Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Ajuste de modelos de difusión para la generación de audio

Tesis de Licenciatura en Ciencias de la Computación

Santiago Fiorino

Director: Pablo Riera

Buenos Aires, 2025

AJUSTE DE MODELOS DE DIFUSIÓN PARA LA GENERACIÓN DE AUDIO

La música, una de las formas de expresión artística más antiguas de la humanidad, ha evolucionado junto con los avances tecnológicos, desde los instrumentos de percusión primitivos hasta las herramientas de síntesis de audio digital modernas. Hoy en día, la inteligencia artificial desempeña un papel central en la generación de música, utilizando las últimas arquitecturas de transformers y técnicas de difusión, siendo así capaz de generar canciones completas a partir de indicaciones en lenguaje natural. A pesar de los recientes avances en modelos privados, como los desarrollados por Udio y Suno AI, que demuestran gran potencial, su naturaleza cerrada limita la investigación científica. En junio de 2024, Stability AI lanzó Stable Audio Open (SAO), un modelo de síntesis de audio basado en difusión de código abierto, democratizando la investigación en este campo. Aunque SAO tiene gran calidad en la generación de efectos de sonido, sus capacidades musicales están limitadas debido a los pocos datos de entrenamiento musicales con licencias abiertas disponibles. Nuestra investigación se centra en mejorar las capacidades de generación musical de SAO mediante el re-entrenamiento, utilizando un conjunto de datos especializado. Abordamos limitaciones específicas, incluyendo la incapacidad del modelo para generar ciertos instrumentos, dificultades para adherirse a elementos musicales especificados e inconsistencias en parámetros técnicos como el tempo y la tonalidad. El trabajo incluye la creación de un pipeline personalizado para la generación del conjuntos de datos, sintetizando audio a partir de archivos MIDI, enriqueciendo metadatos mediante APIs como Spotify y LastFM, y generando indicaciones en lenguaje natural con modelos de lenguaje (LLMs). Utilizando este pipeline, se generó un conjunto de datos de 9 horas de música que abarca diversos géneros, tempos y tonalidades. Los resultados demostraron mejoras significativas en el modelo re-entrenado ("Instrumental Finetune") en comparación con el SAO original, particularmente en calidad de sonido, precisión en la reproducción de instrumentos, adherencia a géneros y tempos, alcanzando un 95,3 % de precisión frente al 77,6 % del modelo original. La precisión del tono y la escala siguen siendo un desafío, pero las métricas basadas en representaciones, como KL-Passt y CLAP Score, indicaron que nuestro modelo ajustado iguala o supera el rendimiento tanto de SAO como del modelo comercial MusicGen, manteniendo capacidades de generalización a pesar de nuestra optimización específica del dominio.

Palabras claves: música, síntesis, difusión, *transformers*

FINE-TUNING DIFFUSION MODELS FOR AUDIO GENERATION

Music, one of humanity's oldest forms of artistic expression, has evolved alongside technological advancements, from primitive percussion instruments to modern digital audio synthesis tools. Today, artificial intelligence plays a pivotal role in music generation, leveraging state-of-the-art architectures like transformers and diffusion models capable of generating complete songs from natural language prompts. Despite recent advances in proprietary models, such as those developed by Udio and Suno AI, which demonstrate great potential, their closed nature limits scientific research. In June 2024, Stability AI released Stable Audio Open (SAO), an open-source diffusion-based audio synthesis model, democratizing research in this field. While SAO excels in sound effect generation, its musical capabilities are constrained by limited open-license training data. Our research focuses on enhancing SAO's musical generation capabilities through fine-tuning on a specialized dataset. We address specific limitations including the model's inability to generate certain instruments, difficulties in adhering to specified musical elements, and inconsistencies in following technical parameters such as tempo and tonality. The research involves creating a custom dataset-creation pipeline by synthesizing audio from MIDI files, enriching metadata using APIs like Spotify and LastFM, and generating natural language prompts via large language models (LLMs). Using our pipeline, a 9-hour music dataset was generated, spanning various musical genres, tempos and tonalities. Results demonstrate significant improvements in the fine-tuned model ("Instrumental Finetune") compared to the original SAO, particularly in sound quality, instrument reproduction accuracy, genre adherence, and tempo adherence, where our model achieved 95.3 % accuracy, compared to 77.6 % in the original model. Tone and scale accuracy remained challenging, and embedding-based metrics such as KL-Passt and CLAP Score indicate that our fine-tuned model matches or exceeds the performance of both SAO and the commercial model MusicGen, maintaining generalization capabilities despite our domain-specific optimization.

Keywords: music, synthesis, diffusion, transformers

A mis abuelas, Olga y Marina.

Índice general

1. INTRODUCCIÓN

Los seres humanos comenzamos a experimentar con la música tan pronto como desarrollamos el lenguaje. Esta forma de arte, quizás la más antigua, la mantenemos hasta el día de hoy como una de las expresiones culturales más importantes, con una industria global que abarca la composición, grabación, promoción y educación, involucrando a millones de trabajadores y generando miles de millones de dólares por año.

Si bien la voz y las manos son suficientes para crear música, hace al menos 40.000 años comenzamos a desarrollar instrumentos musicales para ampliar nuestras capacidades, desde palos y piedras para percusión, hasta flautas para sonidos similares al de los pájaros. Con el avance de nuestra tecnología, los instrumentos fueron evolucionando en paralelo, dando lugar a una enorme diversidad de instrumentos de todo tipo, tanto acústicos como analógicos y digitales.

Actualmente, la computadora tiene un papel central en todos los aspectos de la música, tanto en la creación de la composición y producción como en la distribución y reproducción. Además, la música tiene un lugar relevante en la investigación científica en computación, con el desarrollo de algoritmos de procesamiento de señales, transcripción, análisis de obras, entre otros. Por lo tanto, el reciente renacimiento de la inteligencia artificial no pasó desapercibido en la música, dando lugar a modelos avanzados de recomendación, clasificación, transcripción, composición y, lo que haremos foco en esta tesis, síntesis de audio.

Gracias a los últimos avances, principalmente las arquitecturas de transformers y los modelos de difusión, hoy contamos con modelos de generación de música *end-to-end*, que partiendo de tan solo una frase escrita en lenguaje natural, estos modelos son capaces de generar canciones completas. En base a estos modelos, se crearon empresas dedicadas completamente a brindar este servicio, como Udio y Suno AI, la cual en pocos meses llegó a una valuación de 500 millones de dólares. Estas plataformas permiten crear obras de distintos géneros, con múltiples instrumentos y vocales que siguen letras específicas. Si bien estas empresas cuentan con modelos con gran capacidad, al ser empresas privadas, estos modelos, sus pesos y algoritmos no son públicos, lo que limita su uso en la investigación científica.

En junio del 2024, la empresa Stability AI lanzó Stable Audio Open, un modelo de síntesis de audio por difusión completamente abierto, democratizando la investigación en el área. El modelo tiene una gran capacidad de generar efectos de sonido, pero a la hora de generar música tiene más limitaciones debido a los pocos datos de entrenamiento musicales disponibles bajo licencias abiertas.

En base a esa limitación, nuestro experimento consistirá en la búsqueda de una mejora en la habilidad musical de este modelo, permitiendo que aprenda a sintetizar nuevos instrumentos, nuevos géneros, y pueda ajustarse a aspectos técnicos musicales como el tempo y la tonalidad. Además, dado que Stable Audio Open es un modelo muy reciente, es valiosa la investigación en cuanto al proceso de entrenamiento, los recursos necesarios y su capacidad de personalización.

Para contextualizar con mayor detalle cómo llegamos a este punto tan avanzado en la síntesis de audio, donde podemos generar canciones enteras con solo un clic, en el Capítulo 2 exploramos la historia completa de la síntesis de audio, desde los primeros instrumentos

electrónicos hasta los últimos modelos de inteligencia artificial. En el siguiente capítulo se procede a explicar las bases fundamentales de estas arquitecturas, como los autoencoders, transformers y la difusión, con los detalles particulares de Stable Audio Open para poder entender su funcionamiento. A continuación, el Capítulo 4 presenta el experimento, detallando primero la metodología utilizada para generar el conjunto de datos de entrenamiento, y luego el entrenamiento propio con los resultados obtenidos.

2. HISTORIA DE LA SÍNTESIS DE AUDIO

Si bien actualmente existen modelos *end-to-end* como el que utilizaremos para nuestro experimento, capaces de generar canciones enteras a partir de una frase que las describe, la síntesis de audio cuenta con más de un siglo de avances. Los primeros instrumentos electrónicos eran enormes, difíciles de controlar, costosos y no ofrecían la mejor calidad de sonido. A lo largo de los años, siguiendo los avances tecnológicos, se mejoró progresivamente la portabilidad, el control, la expresividad y lógicamente, la calidad de los sonidos de los instrumentos. Dentro de la historia de innovaciones en la síntesis de audio identificamos cinco etapas principales:

2.1. El comienzo de los instrumentos electrónicos

A finales del siglo XIX y principios del siglo XX se produjeron grandes avances en la generación y distribución de electricidad, así como en la invención de aparatos eléctricos, incluyendo no menos que la bombilla de Edison y los motores eléctricos, los cuales transformaron la industria. Paralelamente, las redes telefónicas comenzaban a expandirse, facilitando la transmisión de música a gran escala.

En este contexto, Elisha Gray, uno de los inventores del teléfono, creó el "Telégrafo musical" (1874), un instrumento diseñado para transmitir tonos musicales a larga distancia a través de la red telegráfica. La intención principal del invento no era completamente musical, sino que era la de transmitir múltiples mensajes a través de la red (precursor de la multiplexación), por lo tanto los tonos transmitidos no eran expresivos y suaves, sino que violentos y duros de escuchar. Basándose en el telégrafo musical, Thaddeus Cahill inventó el "Telharmonium", el primer instrumento electrónico relevante, que sintetizaba música y la distribuía a través de la red telefónica. En este caso el objetivo sí era producir sonidos agradables.

En esa época se descubría que el sonido de los instrumentos está compuesto por el tono fundamental, combinado con parciales armónicos que le dan a cada instrumento su timbre único. Inspirado en esta idea, Cahill buscó crear el "instrumento perfecto", sintetizando perfectamente todos los tonos, eliminando así la imperfección de los instrumentos acústicos. La principal innovación del Telharmonium fue la rueda tónica, un rotor con alternadores de distintos tamaños que giraba dentro de un campo magnético para generar tonos. Cada rueda estaba compuesta por la tonalidad fundamental y seis parciales ascendentes. La cantidad de parciales para cada tono podía controlarse, permitiendo imitar instrumentos orquestales. De esta forma, el Telharmonium se convirtió en el primer **sintetizador aditivo**. No obstante, el control del instrumento no era intuitivo, tenía una distribución de teclas compleja, alternando teclas negras y blancas, que debían ser tocadas en 3 niveles para alcanzar una escala temperada.

El Telharmonium producía un sonido claro y puro, destacándose por su capacidad de imitar diversos instrumentos como trompetas, flautas, oboes, cellos y violines. Si bien el sonido era bueno, no logró éxito comercial por su elevado costo, su tamaño (ocupaba habitaciones enteras) y su alto consumo energético. Además, quedó rápidamente obsoleto con los avances tecnológicos, especialmente la transmisión de radio. Sin embargo, sentó bases para instrumentos como el órgano Hammond.

A principios siglo XX, se descubrió el fenómeno de la **heterodinación**, que ocurre cuando se combinan dos frecuencias de audio altas similares, pero distintas; esto genera una nueva frecuencia baja audible, equivalente a la diferencia entre las dos frecuencias altas, denominada frecuencia heterodina. Explotando este fenómeno, en 1920 Leon Theremin creó el "Theremin", un instrumento que genera una frecuencia fija de 170.000Hz, y una frecuencia variable entre 168.000 y 170.000Hz. Al combinarse las frecuencias, se genera una frecuencia audible de entre 0 y 2.000Hz.

El Theremin se caracteriza por su método de control, sin necesidad de contacto físico. El instrumento cuenta con dos antenas que funcionan como sensores que detectan la posición de las manos del intérprete: una antena ajusta la frecuencia variable, permitiendo alterar la nota, y la otra antena controla la amplitud, permitiendo alterar el volumen.

El sonido del instrumento es etéreo o fantasmal, se desliza entre tonos de forma similar a una voz o un violín. Este sonido misterioso resultó en una alta adopción en bandas sonoras de películas, particularmente de ciencia ficción. Si bien la forma de controlarlo era novedosa, es difícil de dominar. Esto, junto al sonido poco convencional que tiene, limitó su adopción masiva. Aún así, el Theremin se destaca como un instrumento pionero en la historia de la música electrónica.

En 1923, el violonchelista Maurice Martenot conoció a Leon Theremin en un encuentro que lo inspiró a diseñar un instrumento basado en sus ideas. En 1928, patentó el "Ondes-Martenot". Su objetivo era crear un instrumento versátil, inmediatamente familiar para músicos orquestales. Las versiones más conocidas del Ondes-Martenot contaban con un teclado estándar para un control preciso de las notas, y un anillo que se deslizaba a través de un cable para un control más expresivo, que generaba sonidos similares a los del Theremin con efectos de glissando y vibrato. Las versiones posteriores también permitían generar vibrato mediante el movimiento horizontal de las teclas.

El Ondes-Martenot contaba con timbres predefinidos, que permitían modificar la forma de onda entre sinusoidal, triangular, cuadrada, entre otras, ofreciendo así una variedad de timbres. También contaba con diferentes altavoces que aportaban una capa adicional de control sobre el sonido generado, como un gong para sonidos metálicos o una cámara de cuerdas para tonos resonantes. Estas configuraciones permitían expresar una amplia variedad de emociones, y la imitación de distintos instrumentos, aunque con un sonido electrónico distintivo.

Gracias a la versatilidad de los sonidos sintetizados y a la posibilidad de controlarlo de forma precisa mediante un teclado clásico, el Ondes-martenot se convirtió en el primer instrumento electrónico en alcanzar éxito. Fue adoptado principalmente en composiciones de música clásica y bandas sonoras de películas, especialmente en géneros de ciencia ficción y terror. Su influencia perduró hasta hoy, siendo el único instrumento de su generación que se sigue utilizando en la actualidad, apareciendo en discos contemporáneos emblemáticos como "Kid A" de Radiohead, o "Random Access Memories" de Daft Punk.

En ese momento, existía una carrera entre los inventores de instrumentos eléctricos para replicar el sonido de un órgano de iglesia. El Telharmonium hasta cierto punto lo había logrado, aunque sin éxito comercial debido a las limitaciones mencionadas anteriormente. Siguiendo la metodología del Ondes-Martenot, que adaptó el sonido del Theremin al método de control instrumental clásico de teclado, en 1935 Laurens Hammond y John M. Hanert tomaron los principios del Telharmonium (las ruedas tónicas y la síntesis aditiva) para desarrollar el órgano Hammond, uno de los instrumentos electrónicos más populares y duraderos de la historia.

A diferencia del Telharmonium, el órgano Hammond era compacto, no mucho más grande que un piano vertical tradicional. En lugar de la compleja distribución de teclas del Telharmonium, el órgano Hammond incluía dos teclados clásicos de cinco octavas, y barras deslizantes que permitían añadir armónicos parciales al tono fundamental (síntesis aditiva). También contaba con un dispositivo de reverberación electromecánica, que sería copiado en muchos instrumentos electrónicos posteriores, además de efectos de vibrato y chorus, ambos controlables.

Si bien inicialmente fue diseñado y comercializado para iglesias como una alternativa económica a los gigantes órganos de tubos, el órgano Hammond se popularizó rápidamente en músicos de jazz como Fats Waller y Count Basie. Más tarde se extendió al rhythm and blues y al rock progesivo, siendo adoptado por bandas icónicas como como Pink Floyd, Génesis, Yes y Deep Purple.

Diez años más tarde, en 1945, Hanert —uno de los inventores del Hammond— diseñó la "Hanert Electric Orchestra". En este caso la propuesta era más ambiciosa, buscaba revolucionar el paradigma musical con un instrumento capaz de sintetizar y grabar composiciones musicales en un disco a partir de tarjetas de notación musical escritas por un compositor, sin la intervención de un intérprete.

Esto les permitía a los compositores crear "composiciones perfectas", especificando en las tarjetas las notas, envolventes, vibratos, volumen, entre otras propiedades. El tempo final de la pieza podía ajustarse variando la velocidad del dispositivo de escaneo. Por primera vez un compositor podía mezclar, transponer e invertir sus composiciones a voluntad. El instrumento también ofrecía la posibilidad de modificar el timbre sintetizado en cualquier momento.

A pesar de las grandes innovaciones, la Hanert Electric Orchestra no tuvo éxito, solamente fue usada por el propio Hanert. La máquina era demasiado grande, los compositores no se adaptaron al cambio de paradigma que ofrecía y los sonidos eran demasiado sintéticos comparado a los instrumentos de la época, que con las nuevas tecnologías ya tenían una muy buena calidad de sonido. Sin embargo, está claro que este instrumento estableció precedentes en la producción musical, cuyos principios siguen presentes hasta el día de hoy. Sin ir más lejos, RCA en 1950 comisiona una máquina de producción musical que pueda analizar y sintetizar música pop sin intérpretes en medio.

2.2. La ascensión de la síntesis analógica

En 1950, RCA —uno de los mayores conglomerados de entretenimiento de la época— comenzó una investigación para lograr la generación automática de *hits* de pop. Analizando miles de canciones, buscaban entender qué es lo que hace que un *hit* sea un *hit*, y luego reutilizar la formula para generar *hits* propios. El proyecto también pretendía reducir los costos de las sesiones de grabación, automatizando el proceso sin depender de intérpretes, tal como lo había anticipado Hanert.

Con esa idea en mente, en 1951 desarrollaron el RCA Mark I Synthetizer, uno de los primeros **sintetizadores programables**. Este instrumento era básicamente una computadora analógica que procesaba notación musical escrita a través de un sistema similar al de una máquina de escribir, con un formato binario que se perforaba sobre un papel. En esta notación se determinaba el tono, timbre, volumen y envolvente de cada nota. Por cada parámetro, el papel tenía cuatro columnas de agujeros, dándole un rango de 16 niveles a cada parámetro. Con la evolución al RCA Mark II Synthetizer (1957) se añadieron filtros de paso alto y bajo, generadores de ruido, glissando, vibrato y resonancia, logrando obtener una combinación de millones de configuraciones posibles. Para cuando se construyó el Mark II, la idea inicial del análisis de los *hits* de pop ya había sido abandonada.

El sonido del sintetizador se generaba por osciladores de válvulas de vacío. La ruta de la señal se manejaba manualmente con cables, técnica que se adoptó en los sintetizadores modulares de los 60s y 70s. El sonido final se monitoreaba en altavoces y se grababa en un disco. Mediante técnicas de regrabación y mezcla se podían componer piezas con alto grado de complejidad, llegando hasta las 216 pistas de sonido. Las propiedades de secuenciación de música eran las más llamativas para los compositores de la época, dando nacimiento a la complejidad musical, música con ritmos y tempos que son imprácticos o directamente imposibles en instrumentos acústicos.

Si bien este instrumento marca un gran paso en la historia de la música electrónica, a los pocos años quedó obsoleto. Su configuración era difícil, requiriendo un extenso trabajo de cableado y circuitería analógica previa a sintetizar una composición. Su tamaño también era muy grande y pocas personas sabían cómo operarlo. Con la llegada de la tecnología de transistores de estado sólido, llegaron instrumentos más compactos, económicos y fáciles de programar, como el Moog Synthetizer, el próximo hito en la música electrónica.

En 1964, Robert (Bob) Moog —fundador de una compañía dedicada a la producción de instrumentos electrónicos, entre ellos el Theremin— recibió de Herb Deutsch el encargo de crear un instrumento capaz de producir sonidos complejos y experimentales, tonos imposibles de generar con instrumentos o técnicas de grabación existentes. Para el año siguiente, Moog y Deutsch habían completado el primer prototipo del sintetizador Moog, dando nacimiento al primer **sintetizador modular**. Como su nombre indica, este instrumento estaba compuesto de módulos con distintas funciones: osciladores para generar varios tipos de ondas, filtros para modelar frecuencias, amplificadores para controlar la amplitud y envolventes para definir la forma en la que evoluciona el sonido en el tiempo. La interconexión de los módulos se realizaba mediante cables, permitiendo a los usuarios crear un cantidad casi ilimitada de configuraciones únicas para diseñar nuevos sonidos. Este tipo de síntesis, basada en filtros sobre señales de audio para cambiar el timbre es llamada **síntesis sustractiva**.

El diseño no era completamente nuevo, seguía el patrón común en la historia de los instrumentos electrónicos: integrar conceptos preexistentes para crear instrumentos más pequeños, accesibles, de mayor calidad, mejor control y expresividad. Las principales innovaciones de Moog fueron, por un lado, la sistematización de la producción de sonidos generados electrónicamente basada justamente en los distintos módulos que propuso; y por el otro, el uso del voltaje para las señales eléctricas que controlan las distintas funciones de cada módulo, estableciendo el estándar que vive hasta el día de hoy, de 1 volt por octava.

La salida final del sintetizador podía controlarse con un teclado clásico, pero también a través de joysticks, pedales, secuenciadores, o incluso con otro módulo del propio sintetizador. Esto permitía, por ejemplo, partiendo de ruido blanco u osciladores de bajas frecuencias, controlar cualquier aspecto no sólo del sonido en sí, sino de las notas, tempo, y otras variables de la composición. En los instrumentos anteriores, como el Mark II, si bien no se necesitaban intérpretes para generar los sonidos, la composición dependía enteramente de las personas que escribían la notación musical en la máquina. Sin embargo, en el Moog, al conectar distintos generadores de ondas, se pueden generar patrones sonoros y progresiones de notas impredecibles, permitiendo que el propio sintetizador genere composiciones innovadoras.

En el 1970, Moog creó el Minimoog, una versión más compacta del modelo original, autocontenida. Este fue el primer sintetizador en venderse en tiendas de música, a un precio más accesible, y es considerado el sintetizador más famoso de la historia. Su portabilidad era lo que lo destacaba, junto a la gran variedad de sonidos que era capaz generar, desde lineas de bajo y melodías que hasta el momento eran propias de guitarras, hasta sonidos ya reconocibles del estilo de otros sintetizadores.

Naturalmente, la competencia no se quedó atrás. Ante la llegada del Minimoog surgieron competidores, destacándose el ARP Odyssey, introducido por ARP Instruments en 1972. Este sintetizador se destaca por ser el primero en ofrecer capacidades **duofónicas**, es decir, la posibilidad de tocar dos notas al mismo tiempo. Hasta entonces, los sintetizadores eran monofónicos, produciendo solo una nota a la vez.

Cinco años después del ARP Odyssey, en 1977, Dave Smith y John Bowen diseñaron el sintetizador analógico Prophet-5, manufacturado por la empresa Sequential. Este fue el primer **sintetizador polifónico**, y también el primero con una memoria programable. Esto era revolucionario, en los demás sintetizadores era necesario reconfigurar manualmente todos los parámetros cada vez que se deseaba recrear un sonido, sin garantía de obtener el mismo resultado exacto. Con el Prophet-5, los usuarios podían guardar configuraciones de sonidos en memoria y cargarlas en el sintetizador instantáneamente. Gracias a esta innovación, se convirtió en un líder del mercado, ampliamente utilizado en la música pop.

2.3. La revolución digital

Durante la década de 1960 se empezaba a experimentar con la **síntesis digital** de audio, que a diferencia de la analógica, que crea sonidos usando circuitos electrónicos, la digital usa el procesamiento de señales digitales, que luego son convertidas a analógicas. En los 60s lograron generar sonidos digitalmente, pero aún estaban lejos de lo musical y aún más lejos del nivel de lo analógico. No fue hasta 1977 que saldría el primer sintetizador comercial con generación de sonido puramente digital: el Synclavier de New England Digital. Este fue primer sintetizador digital basado en **síntesis por modulación de frecuencias** (FM), una técnica en la que se varía la frecuencia de una señal portadora mediante una segunda frecuencia moduladora, generando una onda con frecuencia modulada. La FM permite generar ondas con alto grado de complejidad con solo dos osciladores, a diferencia de la síntesis aditiva, que requiere un oscilador para cada armónico, o la sustractiva, que precisa de un filtro para modificar la señal. Su elevado precio lo hizo inaccesible para la mayoría de los músicos, pero se popularizó entre los productores y estudios de grabación profesionales.

Dos años después, en 1979, Fairlight lanzó el Fairlight CMI, un sintetizador digital, *sampler* y estación de trabajo de audio. Es el primer sintetizador con un *sampler* integrado, y se le atribuye la creación del termino *sampling*, técnica y término muy utilizados en la producción musical actual. Esta técnica surgió al grabar una nota de piano de la radio para analizarla y luego recrearla en un sintetizador digital, pero descubrir que al reproducir la grabación en distintos tonos se obtenía un sonido de piano más fiel que el sintetizado. Los usuarios podían controlar el *attack, sustain, decay* y *vibrato* de los sonidos. Tras el éxito del Fairlight CMI, New England Digital modificó el sintetizador Synclavier para añadir capacidades de *sampling*.

Si bien el Synclavier y el Fairlight CMI contenían las principales innovaciones en cuanto a la síntesis digital, el Synclavier costaba entre 25.000 y 200.000 dólares, y el Farilight CMI desde 25.000, con solo 100 unidades fabricadas, limitando su impacto al público general. La síntesis digital alcanzó un verdadero éxito masivo en 1983 con la llegada del Yamaha DX7, que era innovador y accesible. Con un precio de 2 mil dólares, logró vender más de 200.000 unidades.

El DX7 contaba con un teclado de cinco octavas, sensores de velocidad y presión para controlar la expresividad. y permitía una polifonía de 16 notas. Fue el primer sintetizador en incluir una pantalla LCD y permitir a los usuarios nombrar sus propios sonidos. Aunque su interfaz de usuario era poco convencional, complicando la programación de sonidos propios, sus sonidos predefinidos se convirtieron en íconos de la musica pop de los 80. Para 1986, el DX7 estaba presente en el 40 % de las canciones del Billboard Hot 100 de Estados Unidos.

En los años siguientes, los sintetizadores digitales siguieron evolucionando y mejorando. Entre los nuevos lanzamientos se destaca el Roland D-50 de 1987, que introdujo la **síntesis de aritmética lineal**, una técnica que combina la síntesis sustractiva tradicional con *samples* representados mediante modulación por impulsos codificados (PCM).

Otro sintetizador notable fue el Korg M1 en 1988, reconocido por sus capacidades como estación de trabajo de audio para la composición de canciones. Incluía un secuenciador de 8 pistas y efectos digitales como reverb, delay, chorus, trémolo, ecualización y distorsión.

Esto les permitía a los músicos producir canciones enteras sin necesidad de un estudio profesional, antes del auge de las estaciones de trabajo de audio digitales (DAWs).

Finalmente, se destaca el Nord Lead, lanzado por Clavia en 1995, el primero al cual se le atribuye el término de **sintetizador virtual analógico** (VA), aunque el Roland D-50 también lo era. Estos sintetizadores generan los sonidos de los sintetizadores analógicos tradicionales simulándolos por software, con algoritmos de procesamiento de señales digitales. La síntesis VA es más confiable que la analógica, ya que los tonos de los osciladores se mantienen estables gracias al reloj digital, y el hardware digital es menos susceptible a las variaciones de temperatura. Además, mientras que en los sintetizadores analógicos se requería de un circuito oscilador para cada voz en la polifonía, los sintetizadores VA permiten tener tantas voces polifónicas como el procesdor en el que corren pueda soportar.

2.4. La era del software

A principios de los años 80 ya existía una amplia variedad de instrumentos electrónicos de diferentes compañías. Sin embargo, cada uno utilizaba su propio estándar, lo que limitaba el crecimiento de la industria. Para resolver este problema, las principales empresas colaboraron en la tarea del diseño de un estándar común, incluyendo representantes de Roland, Yamaha, Korg, Sequential Circuits, y el propio Robert Moog, quien anunció MIDI en 1982 en una revista especializada. En 1983, se realizó una demostración de conexión MIDI entre los nuevos sintetizadores Prophet 600 (de Sequential) y Roland JP-6, los primeros instrumentos con MIDI. Ese mismo año saldría la primera caja rítmica MIDI, la Roland TR-909 y el primer secuenciador MIDI, el Roland MSQ-700.

El término MIDI significa 'Musical Instrument Digital Interface", y su objetivo es permitir la comunicación entre diferentes instrumentos. Un secuenciador MIDI, por ejemplo, podría generar ritmos que luego serían producidos por un módulo de sonido de batería. Cuando se toca una nota en un instrumento MIDI, se genera un mensaje que cumple con ciertas especificaciones, permitiendo la reproducción de sonidos en otro instrumento. MIDI también permite el control de otros parámetros instrumentales como el volumen, efectos, velocidad, etc.

En la época del lanzamiento del estándar MIDI las computadoras personales empezaban a ser cada vez más comunes y poderosas, llegando a ser viables para la producción musical. En 1984 se lanza el Yamaha CX5M, el cual añadió soporte MIDI y secuenciación al sistema MSX, permitiendo conectar un teclado MIDI a la computadora, usar alguno de los 48 sonidos predefinidos y secuenciar hasta 8 canales de música, un flujo de trabajo no muy distinto al actual. La integración a las computadoras se expandió aún más cuando Roland lanzó la primera tarjeta capaz de procesar secuencias y sonidos MIDI, estableciendo la interfaz estándar MIDI a PC.

MIDI definió un formato estándar para guardar, transportar y cargar archivos que contienen secuencias musicales, usando la extensión `.mid`. Estos archivos guardan información como los valores de las notas, tiempos, velocidades, volúmenes, y pueden incluír metadatos como las letras y el tempo de las canciones.

La adopción masiva de MIDI y su integración a la computadora se complementó perfectamente con la llegada de las estaciones de trabajo de audio (DAWs) en formato de software, que cada vez tenían mayor capacidad. Los DAWs ya eran populares previo a las computadores, como parte de los sintetizadores o como equipos independientes, pero el aumento en la velocidad y la disminución en precio de las computadoras personales impulsó

la adopción de los DAWs como software de PC. En 1989, Sonic Solutions lanzó el primer software de edición de audio no lineal (sin modificar la señal original cuando se edita). Dos años más tarde, las principales discográficas adoptaron por completo la digitalización, con el lanzamiento del conocido DAW Pro Tools, el cual sigue siendo usado hasta hoy.

En 1993, Steinberg lanzó Cubase Audio, un DAW con efectos integrados, grabación y reproducción de 8 pistas. Incluía pistas de acordes, las cuales podían ser usadas para armonizar audios y pistas MIDI automáticamente, o para generar arpegios, avanzando así hacia la asistencia en la composición. A esta altura las máquinas ya no sólo sintetizan el sonido, sino que ya cumplen roles dentro de la composición. Otra de sus innovaciones principales, hoy estándar en cualquier DAW moderno, es el inspector de notas, que permitía control preciso para dibujar acordes, cuantizar las notas, transponer acordes y editar la duración de las notas.

Con el lanzamiento de Cubase, Steinberg introdujo la interfaz VST (Virtual Studio Technology), interfaz de *plugins* que integra sintetizadores y efectos de software dentro de los DAWs. Los VST se dividen principalmente en instrumentos (VSTi) y efectos (VSTfx). Los instrumentos consisten en emulaciones de software de sintetizadores y samplers de hardware famosos, imitando el aspecto y funcionalidad de sus versiones físicas. Los VSTi reciben notas mediante mensajes MIDI y generan audio digital. Los VSTfx reciben audio digital y lo procesan generando la versión del audio con el efecto deseado. Cubase incluía VSTfx de reverb, chorus, echo y panner. El primer VSTi fue Neon, incluído en la versión 3.7 de Cubase, y consistía en un sintetizador virtual analógico de 16 voces con 2 osciladores.

La combinación de controladores MIDI conectados a estaciones de trabajo de audio digitales con distintos VSTs se convirtió en la forma estándar de producción musical. Desde las principales discográficas hasta músicos amateurs, se redujo la barrera de entrada a la composición musical como nunca antes.

2.5. El auge de la inteligencia artificial

Si bien la inteligencia artificial hoy en día está en su apogeo, el término y muchas de sus técnicas provienen de décadas pasadas. La idea de generar música automáticamente no es nueva, tan pronto como en 1957 se creó la que es considerada la primera composición hecha por una computadora, la "Illiac Suite"[26]. Para esta pieza, Hiller e Isaacson emplearon distintas técnicas, aplicando una distinta para cada uno de los cuatro movimientos. La técnica destacada de esta composición es la del cuarto movimiento, para el cual usaron gramáticas generativas y cadenas de Márkov.

En 1960, Zaripov publicó en una revista soviética un algoritmo para la composición de música basado en reglas matemáticas que él mismo formuló: cada frase musical debía terminar en una de las tres notas principales de la escala; no se permitían dos intervalos largos consecutivos; el número de notas desplazándose en una dirección no debía superar los 6, entre otras [28]. A pesar de la rigidez de las reglas, dado que Zaripov era músico, comprobó mediante experimentos en la radio donde mezclaban composiciones reales con artificiales que los usuarios disfrutaban de ambas, llegando incluso a preferir las generadas por computadora.

Con el formato MIDI ya adoptado, en 1997 David Cope desarrolló un programa llamado "*Experiments in Musical Intelligence* (EMI)", el cual usa una base de datos de archivos MIDI, idealmente con un estilo en particular, por ejemplo de composiciones de Bach, para generar composiciones que imitan dicho estilo [2]. Este programa primero realizaba un análisis melódico y armónico sobre las piezas del conjunto de datos, identificando patrones melódicos recurrentes, y en particular "firmas", que son aquellos patrones que caracterizan el estilo del autor que se busca imitar. En base a los análisis iniciales, EMI generaba conjuntos de reglas que especifican cómo combinar los distintos patrones detectados, permitiendo la generación de nuevas composiciones mediante la recombinación de estos patrones.

Inspirado por la idea de imitación de estilo de EMI y por los recientes sistemas de composición de música interactivos en tiempo real, en 2002 François Pachet presentó el sistema llamado "*The Continuator*", que unificaba estos dos conceptos en una herramienta capaz de extender la habilidad técnica de los músicos con composiciones que seguían su estilo, el cual era aprendido automáticamente [20]. El modelo utilizaba cadenas de Markov con optimizaciones para mejorar la eficiencia de los métodos de aprendizaje. Aunque esta tecnología era capaz de representar eficientemente patrones musicales, su poder de generación era limitado, ya que no contaba con información a largo plazo, generando así piezas con coherencia en pequeña escala, pero incoherentes cuando se analizan las piezas completas.

Las redes neuronales recurrentes (RNNs) permiten modelar cadenas de Markov e incluso funciones más complejas. Durante la década de los 90 se exploró su uso en la generación de música. Sin embargo, estas redes seguían teniendo el problema de la falta de una estructura global, probablemente debido al problema de desvanecimiento del gradiente. Podían aprender transiciones entre cada nota, reproducir frases, pero fallaban a la hora de generar una composición entera. En este contexto, en 2002, Douglas Eck (quien más tarde lideraría el proyecto Magenta de Google) presentó un modelo de Long Short-Term Memory (LSTM) capaz de aprender un estilo de música blues y componer melodías agradables en ese género [7]. Las redes LSTM ya habían mostrado ser más efectivas que las RNN estándar en tareas con datos secuenciales como los análisis en series temporales y el procesamiento del lenguaje natural. En la música no fue distinto, las composiciones generadas sonaban mejor que los modelos previos de RNNs o cadenas de Markov.

Dentro de la inteligencia artificial hay otra rama muy estudiada, que es la de los algoritmos evolutivos. Dentro de ésta también se abordó la composición de música automática. En 2010 se desarrolló Iamus, un clúster de computadoras capaz de componer piezas de música clásica contemporánea, que a diferencia de los modelos de imitación como el de Cope, tenían un estilo propio [5]. Iamus se basaba en la tecnología Melomics; un sistema que compone piezas mediante una simulación de evolución, donde piezas musicales compiten en función de una métrica de *fitness*, que evalúa criterios formales y estéticos. Las canciones eran codificadas en genomas, y mediante mutaciones y combinaciones entre miembros de la población, el sistema perfeccionaba las piezas.

Hasta el momento, todos los modelos mencionados eran específicos de música y trabajaban directamente con las notas de las canciones, utilizando principalmente archivos MIDI como datos de entrenamiento y generando composiciones en este mismo formato. Sin embargo, en 2016, DeepMind presentó *WaveNet*, un modelo basado en redes neuronales convolucionales que genera predicciones nuevas condicionadas por las anteriores (autoregresivo) [19]. A diferencia de los modelos anteriores, *WaveNet* trabaja directamente con señales de audio, lo que le permite ser entrenado tanto para la generación de música como para la síntesis de voz (*text-to-speech*, TTS). Antes de *WaveNet*, los modelos de TTS generalmente funcionaban concatenando pequeños fragmentos de sonido para formar palabras. *WaveNet* cambió este paradigma, entrenando un modelo con un gran conjunto de datos de grabaciones, logrando que pueda sintetizar habla. Si bien bajo los estándares de hoy las voces no son completamente convincentes, lograron el mejor rendimiento de su época, tanto para el inglés como para el mandarín.

El modelo también fue entrenado con música, generando fragmentos musicales realistas. De esta forma ya estamos ante modelos que combinan la tarea de la síntesis del sonido con la tarea de composición musical, y con tan solo una frase de entrada.

En 2017, el equipo de Magenta, junto a Google Brain y DeepMind, publicó un conjunto de datos llamado *NSynth* [8]. Este conjunto contiene audios de notas musicales de distintos instrumentos, con variaciones en las notas, timbres, envolventes, velocidades, etc. Junto a este conjunto introdujeron un nuevo modelo de autoencoder del estilo de *WaveNet*, entrenado directamente con ondas de audio y utilizando un decoder autoregresivo. El autoencoder se entrenó usando *NSynth*, logrando que aprenda el subespacio natural de las notas musicales, lo que permite realizar interpolaciones entre sonidos de distintos instrumentos y crear nuevos sonidos realistas y expresivos.

Mientras tanto, las redes generativas adversarias (GANs) ganaban popularidad debido a su notable éxito en la generación de imágenes [13, 4]. Estos modelos se basan en dos redes, la generadora y la discriminadora. El entrenamiento se puede modelar como un juego minimax entre dos jugadores, el discriminador busca distinguir los datos reales y los generados artificialmente (maximizar su habilidad), mientras que el generador busca crear datos indistinguibles de los reales (minimizar la habilidad del disciminador).

En este contexto, Dong et. al presentaron *MuseGAN*, un modelo basado en GANs para la generación simbólica de música de múltiples pistas [6]. El modelo fue entrenado con pistas MIDI y es capaz de componer piezas coherentes de cuatro barras sin entrada del usuario, y piezas que se adapten composiciones humanas, acompañándola con más instrumentos.

Si bien las redes convolucionales, recurrentes, los GANs y demás innovaciones ya dieron un gran impulso al auge de la inteligencia artificial, el avance más importante surgió en 2017 con la publicación "*Attention is all you need*", donde se presentó la arquitectura transformer. Los transformers están basados únicamente en mecanismos de atención, sin recurrencias ni convoluciones, lo que permite un entrenamiento más rápido gracias a su capacidad de paralelización.

Inicialmente el modelo se presentó con la tarea de traducción de texto del ingles a alemán y francés, y logró resultados sobresalientes. Dado el gran éxito, rápidamente comenzaron a surgir implementaciones para distintas tareas, en principio de procesamiento del lenguaje natural, pero luego expandiéndose a la visión por computadora con los *vision transformers*, aplicaciones multimodales y, en lo que nos concierne, al procesamiento de audio. En particular, los transformers demostraron ser efectivos para tareas de generación que requieren coherencia a largo plazo, lo que sugiere que podrían ser buenos para modelar música.

Bajo esta premisa, el equipo de Google Brain lanzó en 2018 *Music Transformer*, una variante del modelo estándar de transformers en donde la posición de cada elemento de una secuencia se representa en relación a los demás, facilitando mantener una regularidad en la estructura de la música [17]. Este modelo logró generar composiciones de hasta un minuto de duración con una estructura coherente, además de continuar piezas respetando su estilo y generar acompañamientos condicionados por melodías.

Dentro de los modelos de transformers destacan los modelos fundacionales *GPT*, grandes modelos de lenguaje (LLMs) entrenados con millones de documentos de texto, generalmente extraídos de sitios web, lo que los hace útiles para adaptarlos a tareas específicas. En 2019 se lanzó *GPT-2*, un modelo de mayor tamaño y entrenado con más datos que su predecesor. En base a la tecnología de *GPT-2*, ese mismo año OpenAI lanzó *MuseNet*, una red capaz de generar composiciones musicales de hasta 4 minutos con 10 instrumentos y combinando diversos estilos musicales.

Varios años antes de la revolución de los tranformers, en 2015 se presentó la idea de utilizar procesos de difusión para modelar la generación de imágenes sintéticas [23]. Estos modelos se entrenan para invertir un proceso de difusión, una cadena de Markov que añade ruido gradualmente a los datos hasta que la señal original queda completamente destruida. Al aprender la función inversa, el modelo es capaz de eliminar progresivamente el ruido, hasta llegar a una imagen realista. Sin embargo, a pesar de su innovación, la calidad de las generaciones no alcanzaba el nivel de los GANs y VAEs de la época, y esta técnica no tuvo un impacto inmediato.

Cinco años más tarde, en 2020, se produjo un avance significativo en los modelos de difusión, cuando Jonathan Ho et al. mejoraron la capacidad de los modelos de difusión refinando el proceso de adición y sustracción de ruido, introduciendo un nuevo objetivo de entrenamiento que mejora la estabilidad y convergencia, y utilizando arquitecturas de redes neuronales más avanzadas [15]. Gracias a todas estas mejoras, lograron demostrar que los modelos de difusión son capaces de generar imágenes de alta calidad. A partir de este trabajo la técnica se popularizó rápidamente y surgieron nuevas publicaciones con mejoras, junto a aplicaciones comerciales como *DALL-E*, *Midjourney* y *Stable Diffusion*.

Tras el éxito en la generación de imágenes, los modelos de difusión comenzaron a aplicarse a la generación de audio. La forma más intuitiva de adaptarlo es entrenando un modelo de difusión para generar espectrogramas, una representación gráfica del espectro de frecuencias a lo largo del tiempo. Esta es la idea que utilizó *Riffusion* (2022), una de las primeras aplicaciones en ofrecer generación de audio en base a prompts mediante modelos de difusión [12]. *Riffusion* reentrenó el modelo *Stable Diffusion* de Stability AI, el cual era el mejor modelo abierto hasta el momento, utilizando espectrogramas de música.

El resultado fue un modelo capaz de generar música de alta calidad, comparable con los mejores modelos autoregresivos, pero con una velocidad de inferencia significativamente más rápida.

Con la posibilidad de crear canciones de alta calidad se abrió la puerta a nuevas aplicaciones comerciales. En 2024 se lanzaron Suno AI y Udio, dos plataformas que ofrecen servicios de generación de música personalizable en géneros, letras, instrumentos, entre otros. Al ser productos comerciales, no está detallada su arquitectura y entrenamiento, pero Suno AI reveló que utiliza tanto modelos autoregresivos como modelos de difusión.

Ese mismo año, Stability AI lanzó Stable Audio, otro modelo de generación de audio con modelos de difusión, pero en lugar de generar espectrogramas, Stable Audio trabaja con representaciones de audio latentes generadas por un VAE [9]. Posteriormente, su arquitectura fue mejorada con un *transformer* de difusión en Stable Audio 2.0, permitiendo un mejor modelado para secuencias más largas de audio. Este modelo también incluía nuevas funcionalidades de generación *audio-to-audio*.

Si bien con estas aplicaciones se elevó el estado del arte de la generación de audio a niveles nunca antes vistos, al ser ofrecidos como servicios, todos los modelos son privados, inaccesibles por investigadores. Para abordar este problema, Stability AI presentó Stable Audio Open (SAO), un modelo completamente público, entrenado exclusivamente con audios bajo licencia *Creative Commons* (CC). SAO incluye el código fuente, los pesos del modelo y los algoritmos de entrenamiento y evaluación [11]. Este modelo permite generar hasta 47 segundos de audio, tanto de música como de efectos de sonido. Si bien su capacidad de generación de música y audio es inferior a la de Stable Audio 2.0, su publicación marca un gran avance en la democratización de la investigación en el campo.

Al ser un modelo tan reciente, Stable Audio Open cuenta actualmente con un único modelo reentrenado público, realizado por un usuario de HuggingFace llamado RoyalCities. Este reentrenamiento, llamado *Infinite Pianos*[1], utiliza un conjunto de datos sintetizados con tres tipos de pianos. Los audios se separaron en tres tipos, los de acordes, melodías y combinaciones de ambos, además de incorporar distintas cantidades de *tremolo* y *reverb*. El modelo parece haberse ajustado adecuadamente, mostrando una mejora en su capacidad de generar melodías y sonidos de piano, aunque el modelo original ya demostraba un desempeño relativamente bueno en ese estilo.

Meses antes del lanzamiento de SAO, un importante aporte a la investigación científica fue realizado por parte de Meta, cuando en 2023 publicó el modelo *MusicGen* [3]. A diferencia de SAO, este es un modelo autoregresivo con un decoder basado en transformers, con la posibilidad de condicionarlo tanto por texto como por melodías de audios. Aunque el modelo es capaz de generar música de buena calidad, su principal desventaja radica en el tiempo de inferencia, que es considerablemente más lento en comparación con modelos de difusión como SAO. Además, SAO demostró un mejor desempeño en la generación de música que *MusicGen*, que hasta ese momento era el mejor modelo abierto disponible. Esta capacidad se encuentra aún más pronunciada en Stable Audio 2.0, modelo basado en la misma arquitectura que SAO, resaltando el potencial que tiene SAO.

[1] https://huggingface.co/RoyalCities/RC_Infinite_Pianos

3. BASES TÉCNICAS

El concepto principal del funcionamiento de Stable Audio Open (SAO) es el de difusión, para el cual utiliza un modelo basado en transformers que trabajan sobre el espacio latente de un autoencoder variacional. A continuación explicaremos el funcionamiento de estas técnicas, junto a sus arquitecturas y las métricas que estaremos utilizando en los próximos experimentos.

3.1. Autoencoder variacional

Un **autoencoder** es una red neuronal diseñada para generar como salida exactamente lo mismo que se le provee como entrada. La clave de este proceso radica en la estructura de la red, que tiene forma de reloj de arena, con un cuello de botella en el centro donde la dimensión de los datos se reduce. Por lo tanto, el modelo debe aprender a comprimir los datos en una representación más pequeña, codificando lo más importante para poder luego reconstruir la señal original.

La primera parte de la red, responsable de reducir la dimensión, es denominada **encoder**. Luego, en base a la representación comprimida, el modelo busca reconstruir los datos originales. La segunda parte de la red, encargada de descomprimir la representación del encoder, es denominada **decoder**.

Una vez entrenado, cuando un dato pasa por el encoder, se obtiene una representación de menor dimensionalidad que contiene la esencia del dato original. El espacio en el que se encuentra el conjunto de representaciones generados por el encoder se denomina **espacio latente**.

Una limitación de los autoencoders tradicionales es que tienden a generar espacios latentes discontinuos, con representaciones agrupadas en clusters de datos similares entre sí. El problema es que al tomar un punto intermedio entre dos clusters, al decodificarlo, esperaríamos obtener un dato con propiedades de los datos de ambos clusters por igual. Sin embargo, este no es el caso, ya que las interpolaciones lineales en estos espacios latentes no son naturales debido a que su topología es mucho más compleja.

Para solucionar este problema, se introdujo el **autoencoder variacional** (VAE). A diferencia de los autoencoders tradicionales, los VAE en lugar de aprender espacios latentes discretos, representando cada dato en un punto, los VAE aprenden espacios latentes continuos, representando cada dato con un vector de medias y un vector de desviaciones estándar, modelando el espacio latente como una distribución de probabilidad, típicamente asumida como gaussiana. Esto permite generar nuevos datos similares a los de entrenamiento, ya que se puede obtener fácilmente una muestra aleatoria siguiendo las distribuciones aprendidas, generando datos sintéticos realistas. Además, la naturaleza continua del espacio latente permite interpolaciones más ricas, obteniendo tanto variaciones dentro de una clase de datos como interpolaciones entre dos clases que mantienen características fundamentales de ambas.

Stable Audio Open utiliza un autoencoder variacional que opera directamente sobre las ondas de audio crudas. Su estructura se puede ver en la Figura 3.1. El encoder está compuesto por cinco bloques convolucionales que reducen progresivamente las dimensiones de la entrada (**downsampling**) mientras aumentan el número de canales. Antes de cada

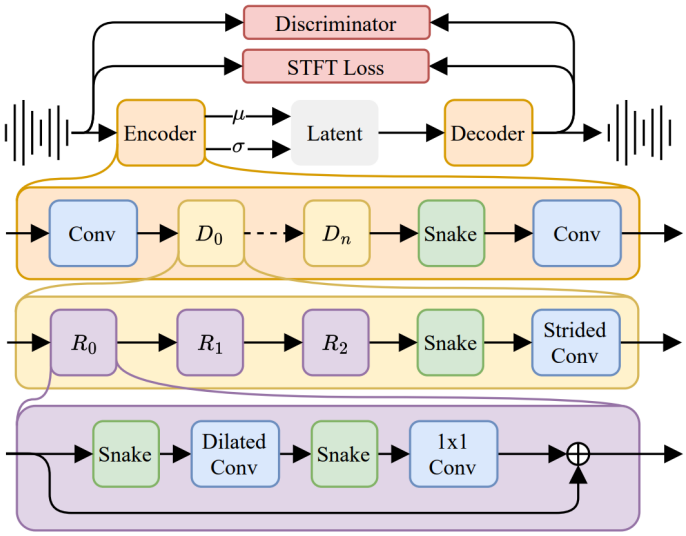


Fig. 3.1: Arquitectura del autoencoder de Stable Audio Open. Figura reproducida de Evans et al., 2024 [10]

bloque de *downsampling* se emplean una serie de capas de redes neuronales residuales, que consisten en aplicaciones sucesivas de convoluciones y de la función de activación "Snake", obteniendo una salida que se agrega a la entrada.

El proceso comienza con audio estéreo, el cual tiene naturalmente 2 canales (izquierdo y derecho) y un largo L, por lo que su dimensión inicial es $2 \times L$. Luego el encoder reduce la dimensión temporal por un factor de 1024, pero aumenta la cantidad de canales a 64, obteniendo así una representación latente de $64 \times L/1024$, lo que equivale a una compresión de datos con un factor de 32. El decoder sigue la misma estructura del encoder pero invertida, utilizando colvoluciones transpuestas para aumentar las dimensiones progresivamente (**upsampling**) y reducir los canales.

Finalmente, para comparar la salida con la entrada y entrenar para que se parezcan lo más posible, utilizan dos técnicas. La primera es un discriminador, el cual de forma similar al entrenamiento de los GANs, busca diferenciar audios reales con audios generados (en este caso por el decoder). La segunda es la *Multi-resolution STFT Loss*, la cual genera los espectrogramas de ambos audios y los compara, aplicando el proceso en M distintas resoluciones como indica la ecuación 3.1.

$$\ell_{MR}(\hat{y}, y) = \frac{1}{M} \sum_{m=1}^{M} (\ell_{SC}(\hat{y}, y) + \ell_{SM}(\hat{y}, y)) \tag{3.1}$$

Con ℓ_{SC} y ℓ_{SM} definidas en las ecuaciones 3.2 y 3.3, donde $|| \cdot ||_F$ es la norma de Frobenius, $|| \cdot ||_1$ la norma L1 y N la cantidad de segmentos en los que se divide la señal original al aplicar STFT.

$$\ell_{SC}(\hat{y}, y) = \frac{|| \, |STFT(y)| - |STFT(\hat{y})| \, ||_F}{|| \, |STFT(y)| \, ||_F} \tag{3.2}$$

$$\ell_{SM}(\hat{y}, y) = \frac{1}{N} || \, log(|STFT(y)|) - log(|STFT(\hat{y})|) \, ||_1 \tag{3.3}$$

3.2. Difusión y métodos de guía

El método generativo de difusión se basa en revertir un proceso gradual de adición de ruido. Se comienza con una muestra de ruido x_T y paso a paso se van generando muestras con menos ruido $x_{T-1}, x_{T-2}, ...$ hasta llegar a una muestra final x_0. Cada paso corresponde a una cantidad de ruido, y la muestra x_t se interpreta como una mezcla de la muestra x_0 con ruido ε, el cual se asume que sigue una distribución gaussiana. Por lo tanto, el modelo aprende a predecir el ruido de una muestra x_t, el cual puede ser eliminado de la muestra para obtener la muestra x_{t-1}. En la publicación original, el modelo se parametriza como la función $\varepsilon_\theta(x_t, t)$, la cual predice el ruido de la muestra x_t.

Durante el entrenamiento, se elige aleatoriamente una muestra x_0 del conjunto de datos, se selecciona un paso t, y se le aplica el ruido correspondiente a x_0, obteniendo así x_t. El objetivo del entrenamiento entonces es $\|\varepsilon_\theta(x_t, t) - \varepsilon\|^2$, el error cuadrático medio entre el ruido real y el ruido predicho.

Con esta técnica, inicialmente se lograron generar imágenes de calidad comparable a la de los mejores modelos de la época. Sin embargo, las imágenes se generaban a partir de ruido, sin ninguna condición específica, lo que limitaba el control sobre el contenido generado.

Para lograr un control sobre la generación, se introdujo la técnica de **classifier guidance**, que consiste en entrenar un clasificador multiclase $p_\phi(y \mid x)$, el cual toma muestras x como entrada y devuelve la distribución de probabilidad sobre las clases y. Luego, si se desea generar imágenes de una clase y_{target}, en cada paso del proceso de difusión se calcula cómo cambiar x_t para que el clasificador aumente su confianza en la clase deseada. Esto se logra calculando el gradiente del logaritmo de la probabilidad asignada por el clasificador a la clase deseada, con respecto a la muestra actual: $\nabla_{x_t} \log p_\phi(y_{target} \mid x_t)$, de forma similar a lo que se haría durante el entrenamiento para un dato de la clase deseada. Ahora, en lugar de utilizar este gradiente para actualizar los pesos del modelo, se usa el gradiente de la imagen para actualizarla en la dirección de la clase deseada. Al combinar este gradiente con el ruido predicho por el modelo de difusión, obtenemos la siguiente función de actualización:

$$\hat{\varepsilon}(x_t, t) = \varepsilon(x_t, t) - w \cdot \sigma_t \cdot \nabla_{x_t} \log p_\phi(y_{target} \mid x_t) \tag{3.4}$$

Donde $\varepsilon(x_t, t)$ es el ruido predicho por el modelo de difusión, σ_t es un factor que ajusta la magnitud de la guía en función de la cantidad de ruido, y w es un parámetro conocido como `guidance_scale`, el cual controla la influencia del clasificador. Con valores muy altos de w, el modelo se ajustará completamente a la clase, pero será más determinístico, obteniendo muestras con poca variedad. Con valores muy bajos de w, el ajuste a la clase será menor, generando muestras ambiguas o que mezclan diferentes clases. El objetivo es encontrar un balance adecuado para este parámetro.

La principal desventaja del *classifier guidance* es que requiere entrenar un clasificador adicional para poder controlar lo que se genera. No se puede aprovechar modelos previamente entrenados, ya que deben ser entrenados con datos ruidosos. Además, el modelo de clasificación solo puede controlar la generación para las clases que aprendió a reconocer.

Para solucionar este problema, se introdujo la técnica de **classifier-free guidance**, que elimina la necesidad de un clasificador [16]. Esta técnica consiste en entrenar el modelo de difusión para predecir el ruido condicionado por una clase y (con técnicas de atención que veremos luego), pero también para predecirlo sin ninguna condición (utilizando el token *null* o un embedding vacío).

Durante la inferencia, en cada paso de difusión, el modelo predice el ruido condicionado por la clase deseada $\varepsilon_\theta(x_t, t \mid y_{target})$ y el ruido sin condicion $\varepsilon_\theta(x_t, t \mid \emptyset)$, y luego combina ambas predicciones de la siguiente forma:

$$\hat{\varepsilon}(x_t, t) = \varepsilon(x_t, t \mid \emptyset) + w \cdot (\varepsilon_\theta(x_t, t \mid y_{target}) - \varepsilon_\theta(x_t, t \mid \emptyset)) \tag{3.5}$$

Donde w nuevamente es el `guidance_scale`, que controla la influencia de la clase en el resultado final.

Con esta técnica, no es necesario entrenar otro modelo adicional además del modelo de difusión, pero el proceso de entrenamiento debe ser modificado levemente. Antes de entrenar, se define el hiperparámetro p_{uncond}, el cual determina la probabilidad de ignorar la clase en cada paso de entrenamiento. De esta forma, en algunos pasos de entrenamiento se entrena con la clase y en otros sin condicion, permitiendo que el modelo aprenda ambos tipos de generación en paralelo.

Una de las principales ventajas de la *classifier-free guidance* es que el condicionamiento no necesariamente tiene que ser una clase de un clasificador, sino que puede ser cualquier cosa. En particular, se puede usar un embedding de texto de un modelo como CLIP, el cual está entrenado para generar representaciones de texto y de imágenes en un espacio latente común, generando representaciones similares para pares ⟨imagen, texto⟩ en donde el texto describe la imagen. De esta forma, usando *classifier-free guidance* podemos condicionar el proceso difusión con frases en lenguaje natural.

3.3. Transformers

Los **transformers** son un tipo de red neuronal introducido en 2017 [25] que revolucionó el campo del *deep learning*, siendo fundamentales en el auge de los grandes modelos de lenguaje (LLMs) y otras aplicaciones como la difusión. A continuación, explicaremos cómo funciona esta arquitectura y los conceptos necesarios para entender el funcionamiento de Stable Audio Open.

En cualquier modelo basado en transformers, la entrada se divide en partes pequeñas denominadas *tokens*. Por ejemplo, si la entrada es texto, los tokens suelen ser combinaciones de dos o tres letras, similares a las sílabas. Cada token se representa con un vector, lo que da lugar a una secuencia de vectores que representan el texto. En el caso de imágenes o audio, la entrada se divide en parches (submatrices), que luego se aplanan en un vector unidimensional y se tratan como tokens. Así, independientemente de la modalidad, la entrada de un transformer siempre será una secuencia de vectores.

El componente principal de un transformer es su **mecanismo de atención**. Intuitivamente, este mecanismo actualiza cada uno de los vectores de la secuencia de entrada en función de los demás. El objetivo de esta actualización es que cada vector represente mejor su significado utilizando el contexto proporcionado por los otros vectores. Por ejemplo, en un modelo de lenguaje, palabras con múltiples significados como “llama” (que puede ser un animal, una flama o un modelo de lenguaje), pueden ajustar su representación basándose en las demás palabras del contexto, diferenciando los significados posibles y obteniendo una representación más rica.

Las operaciones que se realizan y los parámetros que se aprenden en una capa de atención están representados por tres matrices. Las primeras dos son la *query matrix* Q y la *key matrix* K. Intuitivamente, la matriz Q guarda las preguntas que se hacen sobre los datos, y la matriz K proporciona la respuesta a estas preguntas para cada uno de los tokens. Supongamos que comenzamos con los vectores de representaciones $E_1, ..., E_n$. Las primeras multiplicaciones se realizan entre los vectores de representación E_i y la matriz Q, obteniendo nuevos vectores $Q_1, ..., Q_n$, y con la matriz K, obteniendo nuevos vectores $K_1, ..., K_n$, los cuales viven en un espacio dimensional más pequeño. Luego, para cada par de vectores (Q_i, K_j) se calcula el producto punto, obteniendo una matriz de valores. A esta matriz se la divide por $\sqrt{d_k}$, la raíz cuadrada de la dimensión del espacio donde viven los Q_i y los K_i, para mayor estabilidad numérica. Posteriormente, se aplica la función de activación softmax.

Este proceso nos da una matriz que, de forma intuitiva, indica, para cada vector de representación E_i, qué tanto debe influír cada otro vector E_j en la actualización de su significado. Aquí es donde entra en juego la tercera matriz, la *value matrix* V, que codifica la información real que se va a agregar a cada embedding, es decir, cómo va a cambiar su significado. Para ello, se multiplica la matriz V con los E_i, obteniendo vectores $V_1, ..., Vn$, que indican la dirección en la que se debe actualizar cada vector. Finalmente, para cada vector E_i, sabemos qué dirección debe seguir según la influencia de cada otro vector E_j, y qué tanto debería influír cada uno. Entonces, se suman esas direcciones, ponderadas por el peso calculado previamente, y se obtiene la actualización de cada vector de nuestra entrada.

Este proceso se resume en la famosa fórmula 3.6 presentada en la publicación original:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{3.6}$$

Dado que este procedimiento simplemente actualiza los valores de los vectores, luego de la aplicación de una capa de atención, el tamaño de la salida es el mismo que el de la entrada. Una vez actualizados todos los vectores, cada vector pasa por un mismo perceptrón multicapa (MLP) de forma independiente. El MLP genera una salida del mismo tamaño que los vectores de entrada. De esta forma, al aplicarlo a cada vector independientemente, la salida que obtenemos luego de todas las aplicaciones es del mismo tamaño que la entrada.

El proceso de actualización de las representaciones utilizando mecanismos de atención basados en las matrices de *Query, Key* y *Value* recibe el nombre de una **attention head**. Sin embargo, en lugar de realizar una sola actualización con un único head, surgió la idea de **multi-head attention**, que consiste en ejecutar varias heads en paralelo, cada una con sus propios parámetros y enfocándose en diferentes aspectos de los datos de entrada. Los cálculos para cada head se efectúan de manera independientemente mediante el mecanismo de atención clásico ya descrito. Una vez obtenidas las salidas de todas las heads, estas se concatenan y procesan para generar la actualización final de las representaciones de entrada.

Esta técnica no solo mejora significativamente la capacidad de representación de los modelos, sino que también introduce paralelismo en los cálculos, ya que cada head opera de manera independiente. Esto permite entrenar rápidamente estos modelos aprovechando las capacidades de procesamiento masivo de las tarjetas gráficas modernas.

Hasta el momento, los mecanismos descritos permiten actualizar una entrada en base a sí misma, permitiendo que las diferentes partes de la entrada utilicen las demás como contexto para mejorar sus representaciones. Sin embargo, en muchos casos es necesario actualizar las representaciones de una entrada en función de otra fuente de información. Por ejemplo, en la traducción de texto, se desea actualizar las representaciones de una oración en un idioma con las representaciones de otra en un idioma diferente como contexto, o en modelos multimodales, buscamos actualizar la representación de una imagen o un audio utilizando como contexto un prompt que lo describa.

Para abordar estos casos, se introdujo el mecanismo de **cross-attention**. Su funcionamiento es esencialmente el mismo que el de la atención estándar, con la única diferencia que la matriz Q se aplica sobre una secuencia conocida como la **decoder sequence**, y las matrices K y V se aplican sobre otra secuencia, referida como la **encoder sequence**. De esta forma se logra condicionar las nuevas representaciones de un dominio con información proveniente de otro dominio.

3.4. Transformers de difusión

Previamente a la irrupción de los transformers, las redes convolucionales tradicionales dominaban las tareas de clasificación y generación de imágenes. Aunque los transformers fueron introducidos en 2016, inicialmente estaban diseñados para el procesamiento de texto. No fue hasta 2020 que se produjo un gran avance en el reconocimiento de imágenes, cuando Dosovitskiy et al. presentaron los **Vision Transformers** (ViTs), adaptando el modelo de transformers para procesar imágenes. Para utilizar el mecanismo de atención con imágenes, se emplea la técnica de dividirlas en parches de tamaño fijo y aplanarlos en vectores. Poseriormente, de la misma forma que en el caso del texto, se genera una representación de estos vectores, buscando que aquellos parches con información similar queden cercanos en el espacio lineal obtenido.

Debido a que los transformers procesan todos los tokens en paralelo, se pierde la información de la posición de cada token, un aspecto crucial. Para solucionar esto, se utilizan los **positional embeddings**. Uno de los métodos más comunes se basa en frecuencias, generando una serie de funciones de seno y coseno sobre el espacio de entrada y sumándoselas a esta. Esto introduce un patrón único y suave en cada parte de la entrada, permitiendo al modelo captar la posición relativa de cada token y, por ende, entender mejor la estructura subyacente de los datos.

En cuanto a las arquitecturas generativas dedicadas a la técnica de difusión, incialmente también empleaban redes convolucionales, basadas en U-Net. Aunque se realizaron mejoras en los modelos de difusión a lo largo del tiempo, el uso de la arquitectura U-Net se mantuvo constante. Esto cambió en 2023, cuando se demostró que la arquitectura U-Net no era esencial para el alto rendimiento de los modelos de difusión, y que estos también podían beneficiarse del uso de los transformers [21]. En esta publicación, Peebles et al. se basaron en los ViTs para introducir la arquitectura denominada **difussion transformers** (DiTs). Al igual que los ViTs, este modelo opera sobre secuencia de parches, a los que se les aplica un embedding posicional basado en frecuencias.

Una vez que la entrada se encuentra representada en vectores, se pueden aplicar bloques de transformers de manera convencional. Sin embargo, como vimos anteriormente, en los modelos de difusión, además de la entrada principal, es necesario incorporar el paso de difusión t y el embedding de la guía c (si se usa *classifier-free guidance*).

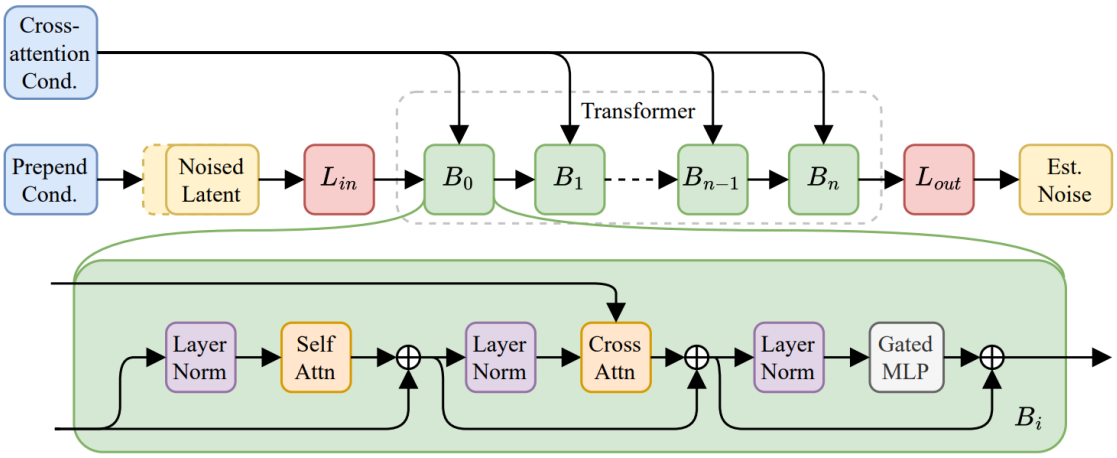


Fig. 3.2: Arquitectura del difussion transformer de Stable Audio Open. Figura reproducida de Evans et al., 2024 [10]

Para integrar esta información en los DiTs existen diversas técnicas, de las cuales destacamos dos, que son las utilizadas por Stable Audio Open (SAO):

1. Agregar t y c como tokens adicionales en la secuencia de entrada.
2. Luego de la capa de self-attention, añadir una capa de *multi-head cross-attention*, en la que los tokens de entrada se actualizan en base a una secuencia que contiene los elementos t y c.

Si bien Stable Audio Open no trabaja con imágenes, sino que con representaciones de audio generadas por un autoencoder, su arquitectura es exactamente la de los ViTs clásicos. En este modelo, se busca condicionar a las generaciones en base a tres señales:

1. El prompt, escrito en texto natural, el cual describe lo que se busca que suene en el audio. La forma de incluírlo es generando un embedding de texto, en este caso usando el modelo preentrenado T5-base, el cual genera vectores de dimensión 768.
2. El *timing*, el cual permite generaciones de longitud variable, entrenando al modelo para llenar la señal hasta donde el *timing* lo indique.
3. El paso de difusión, representado mediante embeddings sinusoidales.

La arquitectura exacta se detalle en la Figura 3.2. Comienza concatenando la información de condicionamiento, que incluye el paso de difusión y el timing, a los vectores de entrada. Estos vectores pasan luego por bloques de transformers $B_0, ..., B_n$. Dentro de estos bloques, primero se realiza una actualización con self-attention, donde los vectores se actualizan utilizando a los demás como contexto. Posteriormente, se aplica una capa de cross-attention, en la que los vectores se actualizan en base a la secuencia que contiene el embedding de texto generado con T5-Base y el timing. Cada uno de estos pasos incluye *skip connections*, mecanismo por el cual la salida de la capa se suma al input inicial, ya que es la idea base de multi-head attention. Además, al inicio de cada capa de atención se aplica una normalización para estabilizar y acelarar el entrenamiento.

3.5. Métricas

3.5.1. Estimación de tempo

Una de las tareas que realizaremos en los experimentos es la estimación de tempo sobre los audios generados. Las métricas que utilizaremos para evaluar qué tan bien se ajusta el modelo al tempo solicitado son las de *Accuracy 1* y *Accuracy 2*, definidas por F. Gouyon et al. [14] en el contexto de evaluación de algoritmos de predicción de tempo. En ese caso de uso, se conoce el tempo original de cada audio, y se lo compara con las predicciones de los algoritmos. Las medidas se definen de la siguiente manera:

- **Accuracy 1:** Porcentaje de predicciones de tempo que se encuentran dentro de un 4 % (la *ventana de precisión*) del tempo real del audio.
- **Accuracy 2:** Porcentaje de predicciones de tempo que se encuentran dentro de un 4 % del tempo real del audio, o del doble, triple, la mitad o un tercio del mismo.

La medida Accuracy 1 es intuitiva, mide la cercanía de las predicciones al valor real. Por otro lado, Accuracy 2 está definida de esa manera ya que frecuentemente las discrepancias entre las predicciones y el valor real correspondían a un enfoque en una métrica distinta, por ejemplo, un múltiplo de 2 o 1/2 en música con métrica binaria, o un múltiplo de 3 o 1/3 en música con métrica ternaria. En nuestro caso de uso, estas medidas son relevantes por dos razones: primero, utilizaremos un predictor de tempo sobre los audios generados, que puede mostrar ese tipo de discrepancias; segundo, incluso si el predictor de tempo fuera perfecto, también es razonable permitir, por ejemplo, una generación de 80 BPM en un prompt que indica 160 BPM, ya que la generación de una muestra con ese valor sigue siendo adecuada para la inclusión en una composición musical.

3.5.2. Métricas basadas en representaciones

Para evaluar las generaciones de audio de un modelo a partir de un dataset de pares ⟨audio, prompt⟩ se pueden utilizar diversos modelos de clasificación y embeddings, tanto aquellos que integran representaciones de audio y texto en un espacio conjunto, como aquellos específicos para audio. A partir de distintos modelos de embeddings se definen distintas métricas. Las dos primeras propuestas por Stable Audio Open, mientras que la última es propia de esta tesis:

- **PaSST**: Este modelo surgió como una optimización de la arquitectura de transformers para trabajar con espectrogramas, utilizando una técnica denominada *patchout*, la cual consiste en eliminar partes de la entrada del transformer durante el entrenamiento [18]. Entre las tareas en las que mejoró el estado del arte se encuentra la de predicción multiclase, en particular la clasificación de audios en las categorías definidas en el conjunto de datos "Audioset"[1]. En base a este modelo se define la métrica KL_{passt}, que consiste en calcular el vector de probabilidades para los audios del conjunto de datos y para las generaciones del modelo, midiendo su similitud mediante la divergencia de Kullback-Leibler (KL). Una vez calculada para cada generación de audio, el valor final se obtiene con un promedio de las divergencias.

[1] https://research.google.com/audioset/dataset/index.html

- **CLAP**: Previamente hablamos de CLIP, un modelo que genera representaciones de texto e imágenes en un espacio latente compartido, logrando que los embeddings de los textos queden cerca de los embeddings de las imágenes que describen. CLAP aplica esta técnica al dominio del audio [27]. Este modelo entrena simultáneamente un encoder de texto y otro de audio, utilizando un método llamado **contrastive learning**, el cual consiste en presentarle pares positivos (textos que describen correctamente al audio) y pares negativos (textos que no lo hacen), buscando maximizar la similitud entre los embeddings de los pares positivos y minimizar la de los negativos. En base a este modelo se define la métrica CLAP_{score}, la cual consiste en generar la representación del prompt y la representación del audio generado, y calcular la similitud coseno. El resultado final se obtiene con un promedio entre todas las similitudes. Existen distintos modelos preentrenados de CLAP, SAO definió la métrica utilizando `630k-audioset-fusion-best`.

- **EnCodecMAE**: Este modelo combina EnCodec, un codificador neuronal de audio, con un **masked autoencoder**. Primero, genera una representación del audio mediante EnCodec, luego le aplica una máscara que descarta partes aleatorias del embedding. Las partes restantes pasan por un encoder y son expandidas al tamaño original. Luego, se añaden embeddings posicionales para informale al decoder la posición de las secciones que debe reconstruír, y finalmente el decodificador procesa la secuencia resultante. Este proceso genera representaciones de audio universales, adecuadas para diferentes tipos de audios como habla, música y sonidos de ambiente. En base a este modelo, definimos las métricas ECMAE_{score} 1 y ECMAE_{score} 2, las cuales se calculan computando las representaciones del audio original y del audio generado, y calculando su similitud de coseno. El valor final se obtiene con un promedio entre todas las similitudes. La diferencia entre la métrica 1 y 2 radica en el modelo preentrenado utilizado, la primera emplea `ec-ec-large-st` y la segunda `mel256-ec-base-st-fma`.

4. ENTRENAMIENTO DEL MODELO

El reciente modelo Stable Audio Open (SAO) ha demostrado una capacidad notable en la tarea de generación de efectos de sonido. Sin embargo, su desempeño en la generación de música presenta más limitaciones. Esto se debe a que fue entrenado con datos bajo licencia *Creative Commons*, restringiendo la cantidad de datos musicales de alta calidad disponibles.

En este experimento buscamos mejorar su capacidad de generación de música. El modelo actual es incapaz de generar audios con ciertos instrumentos y de incluir exactamente los instrumentos solicitados en el prompt, muchas veces añadiendo otros o excluyendo alguno de los mencionados. Por otro lado, el modelo sólo reconoce algunos géneros musicales y con frecuencia genera música que no se alinea con el estilo especificado en el prompt. Tampoco está ajustado para seguir aspectos técnicos musicales, por ejemplo, si se incluye en el prompt un tempo o una tonalidad, las generaciones suelen discrepar con lo esperado. Todas estas limitaciones son las que buscamos mejorar en este experimento.

Además, debido a que SAO es un modelo reciente, existe escasa documentación sobre su flujo de entrenamiento, consumo de recursos y capacidad de adaptación a nuevos datos, por lo tanto en este experimento también buscamos profundizar en estos aspectos para futuras personalizaciones.

4.1. Creación del conjunto de datos

El conjunto de datos utilizado para entrenar este tipo de modelos consiste en pares $\langle \text{audio}, \text{prompt} \rangle$, donde el prompt describe, en lenguaje natural, qué es lo que está sonando en el audio. Lo más influyente en la complejidad de la creación del dataset es el prompt. Actualmente es sencillo encontrar grandes cantidades de música de distintos géneros e instrumentos, sin derechos de autor, gracias a plataformas como YouTube. Sin embargo, esto sólo alcanza para conseguir el audio, acompañado con el título del vídeo, que generalmente es el nombre de la canción, y la descripción del video, que no es frecuente que contenga información del aspecto sonoro del vídeo. Por lo tanto, si bien se podría conseguir fácilmente el audio, los títulos y las descripciones son insuficientes para generar prompts que describan con suficiente detalle lo que suena en el audio. Para generar un conjunto de datos de esta forma, tendríamos que anotar manualmente los aspectos relevantes del audio, metodología que resulta inviable para la gran cantidad de audio necesaria para entrenar un modelo de este tamaño.

Teniendo en cuenta que descargar los audios no es factible, otra posibilidad es sintetizarlos. Esta metodología es la que finalmente utilizamos para generar nuestro conjunto de datos. Al tener control total sobre lo sintetizado, podemos obtener directamente los metadatos necesarios que indiquen los distintos aspectos de los audios generados. Dado que es posible sintetizar audios mediante código de Python, y que la recolección de los metadatos es trivial, ya que es información presente en el proceso de síntesis, todo el procedimiento puede ser definido en un algoritmo, permitiendo generar grandes cantidades de audio y metadatos sin anotadores humanos.

En esta parte hablamos de “metadatos” en vez de hablar directamente de “prompts” porque la información que se puede obtener al sintetizar un audio es información estructurada. Cada audio que se sintetiza es acompañado por una tabla u objeto con los atributos y parámetros que se eligen a la hora de sintetizar el audio. Usando esta información se podría generar directamente un prompt. Por ejemplo, en el caso del dataset usado para el reentrenamiento “Infinite Pianos”, el cual también está sintetizado, se generaron los prompts simplemente concatenando todos los metadatos, separándolos con comas, obteniendo así prompts de la forma:

`[Piano Type], [Modifier][Chord Progression], [Melody Type], [Key], [FX], [BPM], [Bar Count]`

Esta es una forma efectiva y rápida para convertir los metadatos a prompts, pero obliga a los usuarios del modelo a escribir los prompts de una manera absolutamente estructurada para obtener el resultado esperado. Esto creemos que atenta contra la esencia de este tipo de modelos, que es poder describir con lenguaje natural lo que se busca que suene. Tener que escribir los diferentes modificadores en un orden específico, y tener valores prefijados para cada modificador es una experiencia más similar a usar directamente el algoritmo de síntesis a usar un modelo que genera audio en base a prompts. Por esta misma razón, decidimos evitar ese método, y optamos por usar una de las herramientas de inteligencia artificial más prolíficas del momento, que son los *Large Language Models* (LLMs). Con esta herramienta creamos un asistente que transforma los archivos que contienen los metadatos estructurados en descripciones que contengan la misma información, pero escrita de una forma más humana, con adjetivos y conectores que integran todo en un texto coherente, más similar al uso normal que le daría una persona al modelo.

Habiendo explicado el proceso de generación de audio de manera superficial, veamos un poco más en detalle cada paso. En la Figura 4.1 se puede observar el proceso que lleva al cabo el algoritmo de síntesis de audio y generación de prompts.

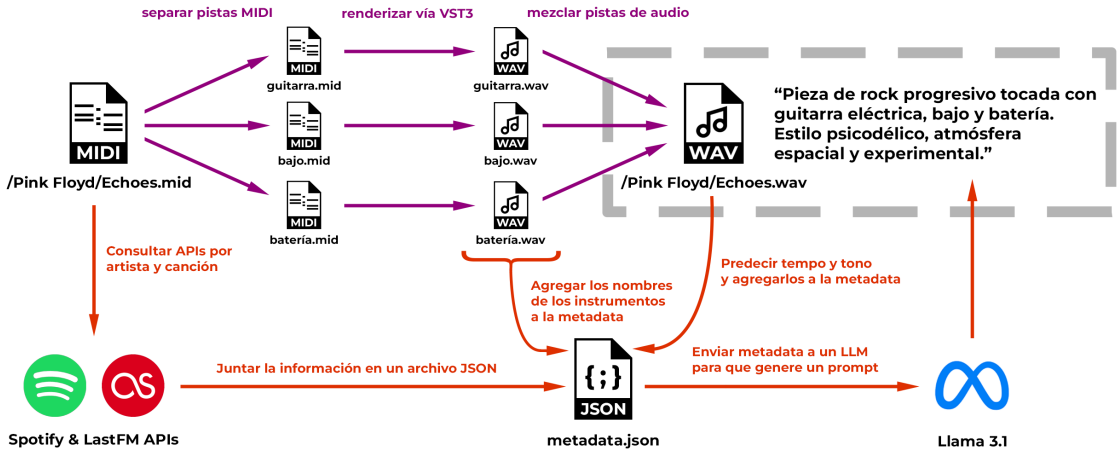


Fig. 4.1: Flujo de datos para la generacion de pares $\langle \text{audio}, \text{prompt} \rangle$. Por arriba y con flechas violetas se ve la síntesis del audio, desde los archivos MIDI hasta el archivo de audio final. Por debajo y con flechas naranjas, la generación del prompt en base a los metadatos y resultados de ejecutar algoritmos sobre el audio sintetizado.

4.1.1. Síntesis del audio

Como se observa en la Figura 4.1, el proceso comienza con un archivo MIDI. Recordemos que estos archivos no son de audio, por lo que no son suficientes para entrenar el modelo; los archivos MIDI deben ser renderizados. La fuente que elegimos para los archivos MIDI es un conjunto de datos ampliamente reconocido y usado en la literatura: "The Lakh MIDI Dataset v0.1". En el conjunto de datos completo, los autores aclaran que no llevaron a cabo ningún proceso de limpieza para eliminar archivos MIDI inválidos y que, por lo tanto, contiene unos miles de archivos corruptos. Sin embargo, proporcionan un subconjunto llamado "Clean MIDI subset", en el cual se aseguraron de eliminar los archivos inválidos. Este es el subconjunto que utilizamos como base para sintetizar los audios. Este conjunto contiene un total de $17,257$ archivos MIDI, organizados en carpetas etiquetadas con el nombre del artista, y el nombre del archivo corresponde al nombre del título de la canción. Sobre estos archivos se efectuó un proceso de limpieza y agregado de metadatos extra, que comprende los siguientes puntos:

1. **Eliminación de duplicados:** Este proceso fue relativamente sencillo, ya que si una canción tenía un duplicado, había un archivo con su mismo nombre pero con el sufijo "`.1`", "`.2`", ..., "`.n`". Sin embargo, también se detectaron casos de archivos que correspondían a la misma canción, pero que en uno de los archivos alguna de sus palabras estaba en mayúscula y en el otro en minúscula. Por ejemplo, "`Here Comes the Sun.mid`" y "`Here comes the sun.mid`". Como se puede ver, en esos casos son archivos duplicados pero ninguno está marcado con el sufijo ".1" al final. Para considerar estos casos, se normalizaron los nombres de los archivos convirtiéndolos a minúsculas, lo que permitió identificar y eliminar duplicados adicionales. El criterio para seleccionar un archivo ante otro fue priorizar siempre el archivo con más pistas MIDI, ya que resultará en un archivo de audio con más instrumentos, más completo y realista. Durante este proceso se eliminaron un total de $6,901$ duplicados.

2. **Separación en múltiples archivos:** Como mencionamos anteriormente, los archivos MIDI contienen potencialmente múltiples pistas, una por instrumento. A la hora de renderizar el audio, nos interesa renderizar cada instrumento por separado, por lo tanto separamos cada archivo MIDI en múltiples archivos que contienen una sola pista con un sólo instrumento. Para esto utilizamos una biblioteca de Python para el manejo de archivos MIDI llamada `pretty_midi`. En el proceso también se eliminaron 101 archivos MIDI, aquellos que esta biblioteca no pudo abrir, probablemente por algún error del archivo.

Una vez finalizada la limpieza del conjunto de datos, disponemos de múltiples archivos MIDI para ser renderizados. Inicialmente, optamos por utilizar una herramienta conocida para este propósito, llamada `fluidsynth`, un sintetizador que genera sonido a partir de archivos MIDI o eventos en tiempo real, mediante el uso de *soundfonts* (archivos `.sf2`). Estos *soundfonts* contienen información de un instrumento basada en muestras de audio. `Fluidsynth` posee una comunidad relativamente grande, por lo que existe una gran variedad de *soundfonts* de distintos instrumentos, grabados por diferentes personas.

Aunque logramos desarrollar un prototipo funcional utilizando `fluidsynth`, decidimos finalmente migrar a instrumentos virtuales (`VSTs`). La razón del cambio fue la mayor flexibilidad que ofrecen los VSTs en comparación a los *soundfonts*, ya que no dependen de las muestras de audio proporcionadas por los creadores de los *soundfonts*. Los instrumentos virtuales permiten la síntesis real de audio, es decir, generar algorítmicamente el sonido, lo cual ofrece un control más preciso sobre el diseño del sonido que se producirá al tocar cada nota. Además, suelen incorporar efectos como *reverb*, *delay*, distorsión o ecualización. Todo esto resulta en un poder expresivo mucho más alto que los *soundfonts*.

Si bien los instrumentos virtuales suelen usarse dentro de una estación de trabajo de audio digital (DAW), utilizamos una bibloteca de Python desarrollada por Spotify, llamada `pedalboard` [22]. Esta herramienta permite cargar instrumentos virtuales, enviarles eventos MIDI y generar audios a partir de ellos. Además, permite el control de los distintos parámetros específicos de cada instrumento virtual, lo que nos permitió aleatorizar algunos de estos parámetros, generando múltiples variaciones de un mismo instrumento y evitando que suenen de forma idéntica en cada ejecución. Usando la biblioteca `mido`, desarrollamos un algoritmo que convierte los archivos MIDI en una lista de eventos tal como espera el VST cargado en `pedalboard`.

El último paso es la mezcla de los audios renderizados en una sola pista. Previamente a eso, notamos que el volumen de salida de cada pista variaba significativamente; algunos instrumentos eran excesivamente fuertes, mientras que otros apenas se percibían. Dado que esas variaciones no respondían a ningún criterio armónico o sonoro, decidimos normalizar el volumen antes de mezclarlos. Para ello utilizamos una implementación de la recomendación ITU-R BS.1770 para medir el volumen percibido de señales de audio, la cual disponibilizaron en un paquete de Python llamado `pyloudnorm` [24]. Tras la normalización, las pistas fueron mezcladas mediante un promedio simple de los audios. Finalmente, el archivo de audio fue segmentado en secciones más pequeñas, siguiendo un criterio que mencionaremos posteriormente.

Este proceso cubre la parte superior de la Figura 4.1, que ilustra la síntesis de audio a partir de archivos MIDI hasta la obtención del archivo final de audio (`.wav`). Ahora, pasemos a describir la generación de los prompts que acompañan cada archivo de audio.

4.1.2. Generación del prompt

Como mencionamos previamente, los archivos MIDI están etiquetados con los nombres de las canciones y los artistas. Además, prácticamente todos los archivos corresponden a canciones populares, por lo tanto fue posible obtener información adicional a través de fuentes externas. En particular, la API de Spotify y la API de LastFM:

1. **API de Spotify:** Utilizamos la ruta `/audio-features` para consultar características musicales de las canciones, tanto técnicas (tonalidad, escala y tempo), como experimentales y perceptuales (*energy*, *acousticness*, *instrumentalness*). Además, la ruta `/audio-analysis` nos proporcionó una segmentación de las canciones en "secciones". Según su descripción, las secciones están definidas por grandes variaciones en el ritmo o timbre de la canción, por ejemplo estribillo, estrofa, puente, solo de guitarra, etc. Esta segmentación es de gran utilidad, ya que los archivos MIDI suelen representar canciones completas, con duraciones de entre 2 y 4 minutos, mientras

que para nuestros fines de entrenamiento requerimos de fragmentos más cortos, de unos 30 segundos aproximadamente. En lugar de cortar los audios aleatoriamente, lo que resultaría en audios sin mucho sentido musical, que empiezan en mitad de frases instrumentales o que terminan repentinamente, aprovechamos la segmentación en secciones que nos provee Spotify para cortar los audios generados en partes que tienen sentido musical y son coherentes.

2. **API de LastFM:** Aunque Spotify nos ofrece algunos datos perceptuales, estos son fijos y no alcanzan para describir correctamente qué es lo que suena en el audio. La API de LastFM soluciona este problema, complementando los metadatos con una lista de etiquetas escritas y asignadas por usuarios a las canciones. Estas etiquetas son de gran utilidad porque incluyen los géneros principales de las canciones, cosa que Spotify no te brinda, pero además contiene adjetivos descriptivos, como por ejemplo "*melancholic*", "*warm*", "*gentle*". Estas etiquetas son esenciales para enriquecer la descripción de los audios, ya que incluyen adjetivos que serían imposibles de inferir solo a partir del archivo MIDI o del audio renderizado.

   Observamos que algunas etiquetas referenciaban a los cantantes o las voces que sonaban en las canciones. Como nuestros renders no contienen vocales, esas etiquetas fueron eliminadas en base a reglas básicas de lenguaje. Por ejemplo, eliminamos aquellas que contenían "vocal" como "*vocalist*", "*vocals*", y aquellas que contenían "singer", "voice", "lyrics", etc.

Las canciones que no fueron encontradas en las APIs de Spotify o LastFM fueron eliminadas del conjunto de datos de archivos MIDI. Esto eliminó un total de $1,180$ archivos. Entre la limpieza previa y esta, el conjunto de datos de archivos MIDI pasó de tener $17,257$ archivos a tener $9,075$, una diferencia significativa, pero que no genera ningún problema ya que siguen siendo una cantidad adecuada.

Además de los datos de fuentes externas, añadimos información usada durante el proceso de síntesis. Por ejemplo, registramos cada pista de cada instrumento que se renderizó, para poder tener en los metadatos la lista de instrumentos que están presentes en el audio. También decidimos realizar nuestras propias predicciones de tempo, tonalidad y escala, ya que la información proporcionada por Spotify son predicciones realizadas sobre la canción original y entera, mientras que nuestros audios son extractos más cortos de versiones renderizadas usando un subconjunto de los instrumentos originales de la canción, sintetizados a partir de un archivo MIDI escrito por alguna persona/algoritmo, por lo tanto hay muchos puntos posibles de fallo o propagación de error que pueden hacer que la información de Spotify no sea correcta.

La predicción del tempo fue realizada usando un modelo de redes convolucionales llamado `deeprhythm`. Esta elección se basó en un experimento que se puede ver en el Anexo 5.1, en el cual resultó mas preciso que otro predictor. La predicción de tonalidad y escala fue realizada usando la implementación de `essentia [1]` de un algoritmo que analiza el espectro de las señales de audio, luego encuentra los picos espectrales y estima el *Harmonic Pitch Class Profile* usando el *HPCP algorithm*, para finalmente estimar el tono y escala basándose en su HPCP usando el *Key algorithm*.

Al combinar los datos de las APIs con las predicciones de `Deeprhythm` y `essentia`, obtenemos, para cada audio generado, un archivo `.json` con la siguiente estructura:

```
{
  "Song": "Here Comes The Sun - Remastered 2009",
  "Artist": "The Beatles",
  "Tags": [
    "sunshine pop",
    "rock",
    "60s",
    // ...
  ],
  "Instruments": [
    "Acoustic Guitar",
    "Drums"
  ]
  "duration_ms": 185733,
  "acousticness": 0.9339,
  "energy": 0.54,
  "key": "A",
  "mode": "Major",
  "tempo": 128,
  "sections": [
    {
      "start": 0.0,
      "duration": 16.22566
    },
    // ...
  ]
}
```

Esta información ya es suficiente para generar un prompt que describe el audio de forma coherente y detallada. Para crear un asistente que transforme los archivos JSON a un prompt que parezca escrito por un humano, usamos un prompt de sistema en el cual se le indica al LLM el trabajo que tiene que hacer, y se describe cada parte los metadatos que recibirá. A la hora de entregarle el prompt de usuario (los metadatos) no se le envía el archivo JSON directamente, sino que en base a él se escribe un texto estructurado en ítems, Por ejemplo, con el JSON previo el prompt de usuario queda de la siguiente manera:

```
- The synthetized song is 'Here Comes The Sun' by 'The Beatles'.
- Its tags are: sunshine pop, rock, 60s, ...
- The instruments playing are: Acoustic Guitar and Drums
- It's an acoustic song.
- It's in the key of A Major, at 128 BPM.
```

Un desafío que enfrentamos al generar los prompts fue la tendencia a seguir una estructura rígida, basada en el orden en el que le presentábamos los atributos en el prompt de sistema. Por ejemplo, si primero explicábamos la lista de etiquetas, luego la lista de instrumentos, y finalmente los aspectos técnicos, el LLM generaba prompts que mantenían ese mismo orden:

```
A sunshine pop/rock song of the 60s, driven by acoustic guitar and drums,
        set in the key of A major, with a lively tempo of 128 BPM.
```

Este es un comportamiento similar al que mencionamos anteriormente que queríamos evitar. No buscamos que los prompts sigan una estructura fija, sino que los usuarios tuvieran la flexibilidad de redactar de manera natural lo que quieren generar. Para abordar este problema, implementamos prompts dinámicos, tanto para el sistema como para el usuario. Esta estrategia consiste en variar aleatoriamente el orden de la explicación de cada atributo en el prompt de sistema, y cambiar el orden de los atributos en el prompt de usuario. Como resultado, logramos generar prompts más diversos: algunos comenzaban con las etiquetas, otros con los instrumentos y otros con los aspectos técnicos.

Otra técnica que utilizamos es la de *few-shot-learning*, que consiste en proporcionarle al LLM unos pocos ejemplos de entradas y salidas deseadas para mejorar su rendimiento. Aunque resulta complicado cuantificar el impacto exacto de esta técnica, observamos que previamente a utilizarla, en ciertos casos los BPM indicados en prompt generado no coincidían con los presentes en los metadatos. Sin embargo, tras aplicar *few-shot-learning*, no se volvió a dectar ese tipo de discrepancia, lo cual sugiere una mejora tangible en la precisión del LLM.

De esta forma queda cubierta la parte inferior de la Figura 4.1, la cual ilustra el proceso de generación del prompt que describe el archivo audio utilizando los distintos metadatos recopilados. No obstante, al generar dichos prompts, observamos que algunos incluían instrumentos que no sonaban en el audio. Esto se debe a que los audios fueron separados en secciones, y cada una heredó los metadatos del audio completo, en particular la lista de instrumentos. Sin embargo, en ciertas secciones algunos instrumentos presentes en el audio completo no aparecían, lo que resultaba en prompts finales que incluían instrumentos que no sonaban en el audio. Para solucionar este inconveniente y además facilitar otras operaciones sobre el conjunto de datos, optamos por desarrollar una aplicación web que ayude con la creación del conjunto de datos.

4.1.3. Aplicación web

El algoritmo de generación de pares $\langle \text{audio}, \text{prompt} \rangle$ explicado previamente produce un directorio con archivos de audio, nombrados en base a un ID numérico, junto a archivos JSON que comparten el mismo nombre que su archivo de audio correspondiente. Hasta el momento, la visualización y modificación del conjunto de datos se realizaba mediante la interfaz propia del sistema de archivos y un entorno de desarrollo (IDE), lo cual no era conveniente. Para simplificar este proceso y abordar el problema de los instrumentos mencionado previamente, decidimos desarrollar una aplicación web, cuya interfaz puede verse en la Figura 4.2.

Como se puede observar en la Figura 4.2, en la parte izquierda de la interfaz se encuentra un menú desplegable, que permite filtrar los audios tanto por género musical como por instrumento. Una vez seleccionada una categoría, a la derecha se muestra una lista

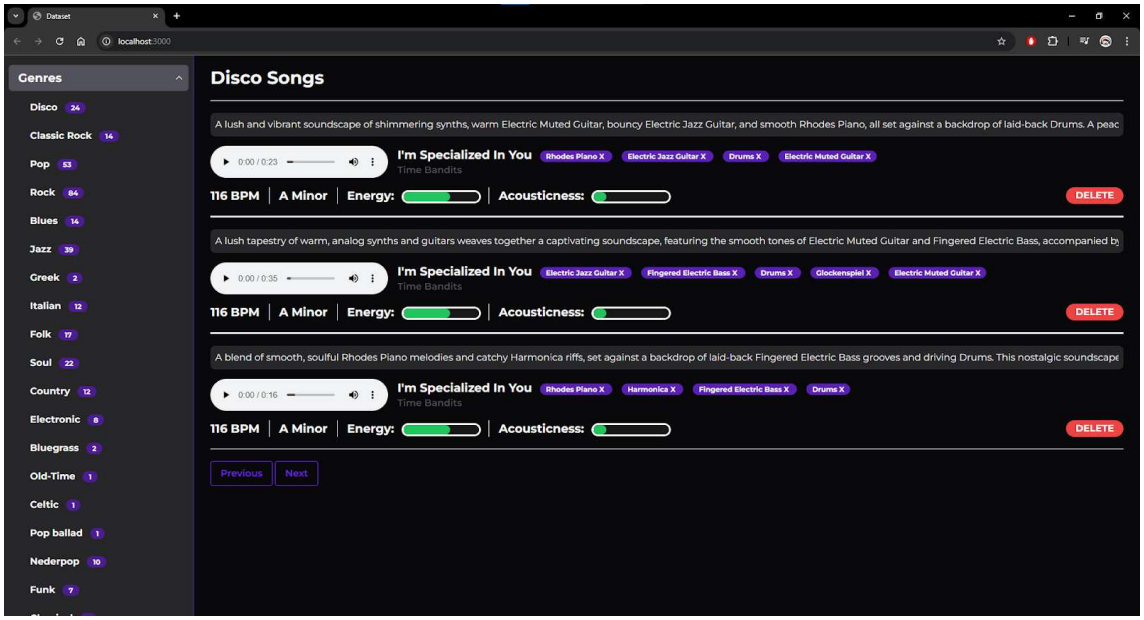


Fig. 4.2: Interfaz gráfica de la aplicación web desarrollada para la visualización y edición de los datos de entrenamiento.

de audios generados, correspondientes a dicha categoría y organizados en páginas. Por cada elemento de la lista, en la parte superior se encuentra el prompt asignado al audio, permitiendo leerlo y editarlo fácilmente. Justo debajo, se encuentra el reproductor para escuchar el audio, junto con el nombre de la canción y artista. Al lado del nombre de la canción, se presenta una lista de etiquetas que corresponde a la lista de instrumentos. Cada una de estas etiquetas tiene una cruz, permitiendo eliminar fácilmente alguno de los instrumentos de los metadatos del audio. Debajo se muestra más información sobre los metadatos del archivo, como su tempo, tonalidad y escala. Finalmente se encuentra un botón que permite eliminar tanto el archivo de audio como el archivo con sus metadatos.

En cuanto al desarrollo de esta herramienta, el *backend* fue implementado en `Node.js` sin paquetes adicionales, dado que no requeríamos de ninguna funcionalidad muy compleja. Si bien paquetes como `Express` podrían haber simplificado el desarrollo de la API, optamos por evitar dependencias innecesarias para facilitar la instalación y uso del sistema: basta con clonar el repositorio y ejecutar el comando `node server.js` para tener la aplicación funcionando.

La API desarrollada dispone de una ruta llamada `/audiosMetadata`, que mediante el verbo `GET` retorna un objeto con los metadatos de todos los audios generados. Luego, tanto los archivos JSON como los WAV son accesibles en la ruta con su mismo nombre, mediante el verbo `GET`. La única diferencia entre ellos es que los archivos de audio, a diferencia de los JSON que el servidor los retorna enteros, son transmitidos por fragmentos, lo que reduce el uso de memoria, optimizando la carga varios de ellos en la interfaz en un instante. En el caso de los archivos JSON, también pueden modificarse mediante el verbo `PATCH`, lo que permite editar los prompts desde la interfaz, y eliminar instrumentos. En ambos casos se envía simplemente un objeto indicando el atributo que se quiere modificar y el valor nuevo a la ruta correspondiente. Finalmente, los archivos JSON pueden eliminarse mediante el

verbo `DELETE`, lo que resultará en la eliminación tanto del archivo JSON como del audio correspondiente.

En cuanto al *frontend*, no presentó mayor complejidad. Al cargar la página, se realiza un `GET` a la ruta que provee los metadatos, y en base a esa información se organizan los audios en las distintas categorías y se crean los menús desplegables. Para el CSS utilizamos el *framework* `Tailwind`, que provee clases predefinidas que se pueden usar directamente en el HTML.

Utilizando el esquema previamente detallado para generar el conjunto de datos, es posible producir rápidamente una cantidad significativa de minutos de audio. Sin embargo, debido al problema en el etiquetado de los instrumentos y la generación ocasional de audios de baja calidad, es necesario realizar una supervisión, lo cual ralentiza el proceso. A pesar de esto, alcanza para la cantidad de audio requerida, de aproximadamente 8 horas. Siguiendo esta metodología, comenzamos a construir el conjunto de datos para entrenar el modelo, hasta que llegamos a las 4 horas de material. En este momento, antes de seguir generando datos sin evaluación intermedia, y ante incertidumbres como la capacidad del modelo para adaptarse a estos audios, la suficiencia de recursos para el entrenamiento, entre otras cuestiones, decidimos efectuar un entrenamiento preliminar para despejar estas dudas.

4.2. Entrenamiento preliminar

Para este entrenamiento preliminar efectuado sobre el modelo SAO disponible públicamente, contábamos con un total de 440 audios con una duración promedio de 30 segundos, sumando un total de 4 horas de material. Debido a los elevados requerimientos de recursos necesarios para entrenar modelos de esta envergadura, es la primera tarea que resulta inviable de realizar localmente. Por lo tanto decidimos utilizar `Google Colab`, el cual ofrece, mediante suscripción, un entorno con 83.5 GB de memoria RAM, y una tarjeta gráfica A100 con 40 GB de memoria.

En dicho entorno, descargamos el conjunto de datos generado y comenzamos con el entrenamiento, cuya configuración es relativamente sencilla, dado que *Stability* proporciona un repositorio[1] con *scripts* en Python que permiten realizar el entrenamiento a medida, simplemente ajustando ciertos parámetros.

Inicialmente, comenzamos el entrenamiento con un *batch size* de 4, pero al verificar que los recursos de cómputo disponibles eran suficientes, lo incrementamos posteriormente a 8. Para esta prueba preliminar, el modelo se entrenó durante una hora, completando $2,000$ pasos, en 36 épocas. La memoria de GPU utilizada se mantuvo constante en 31.5 GB.

Tras finalizar el entrenamiento, realizamos un análisis de cualitativo. Generamos audios a partir de distintos prompts utilizando tanto el modelo reentrenado como el original, para realizar una comparación auditiva. En estos resultados notamos una clara tendencia hacia el estilo de los audios de entrenamiento. Con solo 4 horas de audio y una hora de entrenamiento, el modelo ya mostraba una mayor calidad de sonido que el original, sugiriendo que con un mayor volumen de datos y tiempo de entrenamiento el resultado va a ser el esperado.

Con este modelo no calculamos ninguna métrica objetiva ya que el objetivo del entrenamiento era simplemente explorar el proceso de entrenamiento y verificar que el modelo realmente es capaz de cambiar su estilo, obteniendo resultados exitosos en ambas tareas.

[1] https://github.com/Stability-AI/stable-audio-tools

4.3. Conjunto final

Una vez resueltas las dudas previas al entrenamiento preliminar, y habiendo obtenido resultados satisfactorios, procedimos a completar la generación del conjunto de datos. Antes de continuar, decidimos dividirlo en dos conjuntos: el polifónico (compuesto por audios con múltiples instrumentos), y el monofónico (compuesto por audios de un sólo instrumento), ya que buscamos que el modelo pueda generar ambos tipos de audios. Al realizar esta división, observamos que el conjunto monofónico presentaba una cantidad limitada de audios, la mayoría proveniente de pianos, con escasa o nula representación de otros instrumentos. Para balancear este conjunto, generamos entre 5 y 15 minutos de audio para cada instrumento, resultando en un total de 300 audios, con una representación equitativa de cada instrumento. En cuanto al conjunto polifónico, utilizamos el mismo proceso de generación empleado en el entrenamiento preliminar, lo que nos permitió alcanzar los 400 audios.

Los audios generados al renderizar pistas MIDI poseen un sonido característico, que suele percibirse como artificial, tanto por la manera en la que se ejecutan las notas como por la mezcla, que no es tan compleja como la de una grabación real. En el entrenamiento preliminar el modeló demostró adaptarse a la generación de este tipo de sonidos. No obstante, nuestro objetivo no es replicar ese sonido artificial. Por esta razón, decidimos crear un tercer conjunto de datos, basado en audios instrumentales sin derechos de autor provenientes de YouTube. Como mencionamos al inicio de este capítulo, estos audios son de fácil recolección, pero requieren de un etiquetado manual en cuanto a aspectos perceptuales y subjetivos. Sin embargo, dado que ya contamos con una cantidad significativa de audios en los otros conjuntos, no buscamos generar un conjunto completo con esta metodología manual, lo que sería inviable. Generar una proporción más pequeña es posible, y nos permite guiar al modelo hacia una sonoridad distinta a la de audios MIDI.

Para la creación de este conjunto se descargaron audios instrumentales de diversos géneros y con diferentes instrumentos. Los instrumentos presentes en cada audio se registraron manualmente. Luego, para la lista de etiquetas, se solicitó a un LLM una serie de adjetivos descriptivos asociados a los géneros y estilos respectivos; estos se asignaron a los audios de cada género en función de su afinidad, de manera de asegurar una mayor diversidad en las descripciones y evitar sesgos en la forma en la que describimos la música nosotros mismos. La predicción de tempo, tonalidad y escala se realizó de la misma forma que la empleada en los conjuntos anteriores. Esos metadatos fueron posteriormente enviados al LLM para generar los prompts finales.

El entrenamiento final se realizó utilizando los 3 conjuntos de datos, que en la Tabla 4.1 se explicita su tamaño y duración.

	Conj. Monofónico	Conj. Polifónico	Conj. YouTube	Total
# audios	298	399	326	1023
Duración (avg.)	28.4 seg.	32.4 seg.	33.6 seg.	-
Duración total	141 min.	215 min.	182 min.	538 min. (9h)

Tab. 4.1: Información de la duración de los tres conjuntos de datos generados para el entrenamiento final, incluyendo la cantidad de audios de cada uno, la duración promedio los audios y la duración total de cada conjunto.

Dado que el ajuste al tempo, la tonalidad y la escala es una cualidad esperada en modelo final, es interesante observar su distribución sobre los 1023 audios de entrenamiento. Esto se observa en las Figuras 4.3 y 4.4.

En cuanto al tempo, los audios se encuentran en el rango de $[75, 160]$ BPM, con dos modas claras: una alrededor de los 90 BPM y otra en torno a los 120 BPM. Respecto a la tonalidad, observamos una mayor representación de la notas naturales por sobre las alteradas, con la excepción de Si (B), que presenta pocas apariciones, y La# (A#), que tiene una mayor presencia. En relación a la escala, predominan los audios en escala mayor, aunque también se dispone de una cantidad suficiente en escala menor.

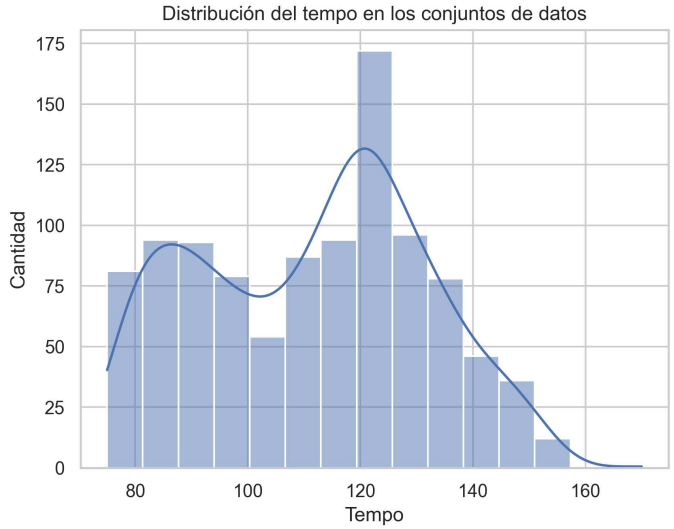
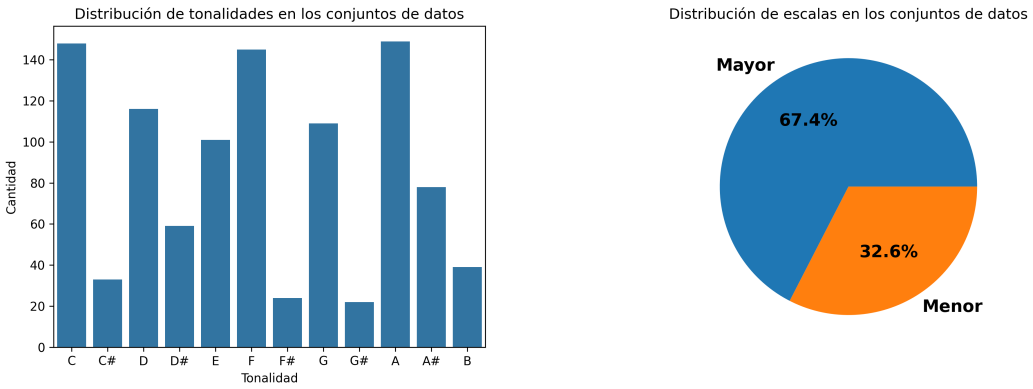


Fig. 4.3: Distribución del tempo medido en los 1023 audios de entrenamiento.



(a) Distribución de la tonalidad.

(b) Distribución de la escala.

Fig. 4.4: Distribuciones de la tonalidad y escala medida en los 1023 audios de entrenamiento.

4.4. Entrenamiento

Con los tres conjuntos de datos, alcanzando más del doble de duración respecto al entrenamiento preliminar, procedimos al entrenamiento completo, el cual ejecutamos durante un total de 5 horas, nuevamente en la gráfica A100.

En el entrenamiento anterior aumentamos el *batch size* a 8, pero los recursos seguían con poder de cómputo por explotar, por lo tanto en este caso lo incrementamos nuevamente a un valor final de 16. En este caso la memoria utilizada por la GPU se mantuvo en torno a los 33 GB. Luego de las cinco horas de entrenamiento se alcanzaron los $4,000$ pasos en 63 épocas, completando así el entrenamiento principal de este primer experimento.

Al modelo final obtenido de ahora en más será referido como "Instrumental Finetune"

4.5. Resultados

Una vez completado el entrenamiento, las primeras evaluaciones realizadas fueron auditivas. Se generaron audios a partir de prompts nuevos y prompts presentes en el conjunto de entrenamiento, lo que permitió evaluar tanto el grado de generalización como el de memorización. Dichas generaciones están disponibles para escuchar dentro del repositorio en el que se encuentra el modelo[2].

Los resultados de esta evaluación fueron exitosos. Se observó una mejora inmediata en la calidad general del sonido, tanto en la mezcla y la definición del audio generado como en los aspectos melódicos y de composición de las canciones. En cuanto a los instrumentos, el modelo mostró una capacidad significativamente mejorada para sintetizar aquellos que el modelo original no era capaz de generar. Además, respondió de manera más precisa la especificación de instrumentos de los prompts, en los ejemplos del respositorio se puede ver que en casos donde se le piden trompetas o saxofones, el Instrumental Finetune produce audios que los incluye, mientras que el modelo original falla en incluirlos y en lugar suenan otros instrumentos o los ignora.

En los géneros musicales también se ve evidenció una mejora notable, generando audios muy característicos de cada género. Incluso en los géneros comunes la esencia que los distingue está más marcada en el Instrumental Finetune, pero esta mejora se evidencia aún más en géneros específicos, donde el modelo original producía audios completamente distintos a los esperados.

Respecto a la memorización, al comparar los audios generados con prompts del conjunto de entrenamiento frente a los audios que refieren, observamos que las nuevas generaciones eran similares pero no idénticas. No encontramos ninguna instancia de memorización por parte del modelo, sugiriendo que realmente aprendió los patrones, permitiendo producir variaciones dentro de los estilos deseados.

En segunda instancia, evaluamos los aspectos técnicos de las generaciones para analizar si el modelo aprendió a adaptarse al tempo y la tonalidad cuando estos son especificados en el prompt. Asimismo, generamos audios utilizando prompts del "Song Describer Dataset" (conjunto que el modelo nunca vio) y comparamos las generaciones con los audios originales de dicho dataset mediante las métricas de embeddings previamente presentadas. De esta forma comparamos el desempeño del modelo ante este conjunto de datos frente al modelo SAO original y a MusicGen. A continuación se detallan cada uno de estos resultados.

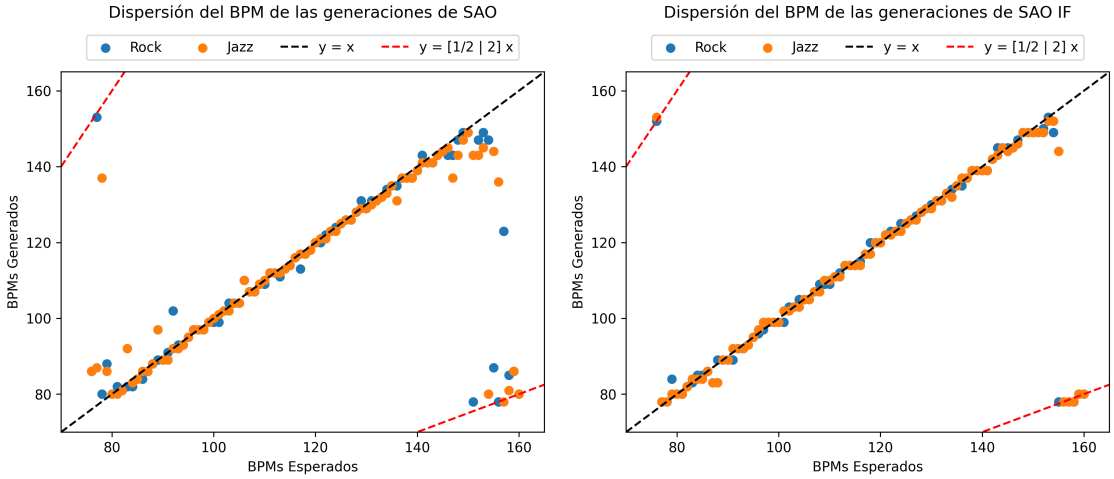
[2] https://huggingface.co/santifiorino/SAO-Instrumental-Finetune

4.5.1. Ajuste al tempo

Para evaluar en qué medida los audios generados se ajustan al tempo especificado en el prompt, se generaron dos prompts para cada valor en el rango $[75, 160]$, uno solicitando una canción de rock, y otro solicitando una de jazz. Esta selección de géneros diferentes entre sí permitió analizar si el grado de ajuste al tempo dependía del género, dado que cada género tiende a seguir una distribución de tempo específica.

Como métricas de evaluación utilizaremos Accuracy 1 y Accuracy 2, detalladas previamente en la Sección 3.5.1, las cuales miden el porcentaje de estimaciones dentro de un rango del valor real, y dentro de un rango de múltiplos del valor real.

A partir de los audios generados por SAO y por el *Instrumental Finetune*, se estimó el tempo de ambos usando `deeprhythm`. La Figura 4.5 muestra los gráficos de dispersión para ambos modelos, y en cada figura particular, la dispersión por cada género.



(a) Dispersión de los audios generados por Stable Audio Open (SAO).

(b) Dispersión de los audios generados por el Instrumental Finetune.

Fig. 4.5: Comparación de la dispersión del tempo esperado contra el tempo generado por ambos modelos ante distintos prompts, que requerían tempos entre 60 y 160 BPM de los géneros de Rock y Jazz.

La Figura 4.5 sugiere que SAO (4.5a) ya presentaba un ajuste razonable al tempo, aunque en los intervalos $(75, 90)$ y $(140, 160)$, se observan varias desviaciones, que no aparecen en el *Instrumental Finetune* (4.5b). Además, los errores no parecen depender del género: entre los audios generados con tempos mayores o menores que los solicitados, se encuentran audios de rock y de jazz en proporciones similares.

	Accuracy 1	Accuracy 2
Stable Audio Open (SAO)	0.747	0.776
SAO Instrumental Finetune	0.876	0.953

Tab. 4.2: Métricas de Accuracy 1 y Accuracy 2 para Stable Audio Open (SAO) e Instrumental Finetune, sobre 200 generaciones de audios a partir de prompts que nunca antes vieron.

En la Tabla 4.2 se muestran las medidas de accuracy introducidas previamente, usando una ventana de un 2 %. Los resultados muestran una mejora significativa: tanto en Accuracy 1 como en Accuracy 2 la diferencia entre SAO y el Instrumental Finetune fue notable. Considerando que para un uso cotidiano de generación de muestras para incorporar a producciones musicales el Accuracy 2 nos indica la proporción de audios útiles (en cuanto a tempo), nuestro modelo generó audios satisfactorios en un 95 % de las pruebas.

Un aspecto visible en la Figura 4.5 es que ambos modelos alcanzan un rendimiento casi perfecto en el rango $(100, 140)$. Sin embargo, en los intervalos iniciales y finales mencionados previamente, es donde surgen los problemas, especialmente para SAO. En esos intervalos el Instrumental Finetune pudo hacer la mayor diferencia en cuanto a Accuracy, ya que prácticamente no tiene valores fuera de la recta $y = x$. Sin embargo, alrededor de los tempos 80 y 160, se pueden ver algunas desviaciones, pero en ambos casos están sobre las rectas $y = (1/2)x$ o $y = 2x$, lo cual explica la mejora del Instrumental Finetune al relajar el criterio del Accuracy 1 con el de Accuracy 2.

Para profundizar en la idea de la independencia de género observada, calculamos los Accuracies de ambos modelos sólo en los prompts que pedían por audios de Rock y lo mismo para el Jazz. Como sugería la Figura 4.5, los valores se mantuvieron prácticamente constantes en todos los casos.

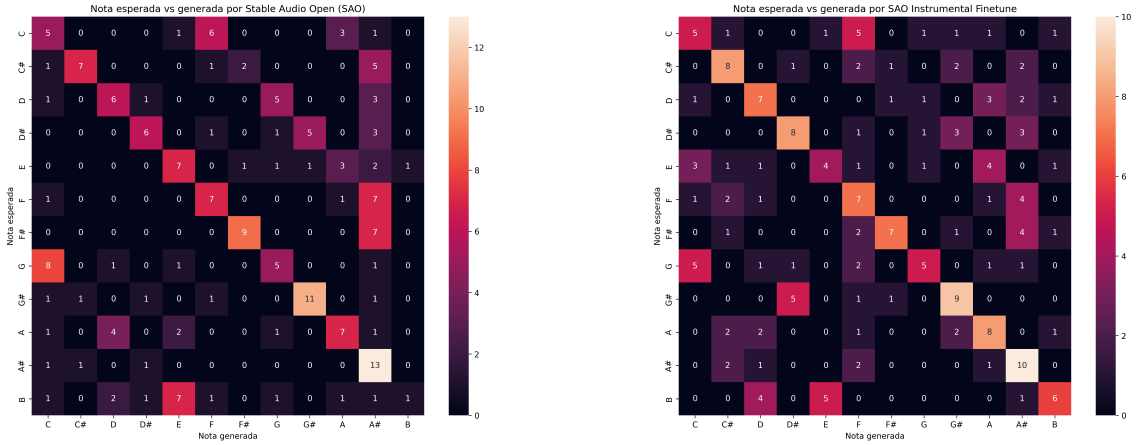
4.5.2. Ajuste a la tonalidad y escala

Para analizar el grado de ajuste a la tonalidad y escala, se llevó a cabo un experimento análogo al realizado para el análisis de tempo. Para cada combinación de nota y escala (producto cartesiano entre {C, C#, D, ..., A#, B} y {menor, mayor}), generamos ocho prompts: dos de rock, dos de jazz, dos de hip hop y dos de pop. Luego se generaron los audios correspondientes a dichos prompts utilizando tanto SAO como el Instrumental Finetune. A continuación, se efectuó la predicción de tonalidad y escala usando `essentia`. Dado que la proporción de nota y escala está perfectamente balanceada, en la Tabla 4.3 se presentan los valores de accuracy clásico, tanto sobre la nota y escala como únicamente sobre la nota. Para evaluar el desempeño en las generaciones de cada nota, la Figura 4.6 muestra las matrices de confusión obtenidas por ambos modelos.

	Accuracy de nota y escala	Accuracy de nota
Stable Audio Open (SAO)	0.21	0.44
SAO Instrumental Finetune	0.27	0.44

Tab. 4.3: Resultados de accuracy obtenidos al comparar la nota y escala (o únicamente la nota) de los audios generados por ambos modelos con la nota esperada (especificada en el prompt).

A diferencia del análisis de tempo, donde ambos modelos mostraron un buen rendimiento, en este caso ambos modelos presentaron una baja adaptación a la tonalidad y escala requeridas en los prompts. Como se observa en la Tabla 4.3, ambos modelos presentaron exactamente el mismo accuracy cuando se evalúa únicamente sobre la nota. No obstante, la matriz de confusión del modelo reentrenado (4.6b) exhibe una distribución de aciertos más equilibrada en comparación con la de SAO (4.6a), en la cual, por ejemplo, ante la solicitud de generar audios en la tonalidad de Si (B), SAO generó un solo audio en esa tonalidad.



(a) Matriz de confusión de los audios generados por Stable Audio Open (SAO), el modelo sin reentrenar.

(b) Matriz de confusión de los audios generados por el Instrumental Finetune, el modelo reentrenado.

Fig. 4.6: Comparación de las matrices de confusión para la nota (tonalidad) esperada y obtenida en las generaciones de los modelos Stable Audio Open y el Instrumental Finetune.

En cuanto a la nota y escala en conjunto, el accuracy del modelo reentrenado (0,27) mejoró levemente con respecto al accuracy del modelo original (0,21). Dado que ambos modelos tienen el mismo nivel de precisión en la nota, este resultado sugiere que el modelo reentrenado muestra una mayor precisión en la escala cuando la nota generada es la esperada.

Ademas, es importante tener en cuenta que el algoritmo utilizado para obtener la nota y escala es otro posible punto de falla, pudiendo el mismo confundir comúnmente una nota con otra. Para minimizar el impacto de esto, con un análisis del algoritmo de estimación se podría definir una métrica similar al Accuracy 2 del tempo, pero para las notas.

En cuanto a los géneros, a diferencia del análisis del tempo, donde los géneros no influyeron en el rendimiento, en este caso sí se observó una mayor variación de accuracy según el género. Por ejemplo, el accuracy del modelo sin reentrenar sobre la tonalidad y escala en audios del género de hip hop es de 0,125, mientras que en el género de rock alcanza 0,27. En el caso del modelo reentrenado, también se nota esta varianza: el accuracy en el género hip hop es de 0,21, mientras que en el género de rock asciende a 0,31.

El bajo desempeño general en esta tarea puede atribuirse a la naturaleza del modelo, que trabaja con representaciones de audio, en la que las notas, sus relaciones y las nociones fundamentales de teoría musical no están explícitamente definidas. El aprendizaje de las notas y escalas se realiza de manera implícita, una desventaja clara ante modelos que, en lugar de generar música *end-to-end*, producen archivos MIDI, permitiendo un control directo sobre las notas. Además, mantener la tonalidad a lo largo de toda la secuencia representa un desafío significativo, especialmente en generaciones polifónicas. Cuando hay varios instrumentos presentes, la tarea es más compleja ya que la desviación de un sólo instrumento puede provocar que la tonalidad de toda la pieza cambie. Esto queda evidenciado al comparar con el modelo reentrenado Infinite Pianos, diseñado para generar música monofónica. En este caso, al estar limitado al piano, el modelo logra un mejor control sobre la tonalidad, según menciona su desarrollador.

4.5.3. Métricas basadas en representaciones

Como un último experimento, utilizamos los modelos de representaciones de audio presentados en la Sección 3.5.2 para comparar las generaciones del Instrumental Finetune con las de SAO y MusicGen. Para las métricas KL_{passt} y CLAP_{score} utilizamos el repositorio `stable-audio-metrics`, desarrollado por Stability AI, que permite calcular fácilmente los valores de estas métricas a partir de audios generados por el modelo.

El cálculo de estas métricas requiere de pares $\langle \text{audio}, \text{prompt} \rangle$ que no hayan sido utilizados durante el entrenamiento, ya que estos se considerarán como las referencias "correctas" u obetivo a la hora de comparar las generaciones de los distintos modelos. Para este propósito, utilizamos un subconjunto de 586 pares del Song Describer Dataset, definido en el repositorio mencionado previamente. A partir del prompt de cada par, generamos los audios utilizando cada uno de los modelos, y luego calculamos las métricas.

	$\text{KL}_{passt} \downarrow$	$\text{CLAP}_{score} \uparrow$	ECMAE_{score} 1 $\uparrow$	ECMAE_{score} 2
MusicGen	0.52	0.31	-	-
Stable Audio Open (SAO)	0.54	0.39	0.20	0.24
SAO Instrumental Finetune	0.52	0.38	0.20	0.24

Tab. 4.4: Comparación de MusicGen, Stable Audio Open (SAO) y SAO Instrumental Finetune, en base a métricas basadas en representaciones, utilizando un subconjunto del Song Describer Dataset como objetivo.

En la Tabla 4.4 se presentan los valores obtenidos para cada métrica y modelo. Comenzando con las métricas definidas por Stability AI, KL_{passt} indica que el Instrumental Finetune, con un valor de 0,52, mejoró con respecto a SAO, el cual obtuvo un valor de 0,54. Este resultado iguala al de MusicGen, el mejor modelo abierto previo a SAO en esta métrica. Por otro lado, la métrica CLAP_{score} sugiere que el Instrumental Finetune empeoró levemente, pasando de 0,39 a 0,38, aunque seguiría generando audios significativamente más fieles al prompt en comparación a MusicGen, que obtuvo un valor de 0,31.

En cuanto a las métricas específicas desarrolladas para esta tésis, los valores obtenidos por SAO y el Instrumental Finetune fueron idénticos. No se calcularon los valores de MusicGen debido al elevado costo computacional que implica generar la cantidad de audios necesarios para el análisis.

Los valores tan bajos de estas métricas (0,20 y 0,24, lejos de 1) pueden atribuirse a que la definición de la métrica, de comparar directamente con similitud coseno las representaciones de EnCodecMAE no alcanza para capturar aspectos como el género musical, instrumentos o estilo general de los audios. Entrenando un clasificador de género con las representaciones se podría mejorar esta métrica.

La invariabilidad observada en todas las métricas basadas en representaciones puede explicarse por la diferencia entre los prompts del Song Describer Dataset y los generados en nuestro conjunto de datos. Los primeros son genéricos y menos detallados, por ejemplo "`A danceable electronic track in the genre of dance`", mientras que los segundos incluyen información más específica como los instrumentos, el tempo o la tonalidad.

De todas formas, este resultado es interesante ya que sugiere que al reentrenar el modelo con un conjunto de datos más especializado, no se pierden las capacidades para generar contenido en el contexto más amplio y generalizado, con el cual fue entrenado el modelo originalmente.

5. CONCLUSIONES

En este trabajo buscamos mejorar las capacidades de generación musical de Stable Audio Open (SAO), el modelo de generación de audio abierto más reciente y con mejor rendimiento. Nuestro enfoque consistió en reentrenar el modelo utilizando un conjunto de datos especializado. Para la creación de este conjunto desarrollamos un pipeline de generación de música y prompts automático, el cual parte de archivos MIDI con múltiples pistas por instrumento, las sintetiza usando instrumentos virtuales, mezcla las pistas en un único archivo de audio, normaliza su volumen y genera un prompt a partir de metadatos obtenidos de distintas APIs (Spotify y LastFM), junto a estimaciones de tempo y tonalidad. Finalmente, dicha información es procesada por un modelo de lenguaje para producir un prompt en lenguaje natural, simulando la entrada que se esperaría de un humano.

A partir de este pipeline generamos un gran volumen de audios, balanceados en términos de instrumentación y monofonía/polifonía. Los audios fueron posteriormente filtrados manualmente mediante aplicación web que desarrollamos para inspeccionar, modificar y eliminar las muestras. Para evitar el sonido característico de las canciones tocadas a partir de archivos MIDI, también se incluyeron muestras de música sin derechos de autor de YouTube. El resultado fue un conjunto curado de 9 horas de audio.

Con este dataset, reentrenamos el modelo original durante 5 horas, en un entorno de ejecución con una tarjeta gráfica A100, utilizando un *batch size* de 16, el cual mantuvo constante una utilización de 33 GB de GPU. Luego de las cinco horas se efectuaron 4000 pasos en 63 épocas, obteniendo nuestro modelo reentrenado "Instrumental Finetune".

Finalmente, nuestro modelo fue evaluado con distintas métricas, comparándolo con SAO y MusicGen. En los resultados de escucha notamos una gran mejora, obteniendo audios de mejor calidad de sonido, mezcla, composición y adherencia a los instrumentos y géneros solicitados. Al contrastar el tempo del audio generado con el tempo solicitado en el prompt, notamos que el modelo mejoró notablemente la adherencia al mismo. Sin embargo, no notamos lo mismo con la tonalidad y escala. Con las métricas basadas en representaciones observamos que el entrenamiento en un conjunto de datos especializado no generó la pérdida de capacidades generales del modelo.

5.0.1. Trabajo futuro

- **Especializar aún más:** Si bien el conjunto de datos generado en la tesis es considerablemente más específico que el utilizado en el entrenamiento de SAO, se pueden crear conjuntos aún más especializados para el reentrenamiento del modelo. Por ejemplo, un conjunto de composiciones de tango de un pequeño conjunto de artistas, tocadas por un pequeño conjunto de bandas. Sobre un conjunto así, se puede evaluar por un lado qué tanto capta la esencia de las composiciones de cada artista y el sonido de cada banda, pero por otro lado la capacidad de generalización, analizando si puede generar composiciones en el estilo de un artista, pero con el sonido de una banda que nunca haya tocado algo del mismo.

- **Entrenar con LoRa:** La técnica de LoRa (Low-Rank Adaptation) se probó efectiva para entrenar grandes modelos similares a SAO de manera eficiente, acelerando el reentrenamiento y reduciendo la cantidad de memoria utilizada, tanto para el proceso de reentrenamiento como para el almacenamiento del modelo final. Durante el desarrollo de esta tesis no existe ninguna implementación finalizada aplicable a este modelo, por lo tanto sería un gran aporte el desarrollo de la misma, o experimentación en el caso de que surja una.
- **Modificar el modelo:** En esta tesis nos centramos en la mejora de las capacidades musicales desde el lado de los datos, manteniendo cada aspecto del modelo por defecto. En este sentido también hay área para experimentar, modificando aspectos arquitectónicos del autoencoder o transformer de difusión, cambiando funciones de activación, modificando hiperparámetros, etc.
- **Refinar la métrica propuesta:** Comparar las representaciones de EnCodecMAE directamente, utilizando distancia coseno, no captó los aspectos de género, instrumentos y estilo general de los audios. Una posible forma de refinar esta métrica es entrenando un clasificador de género utilizando las representaciones de EnCodecMAE, y luego verificar que los audios generados sean clasificados en el género esperado.

BIBLIOGRAFíA

[1] D. Bogdanov et al. "ESSENTIA: an Audio Analysis Library for Music Information Retrieval". En: *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR'13)*. International Society for Music Information Retrieval, 2013, págs. 493-498. URL: http://hdl.handle.net/10230/32252.

[2] Alcedo Coenen. "David Cope, Experiments in Musical Intelligence. A-R Editions, Madison, Wisconsin, USA. Vol. 12 1996." En: *Organised Sound* 2.1 (1997), págs. 57-60. DOI: 10.1017/S1355771897210101.

[3] Jade Copet et al. *Simple and Controllable Music Generation*. 2023. arXiv: 2306.05284 [cs.SD].

[4] Prafulla Dhariwal y Alex Nichol. *Diffusion Models Beat GANs on Image Synthesis*. 2021. arXiv: 2105.05233 [cs.LG]. URL: https://arxiv.org/abs/2105.05233.

[5] Gustavo Diaz-Jerez. "Composing with Melomics: Delving into the Computational World for Musical Inspiration". En: *Leonardo Music Journal* 21 (dic. de 2011), págs. 13-14. ISSN: 0961-1215. DOI: 10.1162/LMJ_a_00053. eprint: https://direct.mit.edu/lmj/article-pdf/doi/10.1162/LMJ_a_00053/1675035/lmj_a_00053.pdf. URL: https://doi.org/10.1162/LMJ%5C_a%5C_00053.

[6] Hao-Wen Dong et al. *MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment*. 2017. arXiv: 1709.06298 [eess.AS]. URL: https://arxiv.org/abs/1709.06298.

[7] Douglas Eck y Jurgen Schmidhuber. "A First Look at Music Composition using LSTM Recurrent Neural Networks". En: (2002).

[8] Jesse Engel et al. *Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders*. 2017. arXiv: 1704.01279 [cs.LG]. URL: https://arxiv.org/abs/1704.01279.

[9] Zach Evans et al. *Fast Timing-Conditioned Latent Audio Diffusion*. 2024. arXiv: 2402.04825 [cs.SD]. URL: https://arxiv.org/abs/2402.04825.

[10] Zach Evans et al. *Long-form music generation with latent diffusion*. 2024. arXiv: 2404.10301 [cs.SD]. URL: https://arxiv.org/abs/2404.10301.

[11] Zach Evans et al. *Stable Audio Open*. 2024. arXiv: 2407.14358 [cs.SD]. URL: https://arxiv.org/abs/2407.14358.

[12] Seth* Forsgren y Hayk* Martiros. "Riffusion - Stable diffusion for real-time music generation". En: (2022). URL: https://riffusion.com/about.

[13] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML]. URL: https://arxiv.org/abs/1406.2661.

[14] F. Gouyon et al. "An experimental comparison of audio tempo induction algorithms". En: *IEEE Transactions on Audio, Speech, and Language Processing* 14.5 (2006), págs. 1832-1844. DOI: 10.1109/TSA.2005.858509.

[15] Jonathan Ho, Ajay Jain y Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG]. URL: https://arxiv.org/abs/2006.11239.

[16] Jonathan Ho y Tim Salimans. *Classifier-Free Diffusion Guidance*. 2022. arXiv: 2207.12598 [cs.LG]. URL: https://arxiv.org/abs/2207.12598.

[17] Cheng-Zhi Anna Huang et al. *Music Transformer*. 2018. arXiv: 1809.04281 [cs.LG]. URL: https://arxiv.org/abs/1809.04281.

[18] Khaled Koutini et al. "Efficient Training of Audio Transformers with Patchout". En: *Interspeech 2022*. interspeech2022. ISCA, sep. de 2022. DOI: 10.21437/interspeech.2022-227. URL: http://dx.doi.org/10.21437/Interspeech.2022-227.

[19] Aaron van den Oord et al. *WaveNet: A Generative Model for Raw Audio*. 2016. arXiv: 1609.03499 [cs.SD]. URL: https://arxiv.org/abs/1609.03499.

[20] Francois Pachet. "The Continuator: Musical Interaction With Style". En: *Journal of New Music Research* 32 (ago. de 2010), págs. 333-341. DOI: 10.1076/jnmr.32.3.333.16861.

[21] William Peebles y Saining Xie. *Scalable Diffusion Models with Transformers*. 2023. arXiv: 2212.09748 [cs.CV]. URL: https://arxiv.org/abs/2212.09748.

[22] Peter Sobot. *Pedalboard*. Jul. de 2021. DOI: 10.5281/zenodo.7817838. URL: https://doi.org/10.5281/zenodo.7817838.

[23] Jascha Sohl-Dickstein et al. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 2015. arXiv: 1503.03585 [cs.LG]. URL: https://arxiv.org/abs/1503.03585.

[24] Christian J. Steinmetz y Joshua D. Reiss. "pyloudnorm: A simple yet flexible loudness meter in Python". En: *150th AES Convention*. 2021.

[25] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[26] Peter Westergaard y Lejaren A. Hiller. En: *Journal of Music Theory* 3.2 (1959), págs. 302-306. ISSN: 00222909. URL: http://www.jstor.org/stable/842857 (visitado 15-11-2024).

[27] Yusong Wu et al. *Large-scale Contrastive Language-Audio Pretraining with Feature Fusion and Keyword-to-Caption Augmentation*. 2024. arXiv: 2211.06687 [cs.SD]. URL: https://arxiv.org/abs/2211.06687.

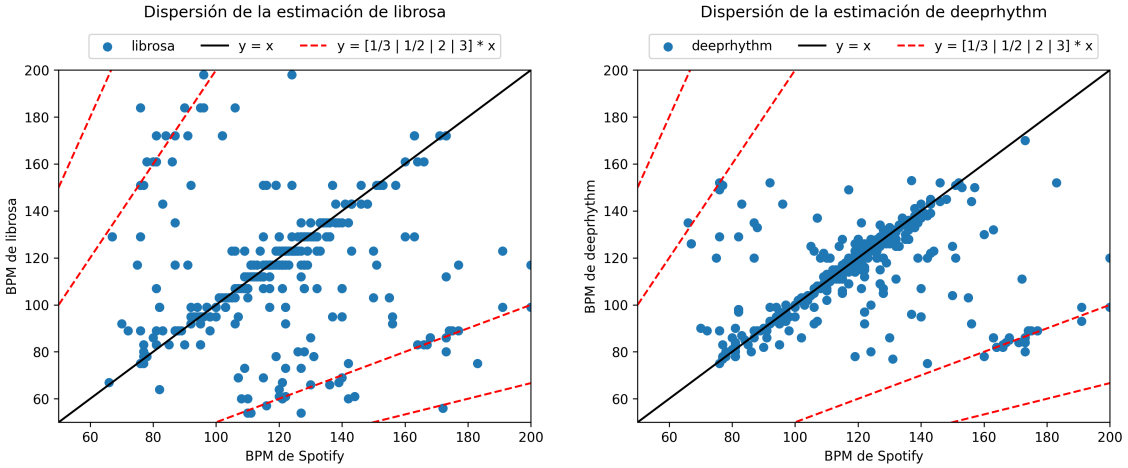
[28] R. Kh. Zaripov. "An algorithmic description of a process of musical composition". En: *Dokl. Akad. Nauk SSSR* 132 (6 1960), págs. 1283-1286. URL: http://mi.mathnet.ru/dan23732.

5. ANEXOS

5.1. Algoritmos de estimación de tempo

Se compararon dos algoritmos de estimación de tempo con enfoques distintos: el primero basado en programación dinámica, implementado por `librosa`, y el segundo, un modelo de redes convolucionales llamado `deeprhythm`.

Previo a este experimento, el conjunto de datos sintético ya había sido generado. Además, en los metadatos de cada audio contábamos con el tempo estimado por *Spotify* sobre la canción original, el cual se utilizó como referencia a la hora de comparar las estimaciones de ambos algoritmos.



(a) Dispersión de `librosa`.

(b) Dispersión de `deeprhythm`.

Fig. 5.1: Comparación de la dispersión del tempo estimado por `librosa` y `deeprhythm`, tomando como base el tempo obtenido de la API de *Spotify*.

En la Figura 5.1 se muestran los gráficos de dispersión de ambos algoritmos. En el caso de `librosa` (5.1a) se observan ciertos pasos o escalones, como líneas horizontales de puntos, lo que sugiere que las estimaciones tienden a redondearse. Este redondeo, lejos de ayudar, provocó una mayor dispersión de los puntos en comparación con `deeprhythm` (5.1b), donde los puntos se acercan más a la línea $x = y$.

Los gráficos de dispersión sugieren una mejora general en favor de `deeprhythm`, pero la Figura 5.2 lo termina de corroborar. En esta figura se grafica la distribución del error (tempo de *Spotify* - tempo estimado), y podemos observar que la distribución de `deeprhythm` es más densa en el 0, mientras que la de `librosa` tiene menos densidad en el 0 y más en los valores negativos. En el caso de `deepthythm` se ve una pequeña moda alrededor del 80, que son aquellos que se ven en la parte inferior derecha del gráfico de dispersión, que generalmente responden a predicciones de la mitad del tempo de *Spotify*, por ejemplo si era 160 y se predijo 80, el error es de -80.

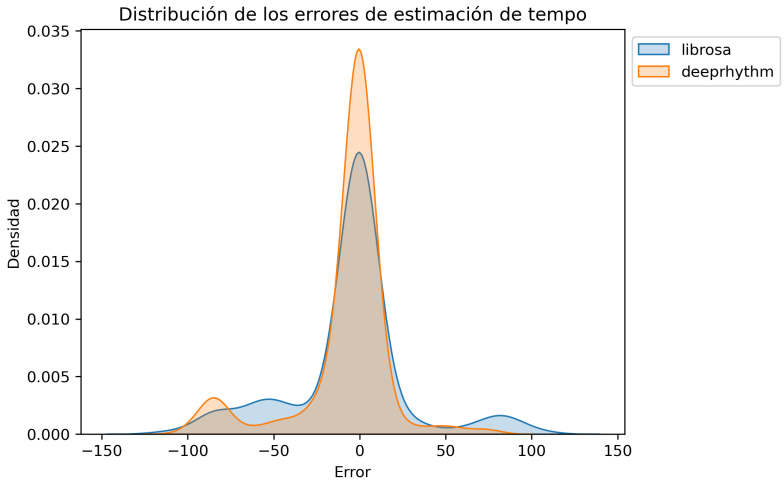


Fig. 5.2: Distribución de los errores de la estimación del tempo de `librosa` y `deepthythm`, tomando como base el tempo obtenido de la API de *Spotify*.