Generador de Letras y Melodías de Canciones

Harold Eustaquio

UNAM, FI Procesamiento del Lenguaje Natural 2025-1

21 de noviembre de 2024

Resumen

Este proyecto desarrolla un generador de letras y melodías musicales combinando procesamiento del lenguaje natural y aprendizaje profundo. Para la creación de letras, se utiliza la API LyricsGenius para analizar contenido emocional y ajustar el modelo `datificate/gpt2-small-spanish`, optimizado para español. En la generación de melodías, se emplea el dataset MAESTRO y un modelo LSTM, logrando secuencias coherentes y estilísticamente relevantes. Las evaluaciones mediante métricas como *RMSE* y *R2* identificaron configuraciones óptimas y áreas de mejora, destacando la necesidad de ajustar arquitecturas para capturar relaciones más complejas. Este trabajo contribuye al desarrollo de herramientas creativas escalables en el ámbito musical.

1. Introducción

El desarrollo de un generador de letras y melodías musicales responde a la creciente necesidad de herramientas creativas que permitan optimizar y democratizar los procesos de composición musical. En el ámbito de la música, la combinación de procesamiento del lenguaje natural (NLP) y aprendizaje profundo ha permitido avances significativos en la generación de contenido creativo, al tiempo que plantea desafíos relacionados con la coherencia, la diversidad léxica y la expresividad emocional.

Este proyecto aborda la creación de un sistema que no solo analiza las letras de canciones existentes para extraer sus características emocionales, sino que también genera nuevas composiciones líricas mediante el ajuste de modelos avanzados como el GPT-2 en español. Asimismo, incorpora un enfoque innovador para la generación de melodías utilizando datos musicales simbólicos procesados por arquitecturas basadas en LSTM, garantizando una salida estilísticamente relevante.

El presente trabajo evalúa los éxitos y limitaciones de las metodologías existentes, desarrollando estrategias para mejorar la calidad y la adaptabilidad de las letras y melodías generadas. Este enfoque integral busca contribuir al campo

del NLP y la inteligencia artificial aplicada a las artes, ofreciendo soluciones escalables y accesibles para creadores y productores musicales.

2. Metodología

2.1. Creación de Letra

2.1.1. Extracción y Análisis de Letras

`Extracción y Análisis de Letras` está diseñada para extraer, procesar y analizar el contenido emocional de letras de canciones. Utiliza la **API Lyrics-Genius** para recuperar letras, procesar el texto para limpiarlo y realizar análisis de sentimientos mediante un léxico predefinido. Los resultados se almacenan en formatos JSON y CSV para su posterior exploración y visualización.

Arquitectura

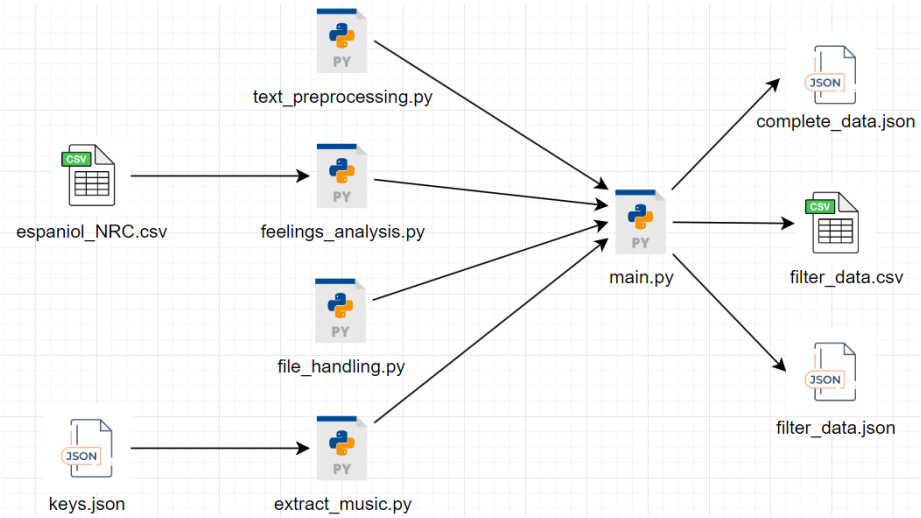

Figura 1: Arquitectura para Extracción y Análisis de Letras

Funciones

- `text_preprocessing.py`
  - Limpiar las letras eliminando elementos como metadatos, puntuación, números y palabras vacías (*stopwords*).
  - Normalizar el texto, convirtiéndolo a minúsculas y reemplazando caracteres acentuados.
  - Preparar las letras para procesos posteriores, como la tokenización y análisis emocional.

- `clean_text(lyrics)`: Limpia y preprocesa las letras de entrada, devolviendo una cadena normalizada.

- `feelings_analysis.py`
  - Este módulo analiza el contenido emocional del texto utilizando un léxico definido en `espaniol_NRC.csv`.
  - `feelings_in_text(text)`: Calcula las tres emociones más destacadas en el texto proporcionado.
- `file_handling.py`
  - Leer y escribir archivos JSON.
  - Guardar datos procesados en formato CSV.
  - `read_json(name)`: Lee un archivo JSON desde el directorio `data`.
  - `save_as_csv(data)`: Guarda datos en un archivo CSV (`filter_data.csv`).
- `extract_music.py`
  - Recupera letras utilizando la API **LyricsGenius**.
  - Limpia el texto con `text_processing`.
  - Analiza el contenido emocional utilizando `feelings_analysis.py`.
  - Guarda datos procesados en formatos JSON y CSV mediante `file_handling.py`.
  - `upload_song_data(artist_name, max_retries=5)`: Recupera y procesa datos de canciones de un artista específico.

Prerequisitos

- Python 3.8 o superior
- Dependencias: `lyricsgenius`, `pandas`.
- Archivos necesarios:
  - `data/keys.json`: Contiene el token de la API **LyricsGenius**.
  - `espaniol_NRC.csv`: Léxico para análisis emocional.

    Para mayor información, visitar Lyrics-Extraction (2024)

2.1.2. Generación de letra

La funcionalidad de **Generación de Letra** está diseñada para cargar, procesar y ajustar un modelo especializado en español. Este proyecto utiliza el modelo preentrenado **datificate/gpt2-small-spanish** de **Hugging Face**, optimizado para tareas de generación de texto en español.

Arquitectura

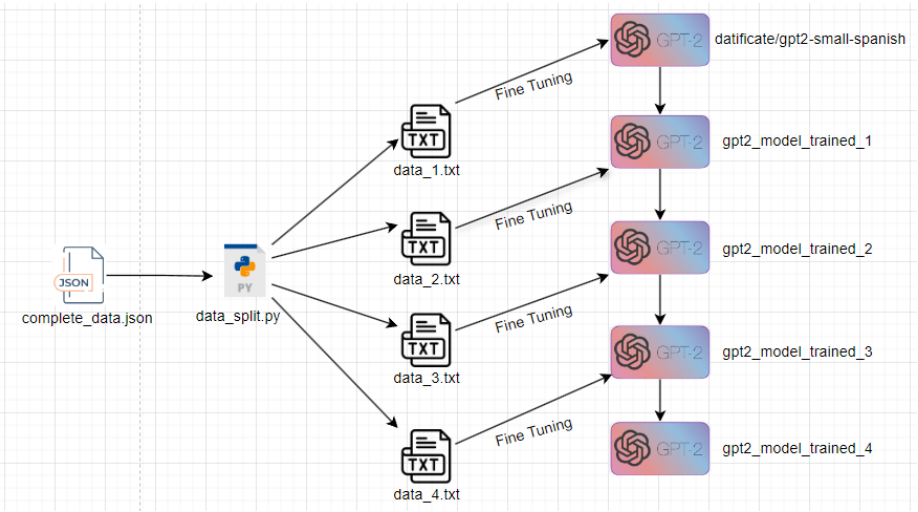

Figura 2: Arquitectura de generación de letra

Arquitectura del modelo GPT-2

El modelo GPT-2 se basa en una arquitectura de *Transformers*, introducida por Vaswani et al. (2017), diseñada para aprender representaciones complejas del lenguaje y generar texto de manera autorregresiva.

- **Capa de Embedding**: Convierte palabras o tokens en vectores numéricos de dimensión fija que representan semánticamente los textos de entrada.
- **Bloques de Transformador**:
  - **Mecanismo de Autoatención**: Identifica y prioriza relaciones clave entre palabras en una secuencia.
  - **Capas Feedforward**: Redes completamente conectadas que extraen características de alto nivel a partir de las representaciones generadas por la atención.
- **Máscara de Atención**: En GPT, la atención es *causal*, lo que asegura que el modelo considere únicamente palabras previas para predecir la siguiente palabra.

- **Capa de Decodificación**: Calcula la probabilidad de cada posible palabra basada en el contexto dado.

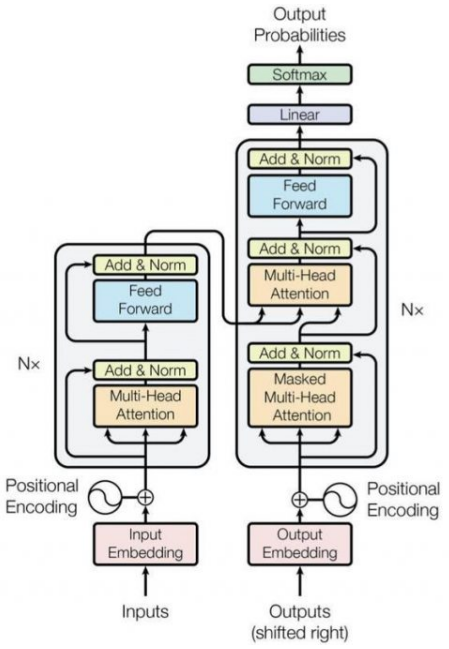

Figura 3: Arquitectura GPT

Modelo `datificate/gpt2-small-spanish`

El modelo `datificate/gpt2-small-spanish`, disponible en **Hugging Face**, es una versión ajustada de GPT-2, especializada en la generación de texto en español Obregón and Carrera (2020). Este modelo es ideal para tareas como la creación de letras de canciones debido a su entrenamiento en un corpus masivo de textos en español.

- **Características principales**:
  - Basado en la arquitectura GPT-2, diseñado para mantener coherencia y fluidez en los textos generados.
  - Modelo compacto con aproximadamente 124 millones de parámetros, equilibrando rendimiento y eficiencia computacional.
  - Amplio vocabulario de 50,257 tokens, permitiendo manejar una variedad de palabras y expresiones en español.
- **Razones para elegir este modelo**:

- Especialización en español, con preentrenamiento en datos exclusivamente en este idioma.
- Eficiencia computacional, adecuada para entornos con recursos limitados.
- Flexibilidad para adaptarse a tareas personalizadas como la generación de letras de canciones.

Preparación y división de datos

- **División de datos**: El script `data_split.py` divide el conjunto de datos recopilado en cuatro partes, facilitando el ajuste del modelo con diferentes subconjuntos.
- **Tokenización**: El tokenizador convierte el texto en una representación que el modelo puede procesar, asegurando que cada secuencia tenga la longitud adecuada.

Proceso de Fine-Tuning

- Configuración del entorno, deshabilitando integraciones innecesarias para simplificar el proceso.
- Definición de parámetros de entrenamiento, como:
  - **Épocas**: 3 iteraciones sobre el conjunto completo de datos.
  - **Tamaño del lote**: 4 muestras por actualización de parámetros.
  - **Tasa de aprendizaje**: 5×10^{-5} para controlar la velocidad de ajuste.
- Inicialización del `Trainer` y ejecución del proceso de entrenamiento.

Métricas de evaluación

- **Perplejidad**: Evalúa qué tan bien el modelo predice una secuencia de texto.
- **Diversidad Léxica**: Indica la riqueza de vocabulario en los textos generados.
- **Coherencia Local**: Analiza la relación semántica entre oraciones consecutivas (se usa `all-MiniLM-L6-v2` mostrado en Reimers and Gurevych (2019))

Para mayor información, visitar Fine-Tuning (2024) y Lyrics-Generation (2024)

2.2. Creación de Melodías

2.2.1. MAESTRO Dataset

El archivo comprimido `maestro-v3.0.0-midi.zip` contiene la versión 3.0.0 del conjunto de datos MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization). Este conjunto de datos está compuesto por aproximadamente 200 horas de interpretaciones virtuosas de piano, capturadas con una alineación precisa de aproximadamente 3 milisegundos entre las etiquetas de las notas y las formas de onda de audio.

Para más información sobre este dataset, visitar: Magenta (2024)

2.2.2. Generación de Melodías

La funcionalidad de `Generación de Melodías` está diseñada para cargar, procesar y ajustar un modelo especializado en datos musicales simbólicos. Este proyecto utiliza un modelo basado en una arquitectura LSTM, optimizado para tareas de predicción y generación de melodías polifónicas en formato simbólico.

Arquitectura

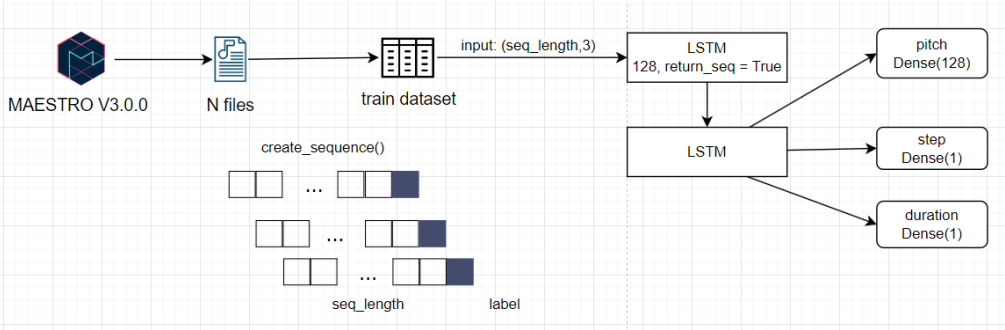

Figura 4: Arquitectura LSTM

Descargar dataset MAESTRO V3

- Se descargaron los datos del conjunto MAESTRO desde la fuente proporcionada y se guardaron en un directorio local.
  - Fuente: https://storage.googleapis.com/magentadata/datasets/maestro/v3.0.0/maestro-v3.0.0-midi.zip
- Una vez descargados, los archivos fueron extraídos automáticamente para facilitar su acceso. Posteriormente, se realizó una búsqueda en el directorio local para recopilar todos los archivos MIDI disponibles, escaneando recursivamente dentro de las carpetas del conjunto de datos.

- Finalmente, se contó el número total de archivos (1276) MIDI encontrados, asegurando que los datos estuvieran completos y listos para su posterior procesamiento.

Funciones para manejar MIDI

- `display_audio()`
  - La función genera una reproducción de audio a partir de un objeto MIDI procesado, utilizando una síntesis de sonido.
  - Convierte el contenido MIDI en una forma de onda de audio utilizando una frecuencia de muestreo definida. Luego, recorta el audio a una duración específica, predeterminada en 30 segundos.
  - Finalmente, devuelve el fragmento de audio generado para que pueda reproducirse directamente.
- `midi_to_notes()`
  - La función convierte un archivo MIDI en un conjunto estructurado de datos que describen las notas contenidas en él.
  - Primero, carga el archivo MIDI y selecciona el primer instrumento. Luego, organiza las notas en orden cronológico según su tiempo de inicio.
  - Para cada nota, extrae información como el tono (pitch), el tiempo de inicio, el tiempo de finalización, el intervalo entre el inicio de la nota actual y la anterior (step), y la duración de la nota.
  - Finalmente, almacena toda esta información en un DataFrame para facilitar su análisis y procesamiento.
- `notes_to_notes()`
  - La función crea un archivo MIDI a partir de un conjunto de datos que describen notas musicales y las guarda en un archivo de salida especificado.
  - Utiliza los datos proporcionados, como el tono (pitch), el intervalo entre notas (step), y la duración para construir notas individuales. Estas notas se asignan a un instrumento especificado por su nombre, con una intensidad predeterminada (velocity).
  - Las notas se agregan al instrumento en orden secuencial, manteniendo la progresión temporal.
  - Finalmente, el instrumento se añade a un objeto MIDI y se guarda como un archivo, permitiendo reproducir o reutilizar la melodía generada.

Dataset de Entrenamiento

- `Carga de archivos MIDI:` Se procesan un número limitado de archivos MIDI, en este caso, los primeros `N` archivos. Para cada archivo, se extraen las notas y sus características (como tono, intervalo entre notas y duración) utilizando la función `midi_to_notes`. Estos datos se almacenan en una lista.
- `Combinación de datos:` Todos los datos de las notas extraídas se combinan en un único DataFrame utilizando `pd.concat`, creando una colección consolidada de notas provenientes de los archivos MIDI.
- `Organización de las características:` Se define un orden específico para las características de las notas (`pitch`, `step`, `duration`) y se organiza la información en una matriz (`train_notes`) donde cada fila representa una nota y cada columna contiene una característica específica.
- `Creación de un dataset de TensorFlow:` Se convierte la matriz de datos en un dataset de TensorFlow (`notes_ds`), lo que permite su uso eficiente en modelos de aprendizaje automático.
- `create_sequence()`
  - `Ajuste de la longitud de las secuencias`: Se define la longitud de cada secuencia como `seq_length + 1`. Esto asegura que cada secuencia incluya tanto los datos de entrada como una etiqueta correspondiente.
  - `Creación de ventanas deslizantes`: El conjunto de datos se divide en ventanas consecutivas de longitud `seq_length`, con un desplazamiento de un elemento entre ventanas. Esto genera secuencias solapadas que preservan la progresión temporal de las notas.
  - `Agrupación en lotes de secuencias`: Los elementos de cada ventana se agrupan en lotes (`batch`) con longitud igual a `seq_length`, asegurando uniformidad en el tamaño de las secuencias.
  - `Normalización de las características`: Las características de las secuencias, como el tono (`pitch`), se normalizan dividiendo por valores específicos, como `vocab_size`, para escalar los datos a un rango relativo.
  - `Separación en entradas y etiquetas`: Cada secuencia se divide en:
    - `Entradas`: Todas las notas excepto la última en cada secuencia.
    - `Etiquetas`: La última nota de cada secuencia, organizada como un diccionario que asocia cada característica (*pitch*, *step*, *duration*) con su respectivo nombre clave.

    - `Mapeo de transformaciones`: Las transformaciones definidas se aplican a todas las secuencias, generando un conjunto de datos final que incluye las entradas normalizadas y las etiquetas, listas para el entrenamiento de un modelo.

- `Barajar las secuencias`: Se mezclan las secuencias generadas para evitar patrones repetitivos que puedan sesgar el aprendizaje del modelo.
- `Organización en lotes`: Las secuencias se agrupan en lotes de tamaño 64, asegurando que cada iteración del entrenamiento procese un conjunto uniforme de datos.
- `Optimización del flujo de datos`: Se utilizan técnicas de prefetching para cargar los datos en paralelo al entrenamiento, minimizando los tiempos de espera y acelerando el proceso.

Creación y Entrenamiento del Modelo

- `Función de pérdida personalizada`: Se define una función personalizada `mse_with_positive_pressure` para calcular la pérdida, que combina el error cuadrático medio (MSE) con una penalización adicional para valores negativos en las predicciones, incentivando valores positivos.
- `SparseCategoricalCrossentropy`: Se utiliza esta función de pérdida para la predicción de `pitch`, ya que es una variable categórica con 128 clases posibles (tonos MIDI). Esta función calcula la entropía cruzada entre las probabilidades predichas y las etiquetas verdaderas, penalizando más aquellas predicciones que asignan baja probabilidad a la clase correcta. Es eficiente porque trabaja directamente con etiquetas enteras sin necesidad de transformarlas a formato *one-hot*.
- `Configuración de los parámetros de entrada y aprendizaje`:
    - La forma de entrada tiene dimensiones (seq_length, 3), donde cada secuencia incluye 31 pasos y 3 características (pitch, step, duration).
    - La tasa de aprendizaje se establece en 0.005 para controlar la velocidad de ajuste del modelo durante el entrenamiento.
- `Estructura del modelo`:
    - Una capa de entrada recibe las secuencias musicales.
    - Dos capas LSTM (Long Short-Term Memory) procesan las secuencias para capturar relaciones temporales en los datos.
    - Tres capas densas independientes producen las salidas correspondientes a pitch, step y duration.
- `Compilación del modelo`: El modelo se compila utilizando el optimizador Adam, conocido por su capacidad para manejar tasas de aprendizaje adaptativas, junto con las funciones de pérdida definidas.

Predicción de Melodía

- `Inicialización de datos de entrada`
  - Se toma una muestra de notas (`sample_notes`) que se utilizarán para iniciar la generación.
  - Se normalizan las notas de entrada iniciales (`input_notes`) para ajustarse al formato esperado por el modelo.
- `predict_next_note()`: Utiliza un modelo entrenado para predecir la siguiente nota musical a partir de una secuencia de notas previa.
  - Expande la dimensión de entrada para ajustarla al formato requerido por el modelo.
  - Genera predicciones para las características de la próxima nota: pitch, step, y duration.
  - Ajusta los valores de pitch con base en la temperatura para añadir más variabilidad.
  - Muestra probabilísticamente el valor de pitch usando una distribución categórica.
  - Asegura que step y duration sean valores no negativos.
- `Configuración de parámetros para la generación`
  - `Temperatura:` Establecida en 2.0 para controlar la aleatoriedad de las predicciones, aumentando la variedad en las notas generadas.
  - `Número de notas a generar:` Fijado en 120, lo que indica cuántas notas se agregarán a la melodía.

Métricas de Evaluación

Para evaluar el rendimiento del modelo en la generación de melodías simbólicas, se utilizan dos métricas fundamentales: el **Error Cuadrático Medio de la Raíz (RMSE)** y el **Coeficiente de Determinación (R^2)**. Estas métricas son ampliamente reconocidas y han demostrado ser efectivas para evaluar modelos de aprendizaje profundo que trabajan con datos secuenciales y dependencias temporales. A continuación, se detallan las características y razones para su selección:

- **Error Cuadrático Medio de la Raíz (RMSE)**: El RMSE mide la magnitud promedio del error entre los valores predichos por el modelo y los valores reales. Es particularmente útil para evaluar características numéricas como la duración (`duration`) y los pasos (`step`). Se define como:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

donde y_i son los valores reales, $\hat{y}_i$ son los valores predichos y N es el número total de observaciones. Un RMSE bajo indica que las predicciones del modelo están cerca de los valores reales.

- **Coeficiente de Determinación (R^2)**: El R^2 evalúa qué proporción de la variabilidad en los datos reales es explicada por el modelo. Es especialmente útil para entender la relación global entre las predicciones y los valores reales. Se define como:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

donde $\bar{y}$ es la media de los valores reales. Un valor de R^2 cercano a 1 indica que el modelo captura eficazmente las relaciones subyacentes en los datos.

El uso de estas métricas está alineado con enfoques recientes en la literatura, como se detalla en el artículo "Hierarchical multi-head attention LSTM for polyphonic symbolic melody generation" Kasif et al. (2024). En este estudio, las métricas de RMSE y R^2 demostraron ser efectivas para evaluar modelos de generación de música simbólica al capturar tanto la precisión como la calidad de las predicciones en comparación con los métodos de referencia.

Proceso de Evaluación

1. **Selección de Datos de Prueba:** Se utilizan las primeras 31 (`seq_length`) notas de un archivo como entrada para el modelo, permitiéndole predecir las `N` notas siguientes.
2. **Comparación con Valores Reales:** Las notas predichas por el modelo se comparan con las notas reales del archivo original, seleccionando las posiciones correspondientes (`seq_length` hasta `N + seq_length`).
3. **Cálculo de Métricas:** Se calculan las métricas RMSE y R^2 para cada característica (`pitch`, `step` y `duration`), permitiendo un análisis detallado del desempeño del modelo.

Para mayor información, visitar Melody-Generation (2024)

3. Experimentos y resultados

3.1. Generación de letra

3.1.1. Fine-Tuning de modelo GPT-2

El proceso de *fine-tuning* del modelo `GPT-2` se realizó en cuatro etapas, utilizando diferentes subconjuntos del conjunto de datos. Cada iteración buscó optimizar la capacidad del modelo para generar texto coherente y relevante. Para evaluar el rendimiento durante el entrenamiento, se registró la pérdida de entrenamiento (*Training Loss*) en función de los pasos (*Steps*).

Step	Fine Tuning 1	Fine Tuning 2	Fine Tuning 3	Fine Tuning 4
1000	0.9643	0.3277	0.2280	0.3158
2000	0.2760	0.3317	0.2493	0.3419
3000	0.2174	0.2831	0.2051	0.2751
4000	0.2268	0.2922	0.2185	0.2997
5000	0.2008	0.2503	0.1882	0.2548
6000	0.2076	0.2852	0.2012	0.2830

Cuadro 1: Training Loss por Step para cada Fine-Tuning.

A continuación, se presenta una gráfica comparativa que ilustra la evolución del *Training Loss* para cada una de las etapas de *fine-tuning*.

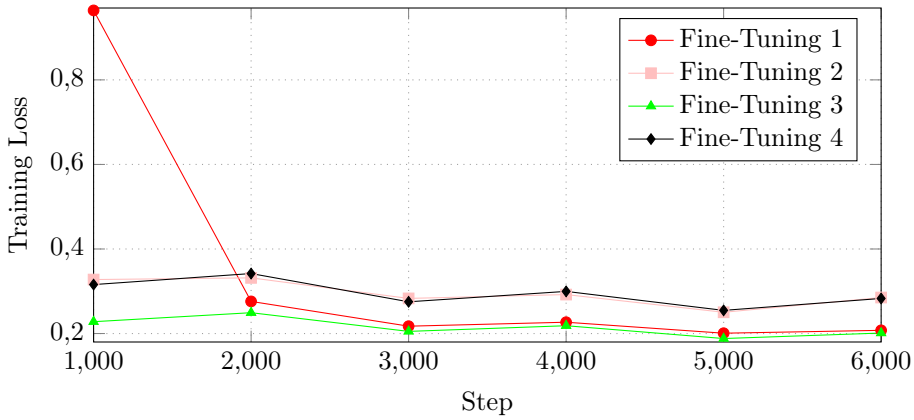

Figura 5: Training Loss por Step para cada Fine-Tuning

- La gráfica muestra que el proceso de `fine-tuning` logró una disminución consistente en la pérdida de entrenamiento en las cuatro etapas, lo que indica que el modelo se ajustó progresivamente a los datos.

- Como `fine-tuning_1` y `fine-tuning_3` tienen una pérdida más baja en comparación con otras, se puede inferir que esos subconjuntos de datos son más consistentes o adecuados para el entrenamiento.
- La pérdida inicial alta observada en `fine-tuning_1` sugiere que los datos iniciales presentaban una mayor complejidad o ruido, lo que dificultó el ajuste inicial del modelo. Sin embargo, la disminución significativa de la pérdida demuestra que el modelo fue capaz de adaptarse progresivamente, aprendiendo los patrones subyacentes en el conjunto de datos.

3.1.2. Evaluación de respuestas

Modelo	Perplexity	Distinct-2	Lexical Diversity	Local Coherence
Fine-Tuning 1	71.48	0.9900	0.9901	0.3658
Fine-Tuning 2	87.86	0.9791	0.9583	0.4930
Fine-Tuning 3	42.92	0.9865	0.9189	0.3888
Fine-Tuning 4	36.79	0.9881	0.9881	0.4463

Cuadro 2: Resultados de métricas para diferentes modelos de fine-tuning.

Perplexity

- `Perplexity` disminuye significativamente en `Fine Tuning 3` y `Fine Tuning 4`, indicando un mejor ajuste del modelo tras más iteraciones de fine-tuning.

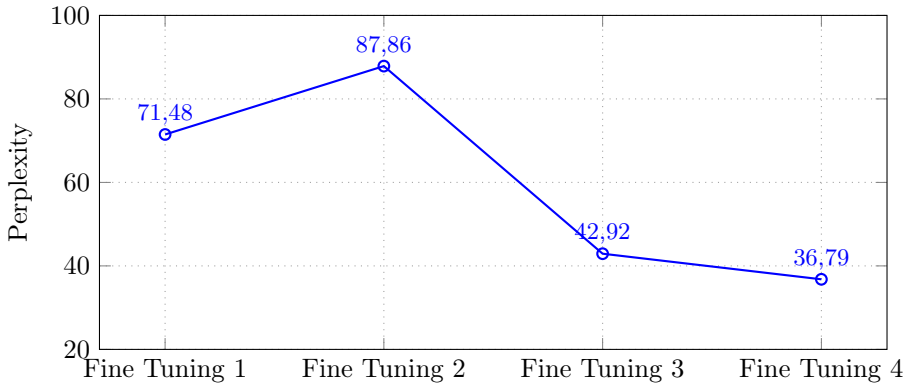

Figura 6: Perplexity para cada Fine-Tuning

Distinct-2

- La caída en el `Fine Tuning 2` podría indicar un ajuste que reduce la diversidad de bigramas generados.

- Los valores de los `Fine Tuning 3 y 4` reflejan una recuperación hacia la diversidad inicial, sugiriendo ajustes más adecuados para mantener el equilibrio entre calidad y diversidad.

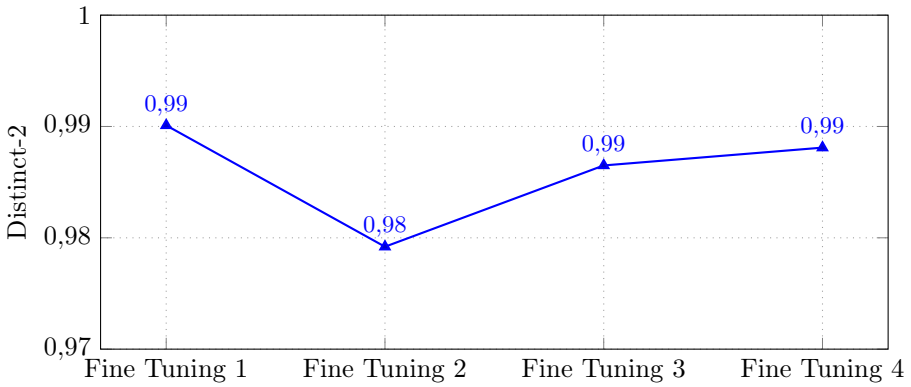

Figura 7: Distinct-2 para cada Fine-Tuning

Lexical Diversity

- Los `fine tuning 2 y 3` parecen comprometer la diversidad léxica, posiblemente debido a un ajuste de parámetros que prioriza otros aspectos.
- El `fine tuning 4` recupera la diversidad léxica, casi igualando al `fine tuning 1`, lo que sugiere una mejora en la calidad general de la generación.

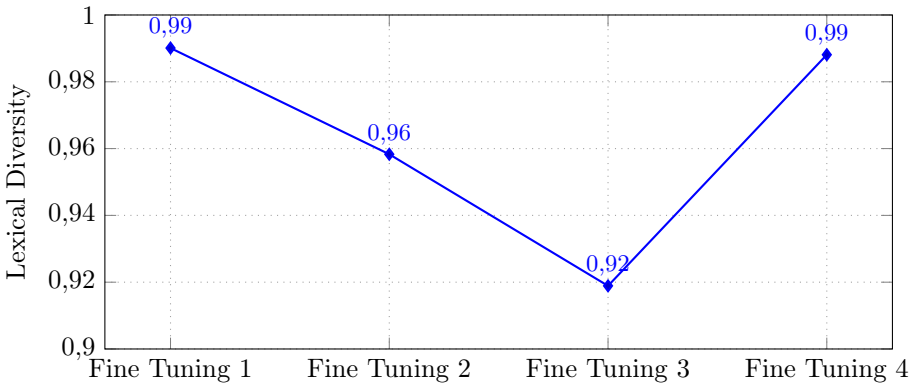

Figura 8: Lexical Diversity para cada Fine-Tuning

Local Coherence

- Los `fine tuning 1 y 3` tienen valores relativamente bajos, lo que podría indicar que las secuencias generadas tienen menos conexión lógica entre las partes del texto.
- El `fine tuning 2` muestra la mejor coherencia local, lo que sugiere que este ajuste prioriza la fluidez y conexión lógica de las ideas generadas.
- El `fine tuning 4` mantiene un equilibrio razonable, mejorando respecto al `fine tuning 1`, pero sin superar al `fine tuning 2`.

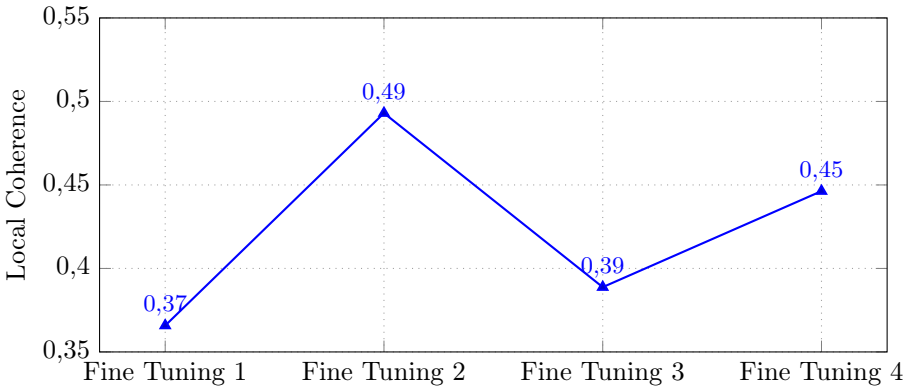

Figura 9: Local Coherence para cada Fine-Tuning

3.2. Generación de melodías

El proceso de generación de melodías se realizó en cuatro escenarios, cada uno entrenado con diferentes cantidades de archivos MIDI: 25, 50, 75 y 100. Cada escenario buscó optimizar la capacidad del modelo para predecir y generar secuencias melódicas coherentes y estilísticamente relevantes. Para evaluar el rendimiento durante el entrenamiento, se registró RMSE y R2 en función a la cantidad de secuencias de 250, 500, 750 y 1000 sonidos.

3.2.1. RMSE para pitch

Configuración	250 pred.	500 pred.	750 pred.	1000 pred.
melody_25	34.2786	33.8429	32.5962	32.9830
melody_50	28.4473	26.2595	25.3609	24.6178
melody_75	36.8140	34.7751	33.9925	33.8988
melody_100	33.1691	32.3058	30.2963	30.7059

Cuadro 3: RMSE de Pitch para Diferentes Configuraciones de Melodías

- La configuración `melody_50` presenta el RMSE más bajo en todas las etapas de predicción, alcanzando un valor mínimo en 1000 predicciones. Esto indica que esta configuración es la más efectiva en términos de precisión del tono.

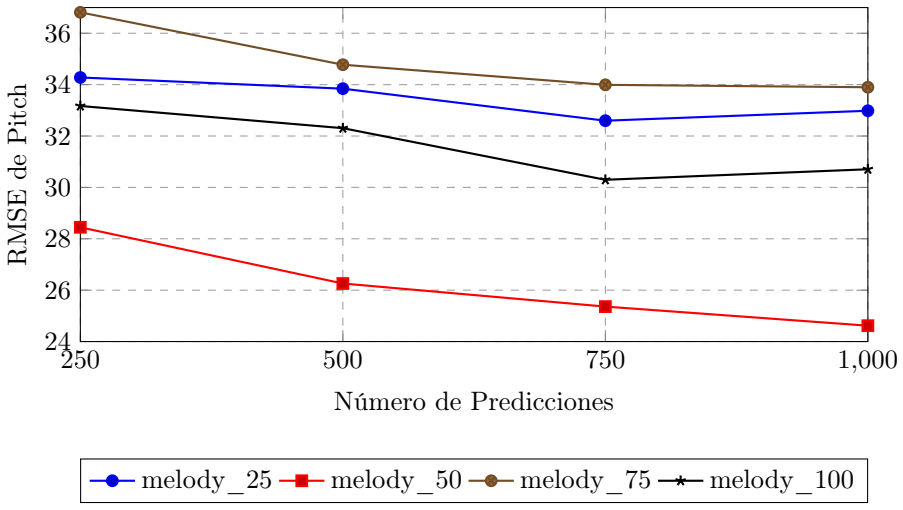

Figura 10: RMSE de Pitch

3.2.2. RMSE para step

Configuración	250 pred.	500 pred.	750 pred.	1000 pred.
melody_25	0.3436	0.2878	0.2576	0.2270
melody_50	0.4032	0.3371	0.2934	0.2769
melody_75	0.5189	0.4741	0.4743	0.4682
melody_100	0.7508	0.7233	0.7176	0.7243

Cuadro 4: Valores de RMSE de Step para Diferentes Melodías y Predicciones

- Las melodías más simples (`melody_25` y `melody_50`) muestran una disminución significativa del *RMSE* a medida que se incrementa el número de predicciones, destacando su mayor estabilidad y precisión en la representación temporal.
- La configuración `melody_25` presenta el mejor desempeño general, alcanzando un *RMSE* de 0.2270 tras 1000 predicciones, lo que la posiciona como la opción más eficiente para tareas que requieren alta precisión en las relaciones temporales simples.

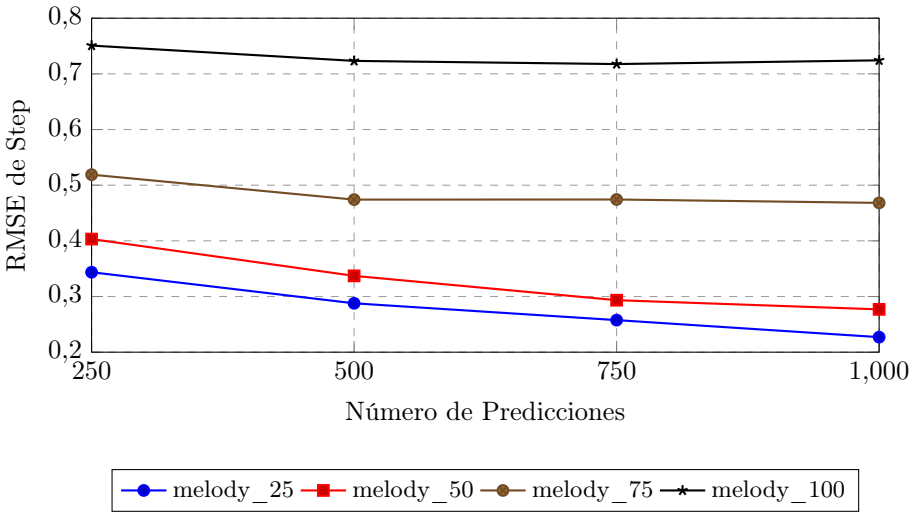

Figura 11: RMSE de Step

3.2.3. RMSE para duration

Configuración	250 pred.	500 pred.	750 pred.	1000 pred.
melody_25	0.4598	0.4199	0.3701	0.3198
melody_50	0.4851	0.4125	0.3328	0.3094
melody_75	0.4489	0.4040	0.3717	0.3758
melody_100	0.6477	0.6331	0.6412	0.6639

Cuadro 5: Valores de RMSE de Duration para Diferentes Melodías y Predicciones

- Las configuraciones `melody_25` y `melody_50` muestran una mejora continua en el *RMSE* para la duración a medida que aumenta el número de predicciones, alcanzando valores de 0.3198 y 0.3094 respectivamente en las 1000 predicciones. Esto indica que el modelo es más eficiente al manejar melodías simples con relaciones temporales menos complejas.
- En contraste, las configuraciones más complejas como `melody_100` mantienen altos valores de *RMSE*, incluso aumentando ligeramente con el número de predicciones (de 0.6477 a 0.6639). Esto sugiere que el modelo tiene dificultades para capturar patrones temporales más complejos en la característica *duration*, lo que podría resolverse mediante ajustes en el preprocesamiento de datos o la arquitectura del modelo.

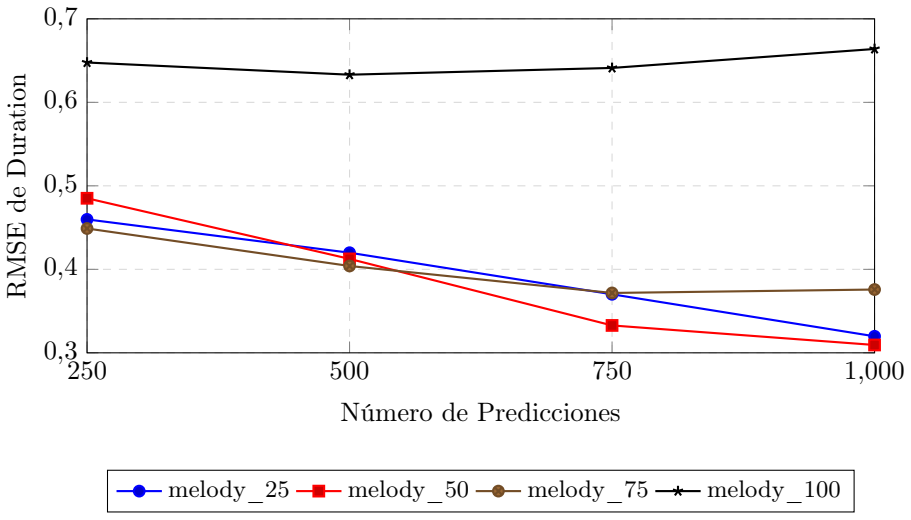

Figura 12: RMSE de Duration

3.2.4. R^2 para Pitch

Configuración	250 pred.	500 pred.	750 pred.	1000 pred.
melody_25	-6.7479	-8.1456	-7.4745	-8.0858
melody_50	-4.3361	-4.5062	-4.1299	-4.0616
melody_75	-7.9364	-8.6564	-8.2161	-8.5974
melody_100	-6.2545	-7.3337	-6.3209	-6.8746

Cuadro 6: Valores de R^2 de Pitch para Diferentes Melodías y Predicciones

- Los valores negativos de R^2 en todas las configuraciones reflejan un desempeño insuficiente del modelo al predecir la variabilidad en el *pitch*. La configuración `melody_50` presenta los mejores resultados relativos, con un valor de R^2 menos negativo (-4.0616 en 1000 predicciones), indicando que es la más adecuada dentro de las opciones evaluadas.
- Las configuraciones más complejas, como `melody_75` y `melody_100`, muestran una mayor dificultad para capturar las relaciones en los datos de tono, alcanzando valores consistentemente bajos de R^2. Esto evidencia la necesidad de revisar el modelo o las características de entrada para mejorar su capacidad predictiva en melodías más complejas.

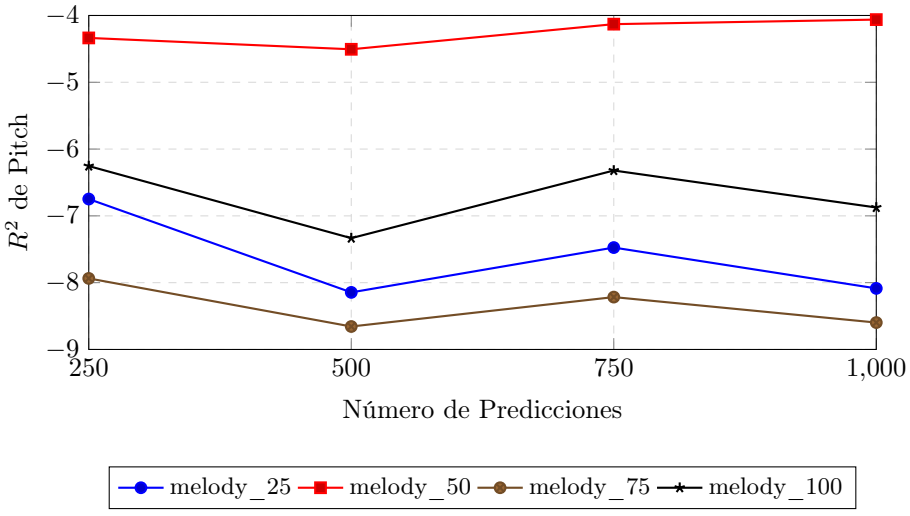

Figura 13: Valores de R^2 de Pitch para Diferentes Melodías y Predicciones

3.2.5. R^2 para Step

Configuración	250 pred.	500 pred.	750 pred.	1000 pred.
melody_25	-0.0804	-0.0961	-0.2800	-0.2772
melody_50	-0.4877	-0.5036	-0.6608	-0.9017
melody_75	-1.4642	-1.9743	-3.3405	-4.4354
melody_100	-4.1600	-5.9236	-8.9349	-12.0093

Cuadro 7: Valores de R^2 de Step para Diferentes Melodías y Predicciones

- La configuración `melody_25` presenta los valores menos negativos de R^2 en todas las predicciones, alcanzando un valor de -0.2772 en 1000 predicciones. Esto indica que el modelo es relativamente más efectivo en capturar las relaciones temporales en melodías simples, aunque sigue siendo inferior a un modelo base ideal.
- Las configuraciones más complejas, como `melody_75` y `melody_100`, muestran un deterioro significativo en los valores de R^2 conforme aumenta el número de predicciones, llegando a -4.4354 y -12.0093 respectivamente en 1000 predicciones. Esto evidencia que la arquitectura actual del modelo no puede manejar adecuadamente las relaciones temporales más complejas, destacando la necesidad de ajustar tanto los datos de entrada como el modelo para mejorar su rendimiento.

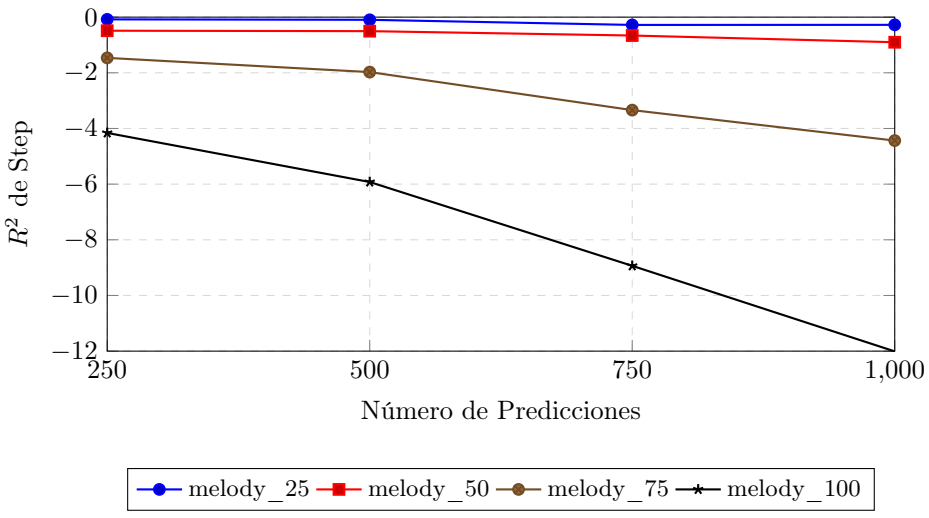

Figura 14: Valores de R^2 de Step para Diferentes Melodías y Predicciones

3.2.6. R^2 para duration

Configuración	250 pred.	500 pred.	750 pred.	1000 pred.
melody_25	-0.3366	-0.4766	-0.5643	-0.4404
melody_50	-0.4873	-0.4246	-0.2645	-0.3484
melody_75	-0.2735	-0.3666	-0.5773	-0.9899
melody_100	-1.6515	-2.3566	-3.6942	-5.2099

Cuadro 8: Valores de R^2 de Duration para Diferentes Melodías y Predicciones

- Las configuraciones más simples, `melody_25` y `melody_50`, muestran valores de R^2 menos negativos en comparación con las configuraciones más complejas. Por ejemplo, `melody_25` alcanza -0.4404 y `melody_50` llega a -0.3484 en 1000 predicciones, lo que indica un mejor desempeño relativo del modelo en melodías con menor complejidad temporal.
- Las configuraciones más complejas, como `melody_100`, presentan un deterioro significativo en los valores de R^2, alcanzando -5.2099 en 1000 predicciones. Esto resalta la incapacidad del modelo para manejar relaciones temporales complejas en `duration`, subrayando la necesidad de ajustes en la arquitectura del modelo y técnicas avanzadas de preprocesamiento de datos para mejorar el rendimiento.

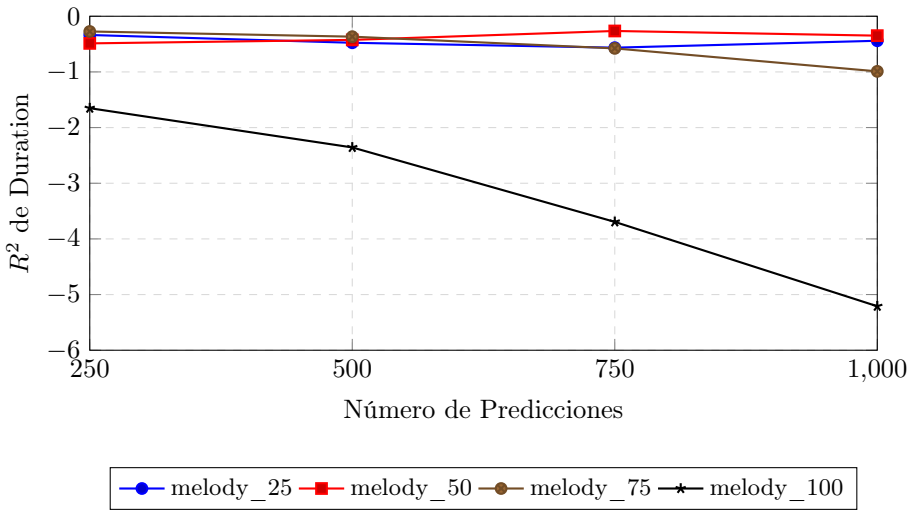

Figura 15: Valores de R^2 de Duration para Diferentes Melodías y Predicciones

4. Conclusiones

En este trabajo se desarrolló un generador de letras y melodías de canciones combinando técnicas de procesamiento del lenguaje natural (NLP) y aprendizaje profundo. A partir de los resultados obtenidos, se destacan las siguientes conclusiones:

- La utilización del modelo `GPT-2-small-spanish` permitió generar letras de canciones coherentes y estilísticamente variadas. El ajuste fino (*fine-tuning*) en diferentes subconjuntos de datos mejoró métricas como la *Perplexity* y la diversidad léxica, alcanzando un equilibrio entre coherencia local y diversidad en las iteraciones finales.
- El análisis emocional de letras existentes, basado en un léxico predefinido, proporcionó información valiosa para entender y replicar patrones emocionales en las composiciones generadas.
- Para la generación de melodías, el modelo basado en *LSTM* mostró un desempeño aceptable en melodías simples, aunque los valores negativos de R^2 para características como *pitch* y *duration* en melodías más complejas reflejan limitaciones en la arquitectura y la necesidad de explorar enfoques más avanzados.
- El uso del conjunto de datos MAESTRO para la extracción y análisis de archivos MIDI permitió desarrollar una representación estructurada de las características musicales, facilitando el preprocesamiento y entrenamiento del modelo.
- Las métricas de evaluación utilizadas, como RMSE y R^2, demostraron ser efectivas para medir el rendimiento del modelo en la predicción de características musicales simbólicas, aunque se identificaron oportunidades para mejorar la generalización del sistema.s

Referencias

Fine-Tuning (2024), 'Melody-lyric-generator: Fine-tuning gpt', https://github.com/haroldeustaquio/Melody-Lyric-Generator/tree/main/Lyrics-Generation/Fine-Tuning-GPT.

Kasif, A., Sevgen, S., Ozcan, A. and Catal, C. (2024), 'Hierarchical multi-head attention lstm for polyphonic symbolic melody generation', *Multimedia Tools and Applications* .
URL: *https://doi.org/10.1007/s11042-024-18491-7*

Lyrics-Extraction (2024), 'Melody-lyric-generator: Lyrics extraction', https://github.com/haroldeustaquio/Melody-Lyric-Generator/tree/main/Lyrics-Generation/Lyrics-Extraction.

Lyrics-Generation (2024), 'Melody-lyric-generator: Lyrics generation', https://github.com/haroldeustaquio/Melody-Lyric-Generator/tree/main/Lyrics-Generation.

Magenta (2024), 'Maestro dataset'. Accessed: 2024-11-20.
URL: *https://magenta.tensorflow.org/datasets/maestro*

Melody-Generation (2024), 'Melody-lyric-generator: Melody generation', https://github.com/haroldeustaquio/Melody-Lyric-Generator/tree/main/Melody-Generation.

Obregón, J. and Carrera, B. (2020), 'Gpt2-small-spanish: Un modelo de lenguaje para generación de texto en español'. Accedido: 19 de noviembre de 2024.
URL: *https://huggingface.co/datificate/gpt2-small-spanish*

Reimers, N. and Gurevych, I. (2019), 'Sentence-bert: Sentence embeddings using siamese bert-networks'.
URL: *https://arxiv.org/abs/1908.10084*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, and Polosukhin, I. (2017), Attention is all you need, *in* 'Advances in Neural Information Processing Systems', Vol. 30.