato JSON requerido por el sistema. Además, se verificó que los valores asociados a cada parámetro fueran coherentes con las expectativas del usuario y las necesidades del proyecto.

Dado que la evaluación de la adecuación de las respuestas tiene un carácter principalmente subjetivo, se optó por un enfoque iterativo: los *prompts* se fueron mejorando progresivamente con cada prueba, ajustándose a la manera más adecuada de formular las consultas para obtener respuestas precisas y útiles. Este proceso permitió afinar la comunicación con el modelo de IA y optimizar la generación de respuestas.

Además, se probó específicamente, utilizando la funcionalidad de preguntar al asistente, que el modelo de IA mantenía el contexto de la creación musical constantemente. Durante las pruebas, se realizaron varias acciones utilizando distintas funcionalidades y, a continuación, se solicitó al modelo de IA que resumiera el contexto actual de la producción. El modelo demostró ser capaz de mantener y devolver correctamente el contexto acumulado, proporcionando un resumen coherente y detallado de las acciones realizadas hasta el momento. Esta capacidad es especialmente relevante, ya que garantiza que el asistente pueda ofrecer respuestas contextualizadas y útiles en cualquier punto del proceso creativo, manteniendo constantemente el contexto del proceso creativo.

Además, la precisión en la generación de respuestas en formato JSON fue especialmente relevante para las funcionalidades clave, donde el cumplimiento del formato y los valores correctos es esencial para la integración con el resto del sistema. Aunque no se realizó una medición estadística de la tasa de acierto en las respuestas del modelo de manera objetiva, la experiencia acumulada durante las pruebas confirmó que el modelo de IA seleccionado es robusto y fiable para interpretar las indicaciones del usuario y generar respuestas estructuradas y útiles.

6. Presupuesto

El presupuesto para este proyecto se calcula teniendo en cuenta los factores materiales, los componentes Software y las horas de trabajo del diseño y desarrollo del software, por lo tanto, se dividirá este punto en costes materiales y costes de personal, calculando finalmente el coste total del proyecto.

6.1. Costes materiales

Para el desarrollo del proyecto ha sido necesario un ordenador portátil sin requisitos mínimos, pero con las características de potencia y capacidad suficientes para poder realizar el desarrollo del software y utilizar la DAW para realizar todas las pruebas necesarias, además de ejecutar el asistente inteligente desarrollado y la conexión con la DAW con un rendimiento óptimo. Además, cabe destacar la necesidad de tener un micrófono utilizable y de calidad integrado en el ordenador para poder ejecutar los comandos de voz sin problema.

De este modo, se ha utilizado un ordenador portátil *HP Laptop 15* adquirido por un valor de 575€. Cabe destacar que todo el software utilizado para la realización del proyecto es de uso libre y por lo tanto gratuito. En la tabla 7 se puede observar el coste total de los costes materiales:

Concepto	Coste
HP Laptop 15	575,00€
Software libre	0,00€
TOTAL	575,00€

Tabla 7. Costes materiales

6.2. Costes de personal

Los costes de personal son una parte fundamental de cualquier proyecto. En este caso este tipo de costes se basan en el sueldo medio de un ingeniero de telecomunicaciones junior y en el número de horas necesarias para realizar este proyecto. Según la plataforma Glassdoor [56], el sueldo medio de este tipo de profesional es de 25.000€.

Esto significa el pago de 12,02€ por hora. Para este proyecto se han empleado 360 horas de trabajo. En la tabla 8 se puede observar el coste total de los costes de personal:

Concepto	Coste
Desarrollador software	$360 \text{ [h]} * 12,02 \text{ [€/h]} = 4.327,20\text{€}$
TOTAL	4.327,20€

Tabla 8. Costes de personal

6.3. Costes totales

A continuación, en la tabla 9 se muestra el coste total del proyecto calculado a partir de los costes materiales y de personal:

Concepto	Coste
Gastos materiales	575,00€
Gastos de personal	4.327,20€
TOTAL	4.902,20€

Tabla 9. Costes totales

7. Impacto del proyecto

La evaluación del impacto de un proyecto constituye una parte fundamental en cualquier trabajo de desarrollo tecnológico, ya que permite analizar de forma sistemática las repercusiones que la solución propuesta puede tener en su entorno. Este análisis no solo abarca el posible impacto a nivel técnico del sistema, sino que también considera sus implicaciones sociales, económicas, medioambientales y educativas, así como su contribución a los Objetivos de Desarrollo Sostenible definidos por Naciones Unidas.

En el contexto actual de transformación digital y democratización del acceso a la tecnología, resulta especialmente relevante valorar cómo proyectos como este pueden influir en la industria musical, en los colectivos de usuarios y en la sociedad en general. La identificación y análisis de estos impactos proporciona una visión global del alcance del proyecto y orienta posibles mejoras futuras, asegurando que la innovación tecnológica se traduzca en beneficios tangibles y sostenibles para la comunidad.

7.1. Contexto general del proyecto

La industria musical ha experimentado en las últimas décadas una profunda transformación impulsada por la digitalización y la integración de tecnologías avanzadas como la inteligencia artificial. Estas innovaciones han modificado no solo los procesos de creación y producción musical, sino también la manera en que los usuarios acceden y participan en la generación de contenido musical. En este contexto, la producción musical doméstica se ha convertido en un fenómeno creciente, facilitado por la disponibilidad de software y hardware, más accesibles tanto a nivel económico como a nivel de facilidad de acceso, que permiten a músicos amateurs y profesionales trabajar desde sus propios espacios con recursos limitados.

Sin embargo, a pesar de estos avances, siguen existiendo barreras técnicas y económicas que dificultan el acceso pleno a herramientas de producción musical, especialmente para usuarios sin formación especializada o con recursos limitados. La complejidad de las DAW, la necesidad de conocimientos técnicos para manejar instrumentos virtuales y efectos, y el coste asociado a equipos y software profesional son factores que limitan la democratización de la creación musical. Este proyecto se enmarca en esta realidad, proponiendo una solución que simplifica y automatiza el proceso creativo mediante la interpretación de comandos de voz y la integración de IA.

Esta herramienta está diseñada para beneficiar a diversos colectivos que, por distintas razones, encuentran dificultades en el acceso y manejo de las herramientas tradicionales de producción musical. Entre estos grupos se encuentran los músicos amateurs y autodidactas, quienes, aunque poseen interés y creatividad musical, carecen de formación técnica avanzada y buscan una forma intuitiva y accesible para plasmar sus ideas sin necesidad de dominar interfaces complejas o programación MIDI. Asimismo, estudiantes y docentes de música pueden aprovechar el asistente como un recurso didáctico que facilita el aprendizaje práctico y la experimentación creativa en entornos educativos. Por otro lado, pequeños productores y

creadores independientes, que a menudo deben optimizar tiempos y recursos, pueden beneficiarse de la automatización de tareas técnicas para centrarse en los aspectos artísticos de su trabajo. Finalmente, el sistema ofrece una oportunidad significativa para personas con discapacidad o limitaciones físicas, quienes pueden superar barreras de accesibilidad gracias a la interacción mediante comandos de voz, haciendo la producción musical más inclusiva.

En definitiva, el proyecto responde a una necesidad creciente de accesibilidad, inclusión y eficiencia en la producción musical, contribuyendo a ampliar el espectro de personas que pueden participar activamente en la creación musical y a fomentar la innovación en un sector en constante evolución.

7.2. Identificación del impacto

El desarrollo de este proyecto implica una serie de impactos que van mucho más allá de la mera innovación tecnológica y afectan de manera significativa a distintos ámbitos de la sociedad.

En el plano social, el proyecto representa una oportunidad real para la democratización de la creación musical, al facilitar el acceso a herramientas avanzadas a personas que, por falta de conocimientos técnicos o recursos, tradicionalmente quedaban excluidas de estos procesos. Esta accesibilidad puede contribuir a reducir la brecha digital y a fomentar la igualdad de oportunidades, especialmente en colectivos vulnerables o con necesidades específicas de accesibilidad. Sin embargo, también es necesario reflexionar sobre el riesgo de una excesiva dependencia tecnológica, que podría limitar el desarrollo de habilidades musicales tradicionales y la creatividad espontánea, así como sobre la posible exclusión de quienes, por diferentes motivos, no puedan acceder a las nuevas tecnologías.

Desde la perspectiva económica, la automatización de tareas y la reducción de costes asociados a la producción musical pueden favorecer el emprendimiento y el autoempleo, permitiendo que pequeños creadores y productores independientes accedan a un mercado cada vez más competitivo. Esto puede suponer una dinamización del sector y la aparición de nuevos modelos de negocio. No obstante, la introducción de sistemas inteligentes también puede afectar a ciertos perfiles profesionales tradicionales, generando la necesidad de adaptación y actualización constante de competencias en el sector musical y tecnológico.

En el ámbito tecnológico e industrial, el proyecto impulsa la innovación y la transformación digital en la música, abriendo nuevas posibilidades de colaboración entre humanos y máquinas. La integración de inteligencia artificial en la creatividad musical plantea retos importantes en cuanto a la transparencia y la gestión de posibles sesgos en la generación de contenido. Como desarrolladores, es fundamental asumir la responsabilidad de diseñar sistemas que sean comprensibles, justos y accesibles para todos los usuarios.

En cuanto al impacto ambiental, la digitalización de la producción musical contribuye a reducir el consumo de materiales físicos y los desplazamientos asociados a la producción en estudios tradicionales, lo que supone un avance hacia prácticas más sostenibles. Sin embargo, el uso

intensivo de dispositivos electrónicos y servidores implica un consumo energético relevante, por lo que la eficiencia en el uso de recursos y la concienciación sobre el ciclo de vida de la tecnología empleada son aspectos clave para minimizar la huella ecológica del proyecto.

Por último, desde el punto de vista ético y legal, el manejo de datos de voz y la interacción personalizada exigen un compromiso firme con la protección de la privacidad y el cumplimiento de la normativa vigente. Además, la automatización de la creatividad plantea cuestiones sobre la autoría y la originalidad de las obras generadas, que deben ser abordadas con transparencia y rigor profesional, garantizando siempre el respeto a los derechos de los usuarios y creadores.

En conclusión, aunque es imprescindible reconocer y gestionar los posibles riesgos y retos asociados al uso de inteligencia artificial en la producción musical, el balance global del impacto de este proyecto es claramente positivo. Los beneficios en términos de accesibilidad, democratización, impulso a la creatividad y sostenibilidad superan a los posibles efectos adversos, siempre que se mantenga una actitud crítica y responsable en el desarrollo y aplicación de la tecnología. Así, el proyecto contribuye de manera significativa al bienestar colectivo y al avance hacia una sociedad más inclusiva, innovadora y sostenible [57].

7.3. Conclusión sobre el impacto del proyecto

Tras analizar los diferentes impactos del proyecto, se puede afirmar que ofrece un balance claramente positivo. Aunque existen desafíos y aspectos a considerar, los beneficios en términos de accesibilidad, innovación y sostenibilidad superan ampliamente los posibles riesgos. Este proyecto responde a una necesidad creciente de democratización tecnológica y fomenta la inclusión, la creatividad y el desarrollo económico en un sector en constante evolución.

Este enfoque se alinea claramente con varios ODS (Objetivos de Desarrollo Sostenible) de Naciones Unidas [58], entre los que destacan:

- **ODS 4. Educación de calidad:** El proyecto facilita el acceso a recursos educativos musicales y promueve el aprendizaje autónomo y creativo, tanto en entornos formales como informales. Al simplificar el proceso de producción musical y hacerlo más accesible, contribuye a que más personas puedan desarrollar competencias artísticas y tecnológicas, independientemente de su nivel previo o situación socioeconómica.
- **ODS 8. Trabajo decente y crecimiento económico:** Al reducir barreras de entrada y automatizar tareas técnicas, el proyecto impulsa el emprendimiento y la autoproducción musical, permitiendo que pequeños creadores y profesionales independientes puedan acceder a nuevas oportunidades laborales y de negocio. Esto favorece el crecimiento económico en el sector creativo y tecnológico, y apoya la generación de empleo de calidad en un mercado en transformación.

- **ODS 9. Industria, innovación e infraestructura:** La integración de inteligencia artificial y control por voz en la producción musical representa un avance en la modernización de la industria cultural. El proyecto promueve la adopción de tecnologías innovadoras y accesibles, contribuyendo al desarrollo de infraestructuras digitales más inclusivas y a la creación de nuevas soluciones que potencian la creatividad y la colaboración.
- **ODS 10. Reducción de las desigualdades:** Al poner herramientas avanzadas al alcance de colectivos tradicionalmente excluidos por motivos económicos, formativos o de accesibilidad, el asistente contribuye a disminuir las desigualdades en el acceso a la tecnología y a la cultura. Esto permite que un mayor número de personas pueda participar activamente en la creación musical y beneficiarse de las oportunidades que ofrece la digitalización.
- **ODS 12. Producción y consumo responsables:** El proyecto fomenta prácticas sostenibles al reducir la necesidad de materiales físicos y desplazamientos, apostando por la digitalización y la eficiencia en el uso de recursos. Además, promueve la concienciación sobre el impacto ambiental de la tecnología y la importancia de un desarrollo digital responsable.

En definitiva, el presente proyecto pone de manifiesto cómo la tecnología puede ser un motor de cambio para lograr una sociedad más equitativa, creativa y sostenible. Gracias a un desarrollo responsable y a la atención a las necesidades reales de los usuarios, la solución propuesta contribuye a ampliar el acceso a la cultura y la educación, fomenta la innovación en el ámbito musical y promueve prácticas más inclusivas y respetuosas con el entorno. Así, se consolida como una aportación relevante para avanzar hacia los objetivos de desarrollo sostenible y el bienestar colectivo.

8. Conclusiones

8.1. Conclusiones generales

Tras la realización del presente proyecto fin de grado, se puede concluir que la solución propuesta responde de manera efectiva al reto planteado inicialmente: facilitar la creación musical mediante un asistente controlado por voz, eliminando barreras técnicas y permitiendo el acceso a la producción musical a usuarios sin experiencia previa o con recursos limitados. A lo largo del desarrollo, se ha conseguido que el sistema reconozca e interprete comandos de voz de forma precisa, permitiendo la interacción natural con el entorno de producción musical y simplificando tareas habitualmente complejas, además de guiar de manera inteligente al usuario hacia una creación y producción musical de calidad, consiguiendo que las ideas del usuario se plasmen en la DAW de una manera rápida y sencilla.

La solución planteada se basa en la integración de tecnologías de reconocimiento de voz y procesamiento de lenguaje natural, utilizando Vosk y Wit.ai, además de la conversión automática de audio a información musical editable, como la transformación de tarareos en datos MIDI. El asistente interpreta instrucciones habladas, genera respuestas contextuales y permite la edición y manipulación de parámetros musicales en tiempo real. Además, se ha implementado la comunicación directa con la DAW Reaper, permitiendo la transferencia y ajuste automático de parámetros de efectos y pistas, lo que facilita el flujo de trabajo creativo incluso para usuarios sin conocimientos avanzados.

Un aspecto fundamental del diseño del sistema es su arquitectura modular, que divide el asistente en componentes independientes y especializados, cada uno encargado de una función concreta. Esta modularidad facilita la escalabilidad, el mantenimiento y la incorporación de nuevas funcionalidades, permitiendo además la sustitución o mejora de módulos específicos sin afectar al conjunto. Gracias a esta estructura, el sistema puede adaptarse fácilmente a futuras actualizaciones tecnológicas o a la integración con otros entornos y plataformas.

Otro aspecto fundamental es el modelo de inteligencia artificial que se utiliza y es que el sistema utiliza un modelo general de lenguaje, **deepseek-chat-v3-0324**, al que se accede mediante la API de OpenRouter, lo que permite una integración sencilla y flexible en el flujo del asistente. Aunque este modelo no ha sido entrenado específicamente en producción musical, la generación de *prompts* cuidadosamente diseñados y adaptados al contexto musical por parte del desarrollador consigue que las respuestas proporcionadas sean relevantes y útiles para las necesidades concretas del usuario durante el proceso creativo y que tenga el contexto suficiente para responder correctamente cuestiones referentes a la producción musical propia del usuario o incluso más general. De este modo, se compensa la falta de especialización del modelo con una contextualización precisa en cada interacción.

Durante la fase de desarrollo, se llevaron a cabo diversas pruebas para garantizar la fiabilidad, precisión y funcionalidad del asistente. En primer lugar, se realizaron pruebas unitarias y de

integración de forma manual, utilizando técnicas de depuración y registros detallados que permitieron verificar el correcto funcionamiento de cada módulo y la comunicación entre ellos. Esta metodología facilitó la detección y corrección de errores lógicos y de implementación, asegurando la estabilidad del sistema.

Posteriormente, se llevaron a cabo pruebas funcionales de caja negra con usuarios externos, quienes evaluaron distintos casos de uso para comprobar que el sistema respondía adecuadamente a las distintas órdenes y situaciones previstas. Este enfoque permitió contrastar el comportamiento real con el esperado, identificando áreas de mejora en la interacción y usabilidad.

En cuanto al reconocimiento de voz, se evaluó la precisión de los comandos utilizando tanto Vosk como la combinación de Vosk y Wit.ai. Se realizaron múltiples pruebas con listas de comandos típicos, midiendo la tasa de aciertos y errores tanto a nivel de palabras como de frases completas. Los resultados mostraron que la integración de ambos motores disminuye significativamente el porcentaje de fallos, mejorando la experiencia del usuario.

Además, se probaron exhaustivamente los mecanismos de comunicación e integración entre el asistente y el DAW Reaper. Se verificó la correcta conversión y transmisión de parámetros generados por la inteligencia artificial a los efectos y controles del software, mediante tablas que reflejaron un alto porcentaje de coincidencia entre los valores esperados y los aplicados en tiempo real.

Por último, se realizaron pruebas sobre el modelo de inteligencia artificial, experimentando con diferentes tipos de *prompts* para optimizar la calidad y coherencia de las respuestas generadas, así como para asegurar la correcta estructuración de datos en formatos JSON, fundamentales para funcionalidades avanzadas del asistente.

Estas pruebas combinadas han sido clave para validar la robustez y eficacia del sistema, confirmando que la solución propuesta cumple con los objetivos planteados y ofrece una experiencia accesible y fiable para el usuario. Durante el desarrollo y la validación se han identificado nuevos retos, como la necesidad de adaptar la interfaz a diferentes niveles de usuario, mejorar la personalización, adaptar el asistente a otros entornos de producción o añadir nuevas funcionalidades que le den un valor añadido al asistente. Estos hallazgos amplían el conocimiento sobre los retos y oportunidades que plantea la integración de tecnologías digitales en la producción musical.

Aunque el proyecto ha alcanzado sus objetivos principales y aporta una solución efectiva, el problema tecnológico no queda completamente cerrado. Por ello, se abren diversas líneas futuras de trabajo para seguir mejorando la experiencia del usuario.

8.2. Trabajos futuros

Aunque el asistente desarrollado cumple los objetivos principales del proyecto, existen múltiples líneas de mejora y expansión que pueden abordarse en el futuro para potenciar su funcionalidad y aumentar su alcance. Algunas de ellas son las siguientes:

- **Compatibilidad con más DAWs:** Una de las prioridades será ampliar la integración del asistente con otras estaciones de trabajo de audio digital populares, como FL Studio o Ableton Live, permitiendo que más usuarios puedan beneficiarse de la herramienta en nuevos entornos de trabajo.
- **Persistencia y gestión avanzada de datos de usuario:** Se propone implementar una base de datos que permita guardar y recuperar la configuración, preferencias y progresos del usuario entre distintas sesiones, facilitando una experiencia personalizada y continua.
- **Entrenamiento y especialización del modelo de IA propio:** En el futuro, sería interesante entrenar modelos de inteligencia artificial específicamente orientados a la producción musical y al caso propio del asistente, para mejorar la precisión y profundidad de las respuestas y recomendaciones ofrecidas por el asistente.
- **Mejoras en la interfaz gráfica:** Se plantea el desarrollo de una interfaz gráfica más intuitiva y atractiva, que incluya elementos visuales avanzados como gráficos en tiempo real que representen las notas, diferencias de volumen y otros parámetros musicales mientras el usuario graba el tarareo o introduce melodías. Esto facilitaría la comprensión y el control del proceso creativo, inspirándose en herramientas de visualización musical ampliamente utilizadas.
- **Nuevas funcionalidades creativas relacionadas con la producción de voz:** Entre las posibles ampliaciones, destaca la opción de incorporar la voz del usuario como elemento creativo dentro de la producción, no solo como referencia melódica sino también como pista vocal editable y procesable. Además, se podrían añadir herramientas para la edición avanzada de voz, automatización de parámetros y generación de acompañamientos o arreglos automáticos para la propia voz como tal, a parte de las que hay actualmente que están más centradas en la melodía.
- **Nueva funcionalidad para la creación de patrones de percusión y baterías:** Se plantea desarrollar un módulo específico que asista al usuario en la generación de ritmos de batería y percusión mediante inteligencia artificial. Este módulo permitiría, crear patrones rítmicos personalizados adaptados al estilo del proyecto, seleccionando instrumentos de percusión, ajustando la complejidad y la dinámica. Además, podría ofrecer la posibilidad de visualizar los patrones generados en la interfaz gráfica y editarlos en tiempo real, facilitando tanto la experimentación como el aprendizaje de estructuras rítmicas. La integración de esta funcionalidad ampliaría notablemente las capacidades creativas del sistema, permitiendo que el usuario no solo genere

melodías, sino también bases rítmicas originales y adaptadas a sus necesidades musicales.

- **Personalización avanzada y adaptación al estado de ánimo:** Desarrollar sistemas que adapten las sugerencias y el flujo de trabajo no solo a los gustos musicales, sino también al estado emocional del usuario, detectado mediante análisis de voz o patrones de uso.
- **Integración de voz sintética y clonación vocal:** Permitir que el usuario genere voces sintéticas o clones de su propia voz para experimentar con líneas vocales, coros o efectos.
- **Integración con servicios en la nube:** Permitir la sincronización de proyectos, configuraciones y archivos entre dispositivos mediante almacenamiento en la nube, facilitando el trabajo colaborativo y la portabilidad.
- **Refuerzo de la accesibilidad:** Explorar nuevas formas de adaptar el asistente a usuarios con diferentes perfiles, necesidades o discapacidades, así como ampliar las opciones de personalización para ajustarse a distintos estilos y flujos de trabajo.

Estas líneas de trabajo permitirán que el asistente evolucione hacia una herramienta aún más potente, versátil y accesible, consolidando su utilidad para la creación musical amateur y dándole valor añadido al proyecto.

9. Referencias

- [1] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed., London, U.K.: Pearson, 2023.
- [2] M. B. Hoy, "Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants," *Medical Reference Services Quarterly*, vol. 37, no. 1, pp. 81-88, Jan 2018.
- [3] Z. Zhang, J. Cui, and Y. Fu, "Recent advances and challenges in intelligent virtual assistants," *arXiv preprint, arXiv:2004.03586*, 2020 [Online]. Available: <https://arxiv.org/abs/2004.03586>. [Accessed: 3 Sep. 2024].
- [4] IBM, "¿Qué es el reconocimiento de voz?," 2025. [Online]. Available: <https://www.ibm.com/es-es/topics/speech-recognition>. [Accessed 20 Sep 2024].
- [5] H. S. Rufiner et al., "Sistema de reconocimiento automático del habla," *Redalyc*, pp. 151-177, 2004.
- [6] Vosk, "Vosk API Documentation," 2025. [Online]. Available: <https://alphacephei.com/vosk/>. [Accessed 20 Sep 2024].
- [7] Wit.ai, "Docs: Natural Language for Developers," 2025. [Online]. Available: <https://wit.ai/docs/>. [Accessed 20 Sep 2024].
- [8] Mozilla Developer Network, "Modelo-Vista-Controlador (MVC)," 2023. [Online]. Available: <https://developer.mozilla.org/es/docs/Glossary/MVC>. [Accessed 22 Sep 2024].
- [9] GeeksforGeeks, "MVC Architecture [Online image]," 4 Jul 2024. [Online]. Available: <https://media.geeksforgeeks.org/wp-content/uploads/20240704102850/MVC-Architecture.webp>. [Accessed 22 Sep 2024].
- [10] D. L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053-1058, 1972.
- [11] Python Software Foundation, "The Python Tutorial — Python 3.13.5 documentation," 2025. [Online]. Available: <https://docs.python.org/3/tutorial/>. [Accessed 2 Oct 2024].
- [12] R. Ierusalimschy, *Programming in Lua*, 3rd ed., Boston, MA, USA: Addison-Wesley, 2016.
- [13] R. Computing, "PyQt5 Reference Guide," 2025. [Online]. Available: <https://www.riverbankcomputing.com/static/Docs/PyQt5/>. [Accessed 2 Oct 2024].

- [14] M. Blech, "sounddevice — Python module for audio playback and recording," 2025. [Online]. Available: <https://python-sounddevice.readthedocs.io/>. [Accessed 3 Oct 2024].
- [15] K. Reitz et al., "Requests: HTTP for Humans," 2025. [Online]. Available: <https://docs.python-requests.org/en/latest/>. [Accessed 3 Oct 2024].
- [16] T. Oliphant et al., "NumPy Documentation," 2025. [Online]. Available: <https://numpy.org/doc/stable/>. [Accessed 3 Oct 2024].
- [17] P. Brossier, "aubio - a library for audio and music analysis," 2025. [Online]. Available: <https://aubio.org/manual/latest/>. [Accessed 6 Oct 2024].
- [18] O. Larkin, "mido: MIDI Objects for Python," 2025. [Online]. Available: <https://mido.readthedocs.io/en/latest/>. [Accessed 6 Oct 2024].
- [19] B. McFee et al., "librosa: Audio and Music Signal Analysis in Python," 2025. [Online]. Available: <https://librosa.org/doc/latest/index.html>. [Accessed 7 Oct 2024].
- [20] M. Cuthbert et al., "music21: A Toolkit for Computer-Aided Musicology," 2025. [Online]. Available: <https://web.mit.edu/music21/doc/>. [Accessed 7 Oct 2024].
- [21] Microsoft, "Visual Studio Code Documentation," 2025. [Online]. Available: <https://code.visualstudio.com/docs>. [Accessed 7 Oct 2024].
- [22] Mixmasterworld, "Comparativa de DAWs: Niveles de dificultad para usuarios principiantes, intermedios y avanzados," 2024. [Online]. Available: <https://mixmasterworld.com/tabla-comparativa-de-daws-niveles-de-dificultad-para-usuarios-principiantes-intermedios-y-avanzados/>. [Accessed 20 Oct 2024].
- [23] Image-Line, "FL Studio Online Manual," 2025. [Online]. Available: <https://www.image-line.com/fl-studio-learning/fl-studio-online-manual>. [Accessed 20 Oct 2024].
- [24] Ableton AG, "Ableton Live - Features Overview," 2025. [Online]. Available: <https://www.ableton.com/en/live/>. [Accessed 20 Oct 2024].
- [25] Cockos Inc., "REAPER | Audio Production Without Limits," 2025. [Online]. Available: <https://www.reaper.fm>. [Accessed 20 Oct 2024].
- [26] J. M. Llorens, "Introducción al protocolo de comunicación Open Sound Control," Mosaic, Universitat Oberta de Catalunya, 2019. [Online]. Available: <https://mosaic.uoc.edu/2019/04/03/introduccion-al-protocolo-de-comunicacion-open-sound-control/>. [Accessed 3 Nov 2024].

- [27] Universidad Nacional de Quilmes, "El protocolo MIDI," 2020. [Online]. Available: <https://static.uvq.edu.ar/mdm/CPS/unidad-09-02.html>. [Accessed 3 Nov 2024].
- [28] Open Sound Control, "Open Sound Control (OSC) Protocol," 2025. [Online]. Available: <https://opensoundcontrol.stanford.edu/>. [Accessed 3 Nov 2024].
- [29] Cockos Inc., "REAPER Cockos Effects Summary Guide," 2021. [Online]. Available: <https://www.reaper.fm/userguide/REAPEREffectsGuide2021.pdf>. [Accessed 3 Nov 2024].
- [30] DSK Music, "DSK Virtuoso," 2025. [Online]. Available: <https://www.dskmusic.com/dsk-virtuoso/>. [Accessed 6 Nov 2024].
- [31] Cockos Inc., "ReaScript - REAPER," 2025. [Online]. Available: <https://www.reaper.fm/sdk/reascript/reascript.php>. [Accessed 9 Nov 2024].
- [32] ExtremRaym, "ReaScript API — Documentation," 2025. [Online]. Available: <https://www.extremraym.com/cloud/reascript-doc/>. [Accessed 9 Nov 2025].
- [33] Automation Anywhere, "¿Qué es el software de automatización?," 2025. [Online]. Available: <https://www.automationanywhere.com/la/rpa/automation-software>. [Accessed 9 Nov 2024].
- [34] Google, "Magenta: Music and Art Generation with Machine Intelligence," 2025. [Online]. Available: <https://magenta.withgoogle.com/>. [Accessed 21 Nov 2024].
- [35] J.-P. Briot, G. Hadjeres, and F. Pachet, "Deep Learning Techniques for Music Generation – A Survey," in *Computational Creativity Research: Towards Creative Machines*, T. Veale and F. Cardoso, Eds. Cham, Switzerland: Springer, 2017, pp. 267–310.
- [36] OpenAI, "MuseNet," 2019. [Online]. Available: <https://openai.com/index/musenet/>. [Accessed 21 Nov 2024].
- [37] OpenAI, "Jukebox: A Generative Model for Music," 2020. [Online]. Available: <https://openai.com/index/jukebox/>. [Accessed 21 Nov 2024].
- [38] A. Agostinelli, T. I. Denk, Z. Borsos, J. Engel, M. Verzett, A. Caillon, Q. Huang, A. Jansen, A. Roberts, M. Tagliasacchi, M. Sharifi, N. Zeghidour, C. Frank, "MusicLM: Generating Music From Text," arXiv preprint arXiv:2301.11325, 2023.
- [39] H. Forsgren and S. Martiros, "Riffusion: Stable diffusion for real-time music generation," GitHub, 2022. [Online]. Available: <https://github.com/riffusion/riffusion-hobby>. [Accessed 22 Nov 2024].

- [40] Hugging Face, "riffusion-model-v1," 2023. [Online]. Available: <https://huggingface.co/riffusion/riffusion-model-v1>. [Accessed 22 Nov 2024].
- [41] C. Hawthorne, E. Elsen, J. Song et al, "Onsets and Frames: Dual-Objective Piano Transcription," in Proc. ISMIR, Paris, France, Sep. 2018, pp. 50-57.
- [42] J. W. Kim, J. Salamon, P. Li, and J. P. Bello, "CREPE: A Convolutional Representation for Pitch Estimation," in Proc. ICASSP, Calgary, Canada, Apr. 2018, pp. 161–165.
- [43] OpenRouter, "OpenRouter," 2025. [Online]. Available: <https://openrouter.ai>. [Accessed 25 Nov 2024].
- [44] Apidog, "Los mejores modelos de IA gratuitos que puedes usar en OpenRouter," 2025. [Online]. Available: <https://apidog.com/es/blog/free-ai-models-2/>. [Accessed 25 Nov 2024].
- [45] OpenRouter, "DeepSeek V3 0324 (free) – Run with an API," 2025. [Online]. Available: <https://openrouter.ai/deepseek/deepseek-chat-v3-0324:free/api>. [Accessed 29 Nov 2024].
- [46] DeepSeek, "DeepSeek R1," 2024. [Online]. Available: <https://www.deepseek.com/>. [Accessed 29 Nov 2024].
- [47] Google, "Gemini 2.0," 2025. [Online]. Available: <https://ai.google/discover/gemini/>. [Accessed 29 Nov 2024].
- [48] Google, "Gemma: Open models built from the same research and technology used to create Gemini," 2024. [Online]. Available: <https://ai.google.dev/gemma>. [Accessed 29 Nov 2024].
- [49] Scout, "Scout LLM 109B," 2025. [Online]. Available: <https://scout.llm.agency/>. [Accessed 29 Nov 2024].
- [50] Riverbank Computing, "QTextEdit Class Reference," PyQt5 Documentation, 2025. [Online]. Available: <https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QTextEdit.html>. [Accessed 10 Dec 2024].
- [51] Riverbank Computing, "QLineEdit Class Reference," PyQt5 Documentation, 2025. [Online]. Available: <https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QLineEdit.html>. [Accessed 10 Dec 2024].
- [52] Riverbank Computing, "QPushButton Class Reference," PyQt5 Documentation, 2025. [Online]. Available: <https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QPushButton.html>. [Accessed 10 Dec 2024].

- [53] BCCN SOFT, "Support for Signals and Slots," PyQt5 Documentation, 2025. [Online]. Available: https://doc.bccnsoft.com/docs/PyQt5/signals_slots.html. [Accessed 10 Dec 2024].
- [54] Carmatec, "Python Logging Best Practices: Complete Guide 2025," 2025. [Online]. Available: <https://www.carmatec.com/blog/python-logging-best-practices-complete-guide/>. [Accessed 10 Dec 2024].
- [55] Bright Security, "Black Box Testing: Types, Techniques, Pros and Cons," 2025. [Online]. Available: <https://www.brightsec.com/blog/black-box-testing-types-techniques-pros-and-cons/>. [Accessed 20 May 2025].
- [56] Glassdoor, "Sueldo: Telecomunicaciones Junior," 2025. [Online]. Available: https://www.glassdoor.es/Sueldos/telecomunicaciones-junior-sueldo-SRCH_KO0,25.htm. [Accessed 28 May 2025].
- [57] C. Fernández Aller and R. Miñano, "Guía para trabajar la Responsabilidad Social y Ambiental (GRSA)," ETSI Sistemas Informáticos, UPM, May 2015. [Online]. Available: https://oa.upm.es/35542/1/Guia_Responsabilidad_Social_y_Ambiental-V2-1.pdf. [Accessed 28 May 2025].
- [58] Naciones Unidas, "Objetivos de Desarrollo Sostenible," 2015. [Online]. Available: <https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/>. [Accessed 28 May 2025].

Anexo

A.1. Manual de usuario

El presente manual está diseñado para guiar al usuario en la instalación y el uso del asistente para producción musical. Su objetivo es facilitar la comprensión del sistema y permitir que cualquier usuario, independientemente de su nivel de experiencia, pueda aprovechar al máximo las funcionalidades ofrecidas.

El presente manual se divide en dos grandes apartados:

- **Manual de Instalación:** Explica los requisitos previos, la instalación y la configuración necesaria para que el asistente funcione correctamente.
- **Manual de Uso:** Describe paso a paso cómo interactuar con el asistente, cómo realizar las distintas acciones y acceder a todas las funcionalidades disponibles en el asistente.

Cada sección incluye instrucciones claras y recomendaciones prácticas para garantizar una experiencia fluida y satisfactoria.

A.1.1. Manual de instalación

Para garantizar el correcto funcionamiento del asistente musical, es necesario seguir una serie de pasos y asegurarse de cumplir con los requisitos previos. Este apartado detalla de manera clara y ordenada todo lo que el usuario debe tener instalado y configurado antes de comenzar a utilizar el sistema.

A continuación, se muestran los requisitos y pasos previos necesarios para el uso del asistente:

El asistente requiere la instalación de Python, preferiblemente en su versión 3.12.5 o superior, así como la presencia del DAW Reaper en el sistema. Además, es fundamental mantener la estructura original de carpetas y archivos del proyecto, ya que el código está diseñado para funcionar bajo un esquema de organización específico. En cuanto a los *plugins*, ReaEQ, ReaComp, ReaVerbate, Stereo Enhancer y ReaSynth vienen incluidos por defecto en Reaper y no requieren instalación adicional. Sin embargo, será necesario instalar el *plugin* DSK Virtuoso siguiendo las instrucciones del fabricante.

Para que la comunicación entre el asistente y Reaper sea posible, es imprescindible disponer de los scripts Lua desarrollados para el proyecto, los cuales deben estar ubicados en la carpeta correspondiente dentro de Reaper para su reconocimiento como acciones de Reaper, generalmente en la carpeta REAPER/Scripts. Asimismo, la configuración de la conexión OSC en Reaper es un paso clave y debe realizarse de la siguiente manera desde el propio Reaper:

- Acceder a Options -> Preferences -> Control/OSC/web.
- Añadir la opción OSC en el parámetro Control Surface Mode.

- Establecer la opción Default en el parámetro Pattern config.
- Establecer el puerto 8000 y el modo Local port.

En la figura 30 se muestra cómo debería quedar esta configuración en Reaper:

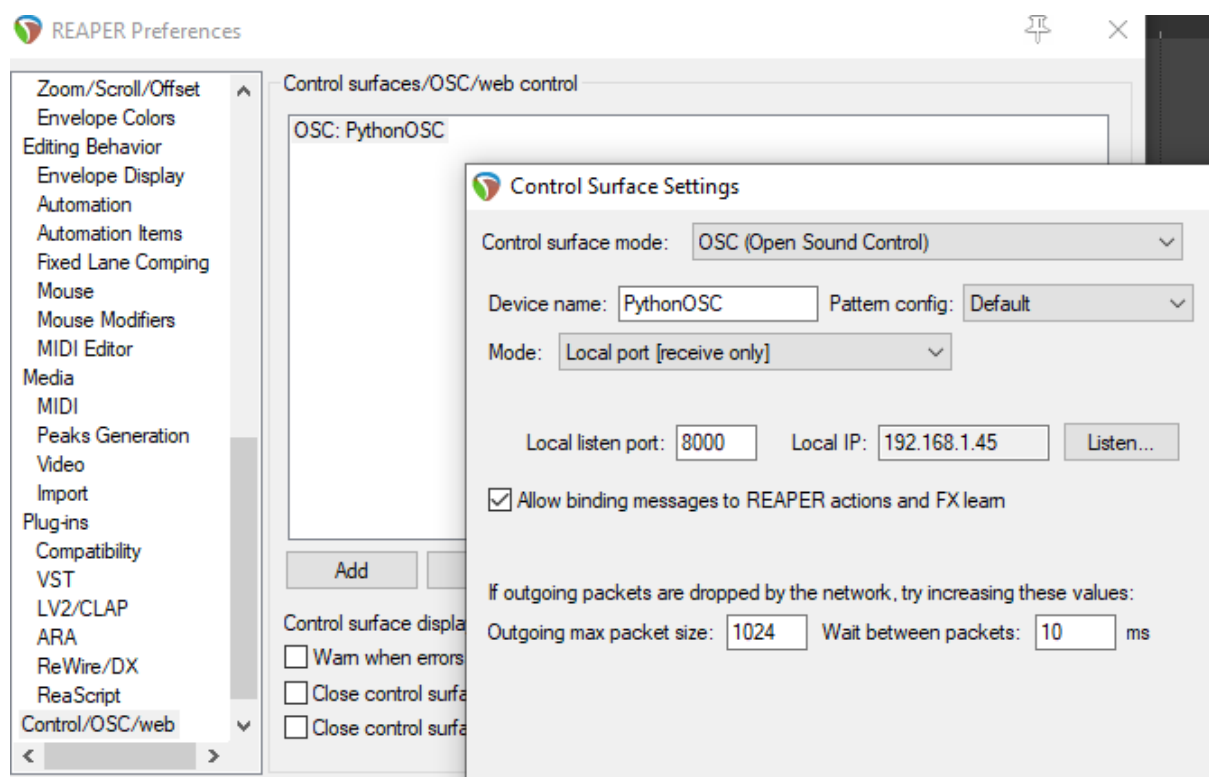


Figura 30. Configuración OSC en Reaper

Por último, deben instalarse las librerías de Python necesarias ejecutando el siguiente comando en la terminal:

>pip install PyQt5 sounddevice aubio numpy mido librosa openai python-osc pyautogui requests vosk music21

Una vez completados estos pasos, se recomienda abrir Visual Studio Code y ejecutar el código del asistente para verificar que todo funciona correctamente. Es importante comprobar que Reaper detecta los *plugins* y scripts, y que la comunicación OSC se establece sin problemas.

A.1.2. Manual de uso

Este manual tiene el objetivo de guiar al usuario en el uso práctico del asistente. El objetivo es que el usuario pueda aprovechar todas sus funcionalidades de forma sencilla, independientemente de su experiencia previa en producción musical. Se incluyen explicaciones generales, ejemplos de comandos y elementos visuales para facilitar la comprensión.

Para utilizar el asistente, el usuario debe ejecutar el script principal, `main.py`, desde un editor de código o desde la terminal.

Al iniciar el asistente, se muestra la interfaz principal, desde la cual es posible introducir comandos utilizando el reconocimiento de voz. La figura 31 muestra la interfaz gráfica una vez iniciado el programa.

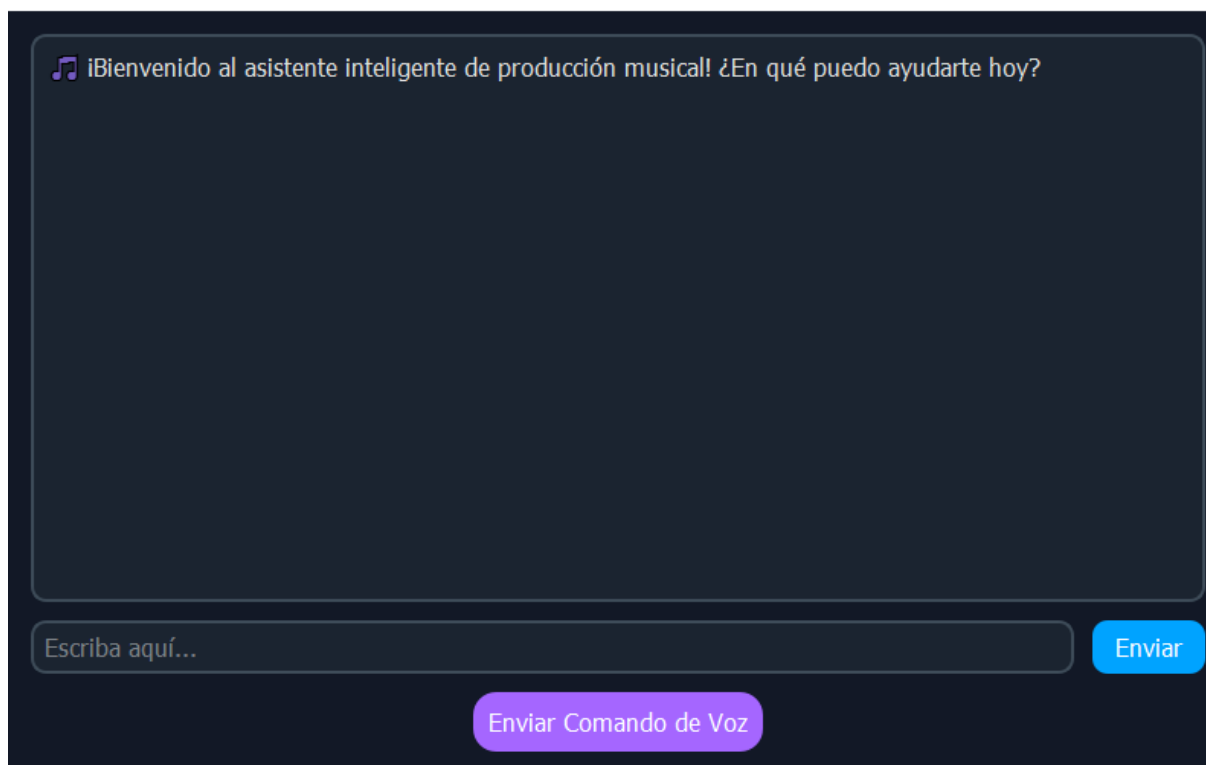


Figura 31. Pantalla de inicio del asistente

El asistente ofrece diversas funcionalidades para la creación y edición musical. A continuación, en la tabla 10, se presentan las principales opciones y un ejemplo concreto de comando para invocar cada una de ellas.

Nº	Funcionalidad	Ejemplo de comando
1	Grabar tarareos y abrir MIDI	Quiero grabar un tarareo y abrirlo con Reaper
2	Grabar tarareos y guardar MIDI	Graba un tarareo y guárdalo en mi carpeta
3	Abrir MIDI en Reaper	Quiero abrir un MIDI en Reaper
4	Cambio de instrumento	Cambia el instrumento de mi pista a violín
5	Aplicar efectos FX	Mejora mi melodía con efectos
6	Consulta de mejora	¿Como puedo mejorar mi creación musical?
7	Pregunta específica	Quiero hacerte una pregunta
8	Generar línea de bajo	Genera una línea de bajo basada en esta melodía
9	Añadir sintetizador	Añade un sintetizador a la melodía

Tabla 10. Funcionalidades y ejemplos de comandos

Cada comando puede adaptarse al flujo de trabajo del usuario. Tras introducirlo, el asistente ejecuta la acción y muestra el resultado en Reaper y en la propia interfaz, guiando al usuario, de una manera sencilla y clara, en la producción musical.

A continuación, se explica de forma detallada cómo debe proceder el usuario para utilizar cada una de las funcionalidades principales del asistente. Para cada opción, se describen los pasos a seguir y se incluyen capturas del uso real del asistente. De este modo, el usuario podrá comprender el flujo de trabajo en cada caso y resolver cualquier duda o incidencia que pueda surgir durante el uso de la aplicación.

Para comenzar a utilizar el asistente el usuario debe presionar el botón *Enviar Comando de Voz* y realizar el comando de voz. La figura 32 muestra la pantalla del asistente durante esta captura del comando:

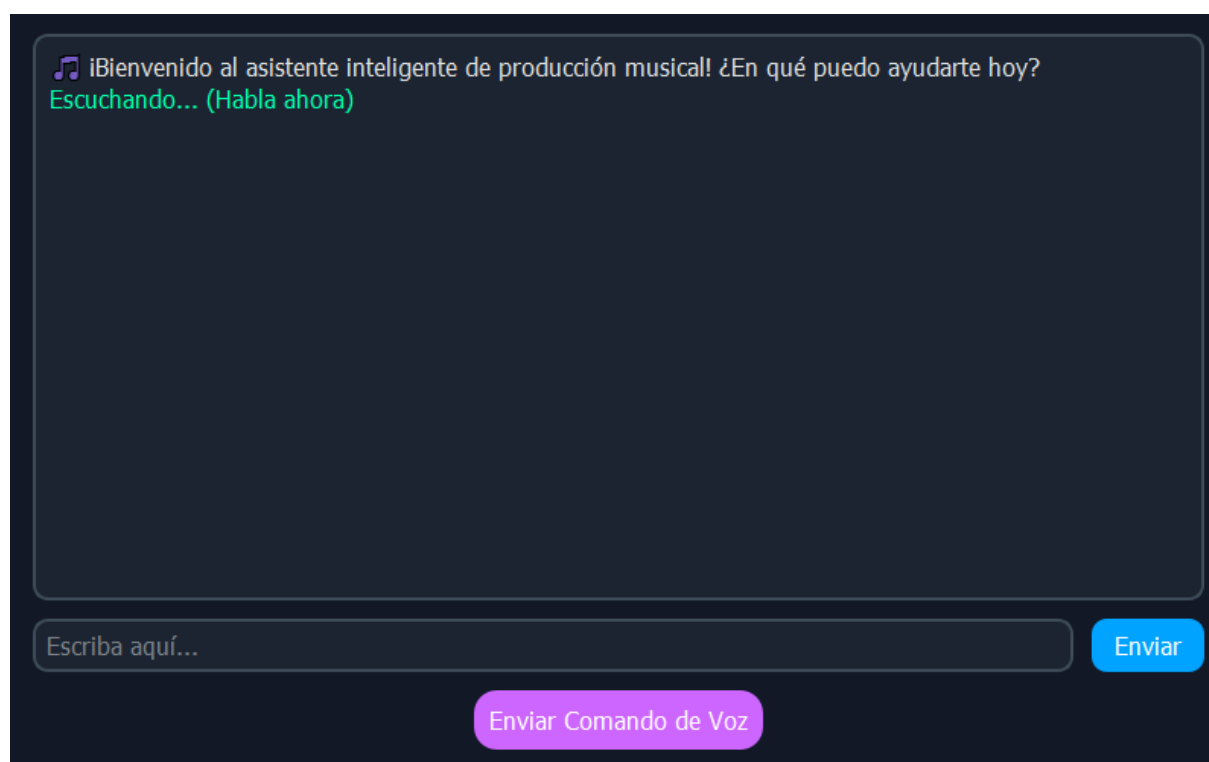


Figura 32. Pantalla del asistente durante la grabación de un comando

En el caso de que el comando realizado por el usuario no sea válido, el asistente lo notificara mediante un mensaje claro. La figura 33 muestra el mensaje por parte del asistente en caso de que el comando no sea válido:

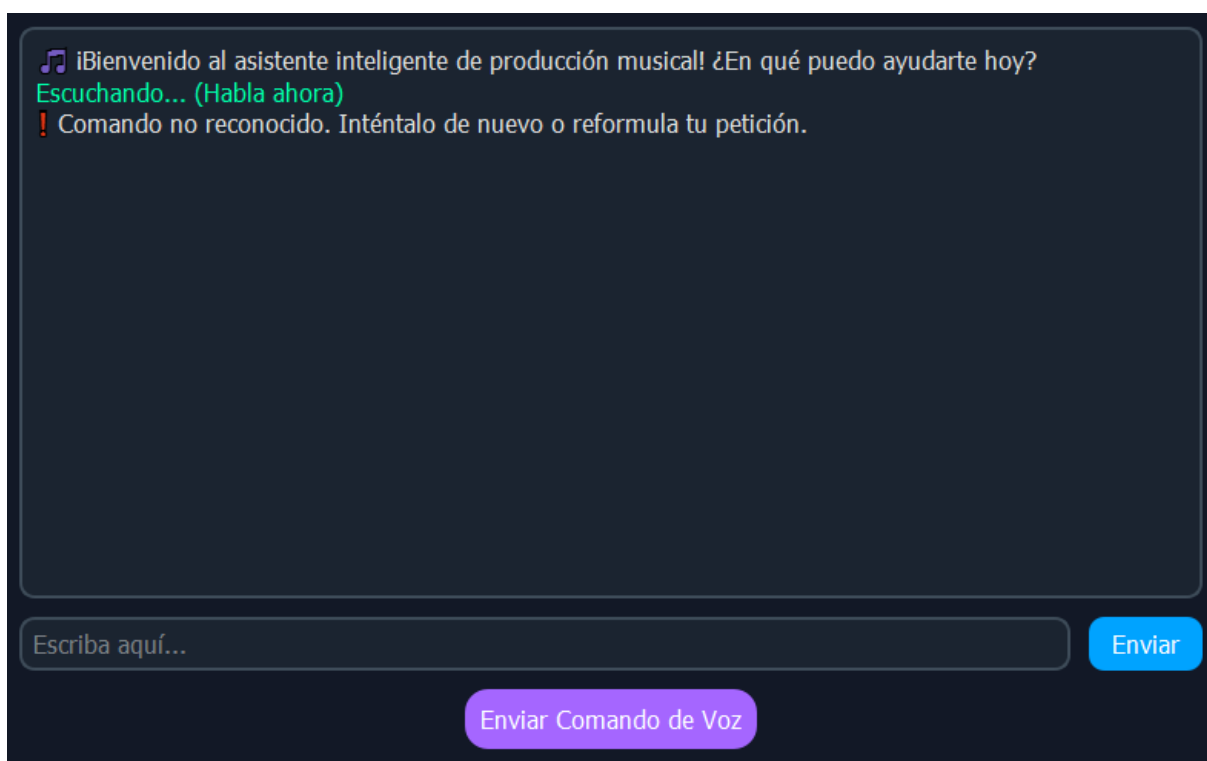
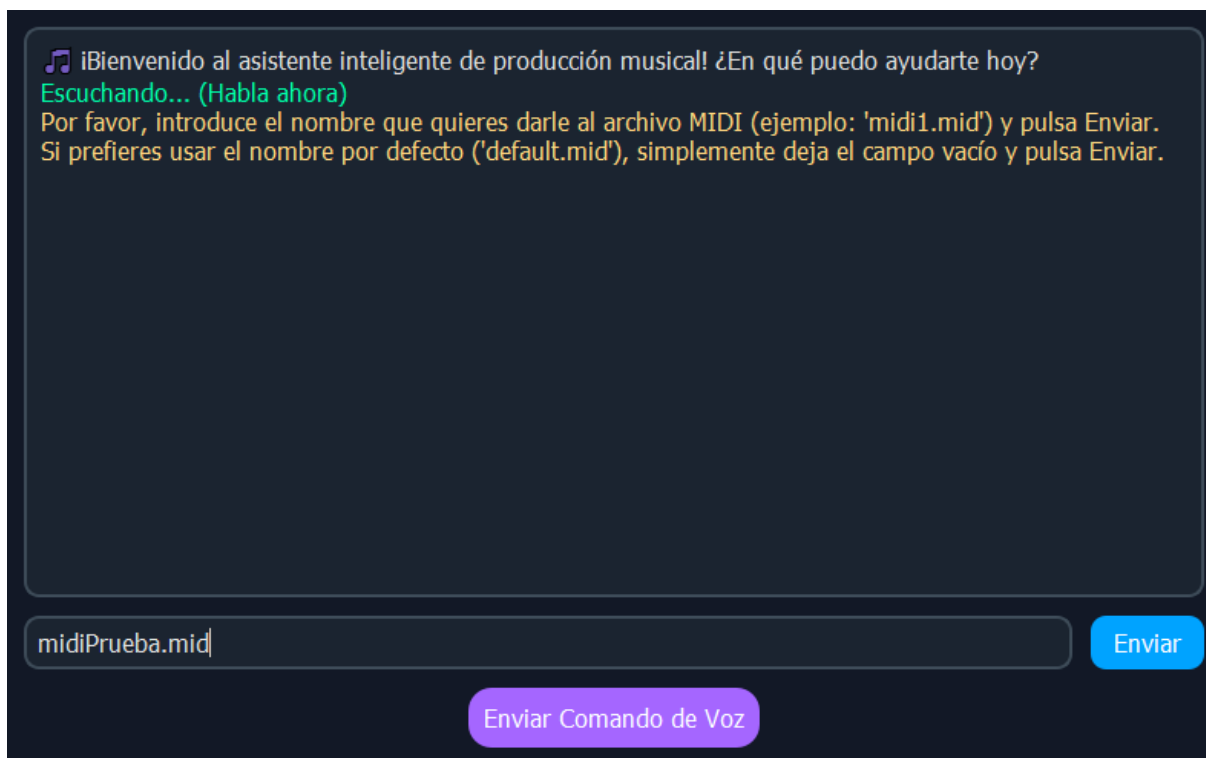


Figura 33. Mensaje del asistente tras un comando no valido

Tras esto, el usuario debe reformular su petición tras volver a presionar el botón para enviar los comandos de voz hasta que el comando sea válido.

Para las 3 primeras funcionalidades del asistente visibles en la tabla 8, una vez que el comando ha sido reconocido, el asistente realizara una petición al usuario sobre que nombre desea ponerle al MIDI que sea generado o que MIDI desea abrir en el caso de la funcionalidad 3 de la tabla 8. En la figura 34 se muestra este mensaje:



🎵 ¡Bienvenido al asistente inteligente de producción musical! ¿En qué puedo ayudarte hoy?

Escuchando... (Habla ahora)

Por favor, introduce el nombre que quieres darle al archivo MIDI (ejemplo: 'midi1.mid') y pulsa Enviar. Si prefieres usar el nombre por defecto ('default.mid'), simplemente deja el campo vacío y pulsa Enviar.

midiPrueba.mid

Enviar

Enviar Comando de Voz

Figura 34. Mensaje de petición para el nombre del MIDI

Tras introducir el nombre, el usuario debe pulsar el botón *Enviar*. Una vez el usuario pulse el botón, y para las funcionalidades 1 y 2 de la tabla 8, se abrirá una pequeña GUI cuya finalidad es la de gestionar la grabación de los tarareos por parte del usuario. La figura 35 muestra esta GUI:

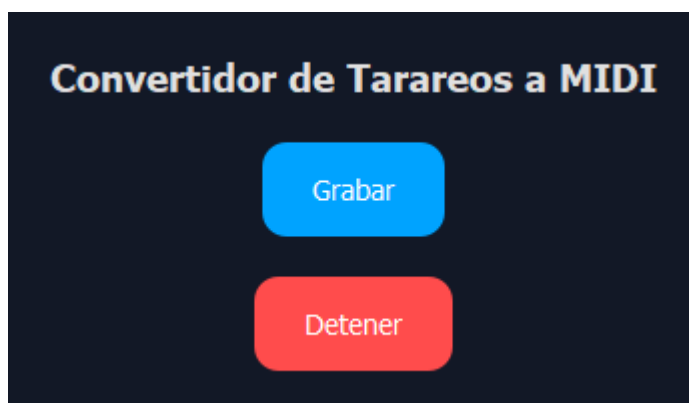


Figura 35. GUI tarareos

Esta GUI tiene un diseño muy básico y funciona de tal manera que cuando el usuario pulsa el botón grabar comienza a grabar los tarareos o melodías que el usuario realice, y una vez que el usuario desee acabar con esta grabación simplemente debe pulsar el botón detener.

Tras pulsar el botón detener y sin que el usuario realice nada más comenzará una automatización en el dispositivo la cual guardará el MIDI generado con el nombre propuesto por el usuario, abrirá el programa Reaper, importará el MIDI generado, ajustará el tempo y le pondrá como instrumento predeterminado un piano acústico a la pista de audio, además de

poner el timeline a 0, preparando el terreno para que el usuario pueda escuchar su creación de manera instantánea. Todo de manera automática sin intervención manual y en unos pocos segundos. En la figura 36 se muestra la pantalla de Reaper abierta y con todo importado, tal y como debería aparecerle al usuario tras esta automatización:

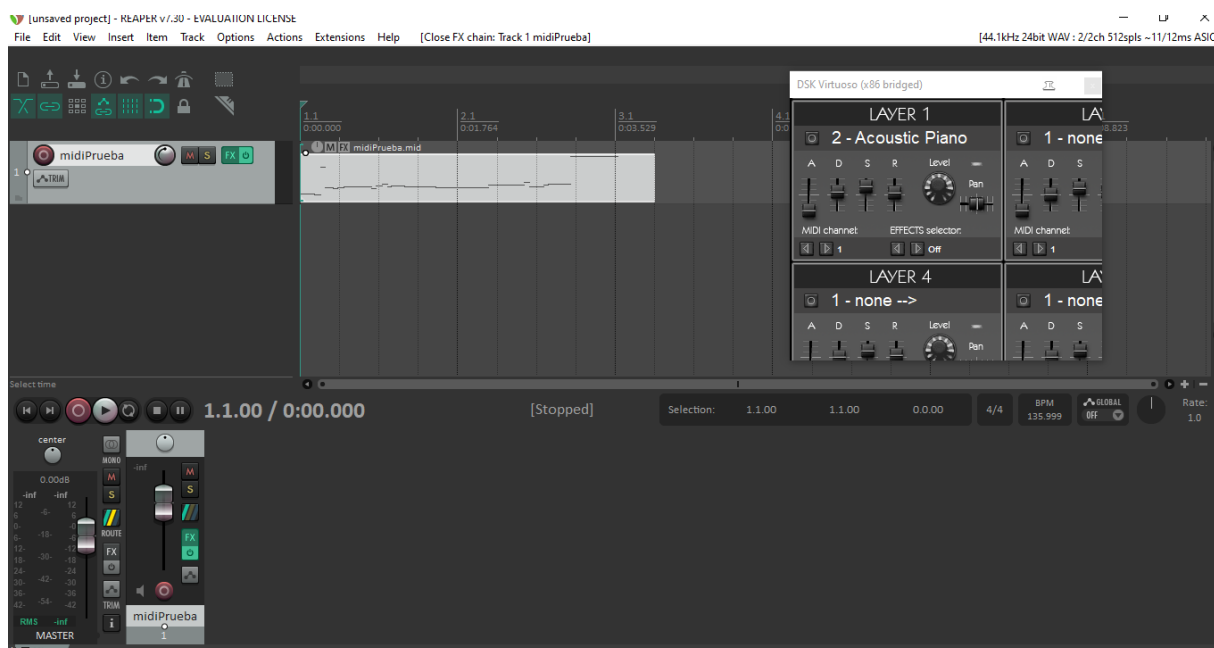


Figura 36. Pantalla de Reaper tras grabar tarareos y abrir con Reaper

Para las funcionalidades 4, 5, 6, 8 y 9 de la tabla 8 el usuario deberá seguir la misma filosofía, es decir deberá introducir los comandos de voz para cada comando en cada caso, tras esto el asistente notificará al usuario de lo que se está realizando y se procederá con las automatizaciones correspondientes. Este procedimiento es sencillo e intuitivo para el usuario y no requiere explicaciones aparte.

Sin embargo, la funcionalidad 7 de la tabla 8 tiene un funcionamiento algo distinto. Tras lanzar el comando, el sistema realiza una petición al usuario de que introduzca la pregunta deseada de forma escrita o de forma hablada. El usuario deberá realizar la pregunta que desee o bien escribiéndola en el lugar de escritura destinado para el usuario y posteriormente pulsar el botón enviar o si por el contrario desea realizarla mediante voz, deberá pulsar el botón Enviar comando de voz y realizar la pregunta. En la figura 37 se muestra la petición realizada por el asistente para la realización de la pregunta:

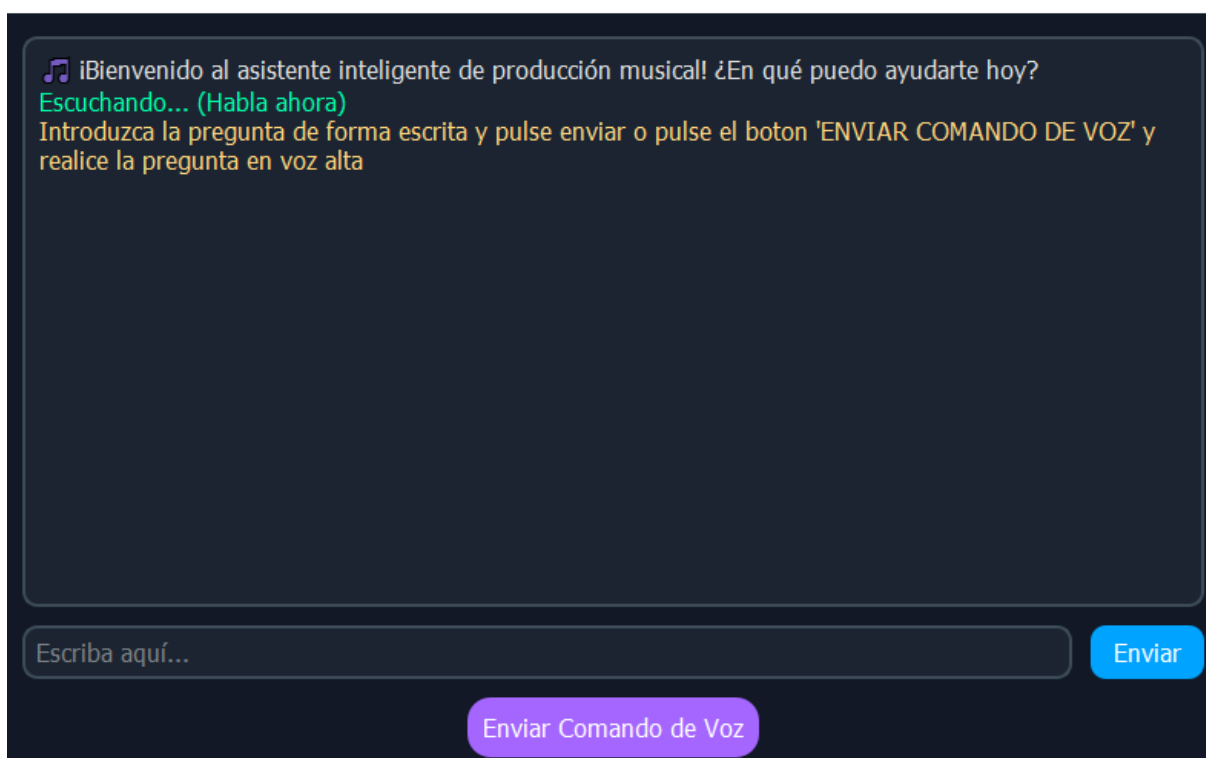


Figura 37. Petición del asistente para la realización de una pregunta por parte del usuario

El usuario además podría utilizar esta pregunta para añadirle contexto extra al asistente. Es decir, podría comentarle al asistente que desea añadirle mas contexto en caso de que haya realizado alguna cosa con Reaper en el proyecto sin utilizar el asistente o comentarle alguna idea que tenga en la cabeza y pedirle consejo o simplemente pedirle que añada este contexto nuevo al anterior.

La utilización del asistente es muy intuitiva y sencilla para el usuario, sin embargo, este manual aborda las posibles dudas que le puedan surgir al usuario a la hora de utilización de este.